

# Reliable Learning and Control in Dynamic Environments: Towards Unified Theory and Learned Robotic Agility

Thesis by  
Guanya Shi

In Partial Fulfillment of the Requirements for the  
Degree of  
Doctor of Philosophy

The logo for the California Institute of Technology (Caltech), featuring the word "Caltech" in a bold, orange, sans-serif font.

CALIFORNIA INSTITUTE OF TECHNOLOGY  
Pasadena, California

2023  
Defended July 21, 2022

© 2023

Guanya Shi

ORCID: 0000-0002-9075-3705

All rights reserved

## ACKNOWLEDGEMENTS

To my **advisors**, Soon-Jo Chung and Yisong Yue. It is extremely fortunate to simultaneously have two amazing advisors. Thank you for your guidance, patience, kindness, and encouragement. This dissertation would not have been possible without your support. I also have so many fond memories of you outside of research. For example, the wonderful group event at Soon-Jo's house and my defense against Yisong when we played basketball at Grant Park.

To other **mentors and collaborators** at Caltech: Adam Wierman, Anima Anandkumar, Kamyar Azizzadenesheli, Steven Low, Joel Burdick, Guannan Qu, and Rose Yu, outside of Caltech: Jean-Jacques Slotine, Byron Boots, Na Li, and Chuchu Fan, and at NVIDIA during my internship: Yuke Zhu, Stan Birchfield, Fabio Ramos, and Jonathan Tremblay. Thank you for your invaluable insights.

To the **ARCL Lab** led by Prof. Soon-Jo Chung: Benjamin (Ben) Riviere, Michael (Mike) O'Connell, Kai Matsuka, Hiroyasu (Hiro) Tsukamoto, Ellande Tang, Soo-Jean Han, Xichen Shi, Wolfgang Hönig, Connor Lee, Sorina Lupu, Kyunam Kim, Jimmy Ragan, Nikhil Ranganathan, John Lathrop, Fengze Xie, Yuta Takahashi, Yashwanth Kumar Nakka, Karena Cai, Rebecca Foust, Vincenzo Capuano, Patrick Spieler, Alexei Harvard, Matt Anderson, Lu Gan, and James Preiss. Thank you for all of the wonderful memories.

To the **Yue Crew** led by Prof. Yisong Yue: Jeremy Bernstein, Geeling Chau, Victor Dorobantu, Alex Farhang, Yujia Huang, Uriah Israel, Ivan Jimenez Rodriguez, Francesca-Zhoufan Li, Kejun (Amy) Li, Yiheng Lin, Hao Liu, Jennifer Sun, Sabera Talukder, Cameron Voloshin, Xuefei (Julie) Wang, Jason Yang, Chris Yeh, Yuxin Chen, Hoang Le, Angie Liu, Joe Marino, Ellen Novoseller, Ugo Rosolia, Jialin Song, Yanan Sui, Eric Zhan, and Stephan Zheng. Thank you for all of the wonderful memories.

To my undergraduate **mentees** at Caltech: Yiheng Lin, Chenkai Yu, Weici Pan, Yang Hu, Ruixiao Yang, Anya Vinogradsky, Alice Jin, Luis Pabon Madrid, and Nelson Badillo. Mentoring and teaching are always bidirectional for me. I've learned a lot from all of you.

To all of the other **friends** and my Caltech **cohort**: Yuanyuan Shi, Tongxin Li, Sara Beery, Elizabeth Qian, Gautam Goel, Forte Shinko, Sahin Lale, Xiaomin Li, Zongyi Li, Navid Azizan, Jiawei Zhao, Shumao Zhang, Nian Guo, Chen Li, Kun

Miao, Xin Su, Botao Hu, Fangzhu Zhao, Yukai Liu, Yalu Chen, Zhiquan Yuan, and many others. Thank you for making my Ph.D. journey memorable. In particular, I want to acknowledge our Wed and Fri basketball group. It is so much fun to play basketball twice a week.

To the **CMS** department at Caltech, especially the **CDS** option. The department has offered me a fantastic environment for explorations and the access to world-class faculty members, challenging but super interesting courses, and top-notch experimental facilities such as the Real Weather Wind Tunnel. In particular, I would like to acknowledge Richard Murray, Steve Brunton, and Katie Bouman for their help when I was on the academic job market. I also really appreciate taking excellent courses from Venkat Chandrasekaran, Richard Murray, Joel Burdick, John Doyle, Houman Owhadi, Andrew Stuart, and Kostia Zuev.

To my **family**, especially my mom Chunling and my dad Yusheng. Thank you for your love. You always support me no matter what I decide to pursue, from high school, undergraduate studies at Tsinghua, to my Ph.D. at Caltech.

Finally, to **funding agencies**: NSF, DARPA, Raytheon, and Caltech CAST, and **prize donors**: the Simoudis family for the Simoudis Discovery Prize and the Chou family for the Ben P.C. Chou Doctoral Prize. Thank you so much for your support on me and my research.



## ABSTRACT

Recent breathtaking advances in machine learning beckon to their applications in a wide range of real-world autonomous systems. However, for safety-critical settings such as agile robotic control in hazardous environments, we must confront several key challenges before widespread deployment. Most importantly, the learning system must interact with the rest of the autonomous system (e.g., highly nonlinear and non-stationary dynamics) in a way that safeguards against catastrophic failures with formal guarantees. In addition, from both computational and statistical standpoints, the learning system must incorporate prior knowledge for efficiency and generalizability.

This thesis presents progress towards establishing a unified framework that fundamentally connects learning and control. First, Part I motivates the benefit and necessity of such a unified framework by the *Neural-Control Family*, a family of nonlinear deep-learning-based control methods with not only stability and robustness guarantees but also new capabilities in agile robotic control. Then Part II discusses three unifying interfaces between learning and control: (1) online meta-adaptive control, (2) competitive online optimization and control, and (3) online learning perspectives on model predictive control. All interfaces yield settings that jointly admit both learning-theoretic and control-theoretic guarantees.

## PUBLISHED CONTENT AND CONTRIBUTIONS

Li, Tongxin, Ruixiao Yang, Guannan Qu, Guanya Shi, Chenkai Yu, Adam Wierman, and Steven Low (2022). *Robustness and consistency in linear quadratic control with untrusted predictions*. In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 6.1, pp. 1–35. DOI: [10.1145/3508038](https://doi.org/10.1145/3508038).

G.S. contributed to the algorithm design and theoretical analysis.

O’Connell, Michael, Guanya Shi, Xichen Shi, Kamyar Azizzadenesheli, Anima Anandkumar, Yisong Yue, and Soon-Jo Chung (2022). *Neural-Fly enables rapid learning for agile flight in strong winds*. In: *Science Robotics* 7.66, eabm6597. DOI: [10.1126/scirobotics.abm6597](https://doi.org/10.1126/scirobotics.abm6597).

The first two authors contributed equally and the order is alphabetic. G.S. co-led the project, designed the learning algorithm and co-designed the adaptive control method with theoretical analysis, and co-designed and co-conducted robotic experiments.

Pan, Weici, Guanya Shi, Yiheng Lin, and Adam Wierman (2022). *Online optimization with feedback delay and nonlinear switching cost*. In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 6.1, pp. 1–34. DOI: [10.1145/3508037](https://doi.org/10.1145/3508037).

G.S. designed the project and defined the research question, contributed to the algorithm design and proofs, and mentored W.P. in numerical experiments.

Shi, Guanya, Wolfgang Hönig, Xichen Shi, Yisong Yue, and Soon-Jo Chung (2022). *Neural-Swarm2: Planning and control of heterogeneous multirotor swarms using learned interactions*. In: *IEEE Transactions on Robotics* 38.2, pp. 1063–1079. DOI: [10.1109/TRO.2021.3098436](https://doi.org/10.1109/TRO.2021.3098436).

G.S. led the project, designed the learning, motion planning, and control algorithms with theoretical analysis, and designed and co-conducted the numerical and robotic experiments.

Yu, Chenkai, Guanya Shi, Soon-Jo Chung, Yisong Yue, and Adam Wierman (2022). *Competitive control with delayed imperfect information*. In: *American Control Conference (ACC)*. URL: <https://arxiv.org/abs/2010.11637>.

G.S. designed the project and defined the research problem, designed the predictive control algorithm and contributed to all proofs, and mentored C.Y. in numerical experiments.

Lin, Yiheng, Yang Hu, Guanya Shi, Haoyuan Sun, Guannan Qu, and Adam Wierman (2021). *Perturbation-based regret analysis of predictive control in linear time varying systems*. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 34. Curran Associates, Inc., pp. 5174–5185. URL: <https://arxiv.org/abs/2106.10497>.

First five authors contributed equally. G.S. co-designed the theoretical analysis framework and contributed to all proofs.

Nakka, Yashwanth Kumar, Anqi Liu, Guanya Shi, Anima Anandkumar, Yisong Yue, and Soon-Jo Chung (2021). *Chance-constrained trajectory optimization for safe exploration and learning of nonlinear Systems*. In: *IEEE Robotics and Automation Letters* 6.2, pp. 389–396. DOI: 10.1109/LRA.2020.3044033.

G.S. contributed to the algorithm design and theoretical analysis, and helped Y.K.N. with the numerical experiments.

Shi, Guanya, Kamyar Azizzadenesheli, Michael O’Connell, Soon-Jo Chung, and Yisong Yue (2021). *Meta-adaptive nonlinear control: Theory and algorithms*. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 34. Curran Associates, Inc., pp. 10013–10025. URL: <https://proceedings.neurips.cc/paper/2021/file/52fc2aee802efbad698503d28ebd3a1f-Paper.pdf>.

G.S. led the project, designed the online meta-adaptive control algorithm with theoretical analysis, and designed and co-conducted numerical experiments.

Shi, Guanya, Yifeng Zhu, Jonathan Tremblay, Stan Birchfield, Fabio Ramos, Animeshree Anandkumar, and Yuke Zhu (2021). *Fast uncertainty quantification for deep object pose estimation*. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5200–5207. DOI: 10.1109/ICRA48506.2021.9561483.

G.S. led the project, designed the uncertainty quantification algorithm, conducted numerical experiments, and co-conducted real-world manipulator experiments.

Liu, Anqi, Guanya Shi, Soon-Jo Chung, Anima Anandkumar, and Yisong Yue (2020). *Robust regression for safe exploration in control*. In: *Learning for Dynamics and Control*. PMLR, pp. 608–619. URL: <https://proceedings.mlr.press/v120/liu20a.html>.

G.S. co-designed the safe exploration algorithm and contributed to the numerical experiments.

Shi, Guanya, Wolfgang Hönig, Yisong Yue, and Soon-Jo Chung (2020). *Neural-Swarm: Decentralized close-proximity multirotor control using learned interactions*. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3241–3247. DOI: 10.1109/ICRA40945.2020.9196800.

G.S. led the project, designed the learning, motion planning, and control algorithms with theoretical analysis, and designed and co-conducted the numerical and robotic experiments.

Shi, Guanya, Yiheng Lin, Soon-Jo Chung, Yisong Yue, and Adam Wierman (2020). *Online optimization with memory and competitive control*. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 33. Curran Associates, Inc. URL: <https://arxiv.org/abs/2002.05318>.

G.S. led the project, defined the research problem, and co-designed the core algorithm with theoretical analysis.

Yu, Chenkai, Guanya Shi, Soon-Jo Chung, Yisong Yue, and Adam Wierman (2020). *The power of predictions in online control*. In: *Advances in Neural Informa-*

*tion Processing Systems (NeurIPS)*. Vol. 33. Curran Associates, Inc., pp. 1994–2004. URL: <https://proceedings.neurips.cc/paper/2020/file/155fa09596c7e18e50b58eb7e0c6ccb4-Paper.pdf>.

G.S. designed the project and defined the research problem, contributed to all proofs, and mentored C.Y. in numerical experiments.

Shi, Guanya, Xichen Shi, Michael O’Connell, Rose Yu, Kamyar Azizzadenesheli, Animashree Anandkumar, Yisong Yue, and Soon-Jo Chung (2019). *Neural Lander: Stable drone landing control using learned dynamics*. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 9784–9790. DOI: 10.1109/ICRA.2019.8794351.

G.S. led the project, designed the learning algorithm with theoretical analysis, co-designed the controller, and co-conducted the robotic experiments.

## TABLE OF CONTENTS

Acknowledgements . . . . .	iii
Abstract . . . . .	v
Published Content and Contributions . . . . .	vi
Table of Contents . . . . .	viii
List of Illustrations . . . . .	xi
List of Tables . . . . .	xvii
Chapter I: Introduction . . . . .	1
1.1 Thesis Organization . . . . .	3
1.2 Thesis Summary and Contributions . . . . .	4
<b>I Neural-Control Family: Deep-Learning-Based Nonlinear Control with Learned Robotic Agility</b>	<b>9</b>
Chapter II: Overview . . . . .	11
2.1 A Mixed Model for Robot Dynamics and the Nonlinear Control Law	11
2.2 Example: Quadrotor Dynamics and Control . . . . .	14
2.3 Organization of Part I: Unifying Learning and Nonlinear Control . . .	18
2.4 Preliminaries on Deep Learning . . . . .	20
2.5 Preliminaries on Nonlinear Control Theory . . . . .	23
Chapter III: Neural-Lander . . . . .	26
3.1 Introduction and Related Work . . . . .	27
3.2 Problem Statement . . . . .	29
3.3 Dynamics Learning and Controller Design . . . . .	30
3.4 Nonlinear Stability Analysis . . . . .	30
3.5 Experiments . . . . .	34
Chapter IV: Neural-Swarm . . . . .	43
4.1 Introduction . . . . .	44
4.2 Related Work . . . . .	47
4.3 Problem Statement . . . . .	48
4.4 Learning of Swarm Aerodynamic Interaction . . . . .	51
4.5 Interaction-Aware Multi-Robot Planning . . . . .	57
4.6 Interaction-Aware Tracking Controller . . . . .	63
4.7 Experiments . . . . .	64
Chapter V: Neural-Fly . . . . .	79
5.1 Introduction . . . . .	80
5.2 Related Work . . . . .	83
5.3 Problem Statement . . . . .	88
5.4 Offline Meta-Learning . . . . .	89
5.5 Online Adaptive Control and Stability Analysis . . . . .	94

5.6 Experiments . . . . .	99
5.7 Appendix . . . . .	109
Chapter VI: Safe Exploration . . . . .	117
6.1 Introduction and Related Work . . . . .	118
6.2 Problem Statement . . . . .	120
6.3 Uncertainty Quantification under Domain Shift . . . . .	124
6.4 Safe Planning and Control . . . . .	126
6.5 Simulations . . . . .	127
Chapter VII: Discussion and Future Work . . . . .	132
7.1 Neural-Control for Other Robotic Systems . . . . .	132
7.2 End-to-End Learning and Control Guarantees . . . . .	135
7.3 Model-Free and Model-Based Learning . . . . .	135
7.4 Neurosymbolic Learning for Data-Efficient Decision Making . . . . .	136
7.5 Other Residual Learning Methods . . . . .	137
<b>II Unifying Interfaces Between Learning and Control Theory</b>	<b>139</b>
Chapter VIII: Overview . . . . .	141
8.1 The Importance of Building Interfaces Between Learning and Control	141
8.2 Organization of Part II: Three Interfaces . . . . .	143
8.3 Preliminaries on Online Optimization and Learning . . . . .	144
8.4 Preliminaries on Online Optimal Control . . . . .	147
Chapter IX: Online Meta-Adaptive Control . . . . .	154
9.1 Introduction . . . . .	155
9.2 Problem Statement . . . . .	156
9.3 Online Meta-Adaptive Control (OMAC) Algorithm . . . . .	159
9.4 Different Instantiations of OMAC and Theoretical Analysis . . . . .	162
9.5 Simulations . . . . .	171
9.6 Related Work . . . . .	174
Chapter X: Competitive Online Optimization and Control . . . . .	180
10.1 Introduction . . . . .	180
10.2 Problem Statement: OCO with Structured Memory . . . . .	182
10.3 Algorithms and Theoretical Analysis . . . . .	185
10.4 Connections to Competitive Control . . . . .	189
10.5 Simulations . . . . .	195
10.6 Extension: Nonlinear Switching Cost and Feedback Delay . . . . .	196
10.7 Appendix . . . . .	201
Chapter XI: Online Learning Perspectives on Model Predictive Control . . . . .	208
11.1 Introduction . . . . .	209
11.2 Problem Statement . . . . .	211
11.3 Dynamic Regret of MPC in LQ Systems . . . . .	216
11.4 Simulations . . . . .	234
11.5 Extension I: Delayed Inexact Predictions . . . . .	235
11.6 Extension II: Time-Variant Systems and General Costs . . . . .	240
Chapter XII: Discussion and Future Work . . . . .	246

## LIST OF ILLUSTRATIONS

<i>Number</i>	<i>Page</i>
3.1 Neural-Lander enables agile flight maneuver very close to the ground.	26
3.2 Training data trajectory. Part I (0 to 250 s) contains maneuvers at different heights (0.05 m to 1.50 m). Part II (250 s to 350 s) includes random $x$ , $y$ , and $z$ motions for maximum state-space coverage. . . .	35
3.3 DNN prediction performance. (a) Learned $f_{a,z}$ compared to the ground effect model with respect to height $z$ , with $v_z = v_x = v_y = 0$ m/s, $R = I$ , $u = 6400$ RPM. Ground truth points are from hovering data at different heights. (b) Learned $f_{a,z}$ with respect to rotation speed $n$ ( $z = 0.2$ m, $v_z = 0$ m/s), compared to $C_T$ measured in the bench test. (c) Heatmaps of learned $f_{a,z}$ versus $z$ and $v_z$ . (Left) ReLU network with spectral normalization. (Right) ReLU network without spectral normalization. . . . .	36
3.4 Baseline Controller and <i>Neural-Lander</i> performance in take-off and landing. Means (solid curves) and standard deviations (shaded areas) of 10 trajectories. . . . .	37
3.5 <i>Neural-Lander</i> performance in take-off and landing with different DNN capacities. 1 layer means a linear mapping $\hat{f}_a = Ax + b$ ; 0 layer means $\hat{f}_a = b$ ; Baseline means $\hat{f}_a \equiv 0$ . . . . .	38
3.6 Heatmaps of learned $\hat{f}_{a,z}$ and trajectory tracking performance. (a) Heatmaps of learned $\hat{f}_{a,z}$ versus $x$ and $y$ , with other inputs fixed. (Left) ReLU network with spectral normalization. (Right) ReLU network without spectral normalization. (b) Tracking performance and statistics. . . . .	39
4.1 Overview of <i>Neural-Swarm</i> . We learn complex interaction between multirotors using heterogeneous deep sets and design an interaction-aware nonlinear stable controller and a multi-robot motion planner (a). Our approach enables close-proximity flight (minimum vertical distance 24 cm) of heterogeneous aerial teams (16 robots) with significant lower tracking error compared to solutions that do not consider the interaction forces (b,c). . . . .	43

- 4.2 Illustration of Theorem 4.2. We first find a homeomorphism  $\mathbf{q}_K(\cdot)$  between the original space and the latent space, and then find a continuous function  $\bar{\rho}(\cdot)$  such that  $\bar{\rho}(\mathbf{q}_K(\cdot)) = h(\cdot)$ . . . . . 55
- 4.3 Example for Stage 1 of our motion planning with learned dynamics. Here, we have an initial solution for a small (blue) robot and plan for a large (orange) robot. The created search tree of the large robot is color-coded by the magnitude of the interaction force on the orange robot. During the search, we reject states that would cause a significant change in the interaction force for the blue robot (edges in blue). . . . . 60
- 4.4 Hierarchy of control and planning blocks with information flow for commands and sensor feedback. We use different colors to represent heterogeneous neighbors. Note that the neighbors will influence the vehicle dynamics (dashed arrow). . . . . 63
- 4.5  $f_{a,z}$  prediction from the trained  $\{\rho_{\text{small}}, \rho_{\text{large}}, \phi_{\text{small}}, \phi_{\text{large}}, \phi_{\text{env}}\}$  networks. Each heatmap gives the prediction of  $f_{a,z}$  of a vehicle in different horizontal and vertical (global) positions. The (global) position of neighboring drones are represented by drone icons. . . . . 68
- 4.6 Example motion planning result for a three-robot swapping task in 2D (blue and orange: small robots; green: large robot). Top row:  $yz$ -state space plot, where the arrows indicate the velocities every second, and the circles show the robot collision boundary shape at the middle of the task. Bottom row: interaction force for each robot over time (dashed: desired limit per robot). Left: Sampling-based motion planning with neural network to compute trajectories where the large robots moves below the small robots. Middle: Refined trajectories using SCP (dashed) and tracked trajectories (solid). Right: Planned trajectories when ignoring interaction forces (dashed) and tracked trajectories (solid). In this case, a dangerous configuration is chosen where the large robot flies on top of the small robots, exceeding their disturbance limits of 5 g. . . . . 71



4.7	Motion planning results for different scenarios (e.g., SSL refers to two small robots and one large robot) comparing planning without neural network (BL) and planning with neural network (NN) over 5 trials. Top: Worst-case tracking error. Ignoring the interaction force can result in errors of over 10 cm. Bottom: Worst-case interaction force for small and large quadrotors. The baseline has significant violations of the interaction force bounds, e.g., the SL case might create interaction forces greater than 10 g for the small quadrotor. . . . .	72
4.8	Flight test results comparing our solution with learned interaction compensation (NN) with the baseline (BL) in different scenarios. For each case, robots are initially arranged in a circle when viewed from above but at different $z$ -planes and are tasked with moving linearly to the opposite side of the circle in their plane. For each swap, we compute the worst-case $z$ -error of the lowest quadrotor and plot the data over six swaps. . . . .	73
4.9	Generalization to a team of five multirotors. Three small multirotor move in a vertical ring and two large multirotor move in a horizontal ring. The maximum $z$ -error of a small multirotor in the vertical ring with powerful motors is reduced from 10 cm to 5 cm and $f_a$ is predicted accurately. . . . .	74
5.1	Visualization of wind effect for a drone flying in Caltech Real Weather Wind Tunnel. . . . .	79
5.2	Agile flight through narrow gates. (A) Caltech Real Weather Wind Tunnel system, the quadrotor UAV, and the gate. In our flight tests, the UAV follows an agile trajectory through narrow gates, which are slightly wider than the UAV itself, under challenging wind conditions. (B-C) Trajectories used for the gate tests. In (B), the UAV follows a figure-8 through one gate, with wind speed 3.1 m/s or time-varying wind condition. In (C), the UAV follows an ellipse in the horizontal plane through two gates, with wind speed 3.1 m/s. (D-E) Long-exposure photos (with an exposure time of 5 s) showing one lap in two tasks. (F-I) High-speed photos (with a shutter speed of 1/200s) showing the moment the UAV passed through the gate and the interaction between the UAV and the wind. . . . .	82

- 5.3 Offline meta-learning and online adaptive control design. (A) The online adaptation block in our adaptive controller. Our controller leverages the meta-trained basis function  $\phi$ , which is a wind-invariant representation of the aerodynamic effects, and uses composite adaptation (that is, including tracking-error-based and prediction-error-based adaptation) to update wind-specific linear weights  $\hat{a}$ . The output of this block is the wind-effect force estimate,  $\hat{f} = \phi\hat{a}$ . (B) The illustration of our meta-learning algorithm DAIML. We collected data from wind conditions  $\{w_1, \dots, w_K\}$  and applied Algorithm 5.1 to train the  $\phi$  net. (C) The diagram of our control method, where the grey part corresponds to (A). Interpreting the learned block as an aerodynamic force allows it to be incorporated into the feedback control easily. . . . . 84
- 5.4 Training data collection. (A) The xyz position along a two-minute randomized trajectory for data collection with wind speed 8.3 km/h (3.7 m/s), in the Caltech Real Weather Wind Tunnel. (B) A typical 10-second trajectory of the inputs (velocity, attitude quaternion, and motor speed PWM command) and label (offline calculation of aerodynamic residual force) for our learning model, corresponding to the highlighted part in (A). (C) Histograms showing data distributions in different wind conditions. (C) Left: distributions of the  $x$ -component of the wind-effect force,  $f_x$ . This shows that the aerodynamic effect changes as the wind varies. (C) Right: distributions of the pitch, a component of the state used as an input to the learning model. This shows that the shift in wind conditions causes a distribution shift in the input. . . . . 91

5.5 t-SNE plots showing the evolution of the linear weights ( $a^*$ ) during the training process. As the number of training epochs increases, the distribution of  $a^*$  becomes more clustered with similar wind speed clusters near each other. The clustering also has a physical meaning: after training convergence, the right top part corresponds to a higher wind speed. This suggests that DAIML successfully learned a basis function  $\phi$  shared by all wind conditions, and the wind-dependent information is contained in the linear weights. Compared to the case without the adversarial regularization term (using  $\alpha = 0$  in Algorithm 5.1), the learned result using our algorithm is also more explainable, in the sense that the linear coefficients in different conditions are more disentangled. . . . . 94

5.6 Depiction of the trajectory tracking performance of each controller in several wind conditions. The baseline nonlinear controller can track the trajectory well, however, the performance substantially degrades at higher wind speeds. INDI,  $\mathcal{L}_1$ , and Neural-Fly-Constant have similar performance and improve over the nonlinear baseline by estimating the aerodynamic disturbance force quickly. Neural-Fly and Neural-Fly-Transfer use a learned model of the aerodynamic effects and adapt the model in real time to achieve lower tracking error than the other methods. . . . . 104

5.7 Mean tracking errors of each lap in different wind conditions. This figure shows position tracking errors of different methods as wind speed increases. Solid lines show the mean error over 6 laps and the shade areas show standard deviation of the mean error on each lap. The grey area indicates the extrapolation region, where the wind speeds are not covered in training. Our primary method (Neural-Fly) achieves state-of-the-art performance even with a strong wind disturbance. . . . . 105

5.8 Outdoor flight setup and performance. Left: In outdoor experiments, a GPS module is deployed for state estimation, and a weather station records wind profiles. The maximum wind speed during the test was around 17 km/h (4.9 m/s). Right: Trajectory tracking performance of Neural-Fly. . . . . 107

6.1 Motivating example: drone landing trajectories with different speeds. 117

6.2 Episodic learning and control diagram. . . . . 121

6.3	Experimental results. Top: The pendulum task: (a)-(c) are the phase portraits of angle and angular velocity; Blue curve is tracking the desired trajectory with perfect knowledge of $f$ ; the worst-case possible trajectory is calculated according to the uncertainty bound; heatmap is the difference between predicted dynamics (the wind) and the ground truth; and (d) is the tracking error and the maximum density ratio. Bottom: The drone landing task: (e)-(g) are the phase portraits with height and velocity; heatmap is difference between the predicted ground effect) and the ground truth; (h) is the comparison with GPs in landing time. . . . .	127
7.1	Detection results of the milk object of three deep learning models in two images. In the left image, these three models disagree much more than the right image. . . . .	134
7.2	From Part I to future research (more data required from left to right).	136
8.1	Motivations of building the interface between learning and control, and the organization of Part II. . . . .	141
8.2	A trajectory tracking problem to explain how MPC works. . . . .	150
8.3	An example illustrating why beyond regret. . . . .	151
9.1	The motivation of Chapter 9. . . . .	154
9.2	Drone experiment results. . . . .	173
10.1	Controllable canonical form. . . . .	189
10.2	Numerical results of Optimistic ROBD in 1-d and 2-d systems, with different $\lambda$ . LC means the best linear controller in hindsight and OPT means the global optimal controller in hindsight. LC is numerically searched in stable linear controller space. We consider two different types of $w_t$ : $w_t$ is i.i.d. random/random walk, and also two different settings: $w_t$ is known/unknown at step $t$ . . . . .	195
11.1	The power of predictions in online tracking. The left five figures show the desired trajectory (blue) and the actual trajectories (orange). The rightmost figure shows the cost difference (regret) between MPC using $k$ predictions and the offline optimal policy. Note that the y-axis of the rightmost figure is in log-scale. . . . .	234

## LIST OF TABLES

<i>Number</i>	<i>Page</i>
1.1 Number of deaths from different transportation methods every $10^9$ miles in the United States. . . . .	2
2.1 The summary of Part I. . . . .	19
4.1 System identification of the used quadrotors. . . . .	65
4.2 12 scenarios for data collection. . . . .	66
4.3 Ablation analysis. Top: $L = 4$ and $H$ varies. Bottom: $H = 20$ and $L$ varies. The error is the mean squared error (MSE) between $f_{a,z}$ prediction and the ground truth. . . . .	70
5.1 Tracking error statistics in cm for different wind conditions. Two metrics are considered: root-mean-square (RMS) and mean. . . . .	104
9.1 OMAC with convex loss. . . . .	162
9.2 OMAC with element-wise convex loss. . . . .	168
9.3 OMAC with bilinear model. . . . .	169
9.4 OMAC with deep learning. . . . .	172
9.5 ACE results in pendulum (top) and drone (bottom) experiments from 10 random seeds. . . . .	174

*Chapter 1*

## INTRODUCTION

Since the 2010s, deep-learning-based decision making methods have made exciting progress in many domains such as playing real-time strategy games. For example, in 2016, *AlphaGo* (Silver et al., 2016) developed by DeepMind could master the game of Go and defeat the human champion with a large margin, which had been widely considered as a nearly impossible artificial intelligence task for a few decades. Later on, this model-free reinforcement learning methodology quickly generalized to multi-agent game settings: in 2019, another DeepMind team developed *AlphaStar* (Vinyals et al., 2019) and achieved grandmaster-level performance in StarCraft, which is a challenging real-time multi-agent strategy game.

Besides playing games, deep-learning-based decision making methods have also shown fabulous progress in robotics. For instance, in 2022, a deep-learning-based policy (Wurman et al., 2022) developed by researchers at Sony AI outraced the world’s best drivers in the PlayStation game Gran Turismo, which faithfully reproduces the non-linear control challenges of real race cars while also encapsulating the complex multi-agent interactions. Such a novel learned capability also exists in real-world robotic control tasks. For example, in 2019, the OpenAI robotics team developed a deep-learning-based strategy which could solve non-trivial manipulation tasks, such as solving a Rubik’s cube only using one hand (Akkaya et al., 2019).

Although those novel capabilities in decision making enabled by deep learning are fascinating, they are still limited in either game or highly controllable or non-safety-critical settings. In those settings, data collection is relatively cheap and theoretical guarantees are typically not required. For example, in the game of Go or StarCraft, a crazy or unexpected move by the AI agent is acceptable; similarly, for a robotic manipulator solving a Rubik’s cube, dropping the cube is not a big deal.

However, this is not the case in real-world tasks involving high stakes, such as robotic control in hazardous environments, health care, and space exploration. In these tasks, computational resources are limited, data collection is expensive, and theoretical guarantees such as safety are extremely important. Unfortunately, deep-learning-based methods are not ready to be deployed in those settings. For example,

when visiting Caltech, an aerospace director said:

*“I would love to incorporate deep learning into the design, manufacturing, and operations of our aircraft. But I need some guarantees.”*

Such a concern is definitely not unfounded, because the aerospace industry has spent over 60 years making the airplane safer and safer such that the modern airplane is one of the safest transportation methods. In a recent talk “Can We Really Use Machine Learning in Safety-Critical Systems?”<sup>1</sup> at UCLA IPAM, Prof. Richard Murray discussed the number of deaths from transportation every  $10^9$  miles in the United States (see Table 1.1).

Human-driven car	Buses and trains	Airplane	Self-driving car
7	0.1-0.4	0.07	?

Table 1.1: Number of deaths from different transportation methods every  $10^9$  miles in the United States.

Based on this analysis, if we travel from LA to San Francisco, on average, taking a flight is 100 times safer than driving ourselves (also faster). Note that the question mark “?” in Table 1.1 indicates that the self-driving car industry has not tested enough to even get a statistically reliable number (i.e., the total mileage is much less than  $10^9$  miles), although there have already been multiple lethal accidents from deep-learning-based self-driving cars (Greenblatt, 2016).

To summarize, with the unprecedented advances of modern machine learning comes the tantalizing possibility of smart data-driven autonomous systems across a broad range of real-world settings, but we are still begging for answers to the following question:

*For deep-learning-based autonomous systems, how do we ensure a comparable level of safety to human or classic methods while maintaining advantages from deep learning?*

To answer this question, we must confront several key challenges before the widespread deployment of machine learning: (1) Real-world systems have complex and time-varying uncertainties, which require robust and adaptive learning methods. (2)

<sup>1</sup>Video link: <https://youtu.be/Wi8Y--ce28>

High-stakes tasks need control-theoretic guarantees such as safety and stability, which require a framework to unify learning and control theoretical results. (3) In order to be useful, such a theory needs to enable tractable system design.

In light of these challenges, this thesis is to lay the groundwork for learning and control that enables the long-term autonomy of complex real-world systems, such as urban air mobility, space exploration, and medical robots. In pursuit of this vision, this thesis is centered around *establishing a unified algorithmic and theoretical framework that can simultaneously reason over trade-offs in learning and control and using that theory to enable tractable system design to unlock new capabilities in autonomous systems*. The proposed unified framework supported by rigorous yet practical theory advances the state of the art in that complex problems can be simplified greatly in a modularized way, making it much easier to reason about high-level goals.

## 1.1 Thesis Organization

This thesis consists of two parts. In this section, we briefly introduce these two parts and their organizations.

### **Part I: Neural-Control Family: Deep-Learning-Based Nonlinear Control with Learned Robotic Agility**

Part I will build a holistic view on *Neural-Control Family*, which is a family of deep-learning-based nonlinear controllers with not only theoretical guarantees but also learned novel capabilities in agile robotic control.

A summary of Part I can be found in Table 2.1 in Chapter 2. In particular, Chapter 2 will give an overview about *Neural-Control Family*, define the research problem in general robotic systems and specific drone systems, and introduce necessary preliminaries. In Chapter 3, Chapter 4, Chapter 5, and Chapter 6, we will introduce four “children” in the family. Finally, Chapter 7 discusses limitations and future research directions.

### **Part II: Unifying Interfaces Between Learning and Control Theory**

Part II will discuss three unifying interfaces between learning and control theory. Such interfaces deepen the fundamental connections between the two fields, enable much more efficient translation, and bring new perspectives and algorithmic principles.



A summary of Part II can be found in Fig. 8.1 in Chapter 8. In particular, Part II will focus on joint end-to-end learning and control theoretical guarantees in online decision-making problems. Chapter 8 will give an overview and introduce necessary preliminaries. In Chapters 9 to 11, we introduce the three interfaces, respectively. Finally, Chapter 12 discusses future research directions.

## 1.2 Thesis Summary and Contributions

We summarize this thesis and its contributions from three aspects: robotics, theory, and algorithm.

### **Robotics: Agile Control with Novel Capabilities and Certifiable Guarantees**

This thesis focuses on reliable agile robot control with new capabilities which have not been achieved by either pure control or learning methods. In particular, the *Neural-Control Family* in Part I demonstrates state-of-the-art flight control performance in the presence of unknown unsteady aerodynamics. Beyond flight control, the *Neural-Control* idea has been successfully adopted by many other researchers (e.g., for legged, underwater, soft robots), companies, and national agencies, including JPL and DARPA. For the first time, *Neural-Lander* (Chapter 3) enables agile maneuvers only a few millimeters from the ground; *Neural-Swarm* (Chapter 4) enables close-proximity flights (minimum distance 24cm) of a heterogeneous aerial swarm (16 drones), while prior works have to keep a safe distance of 60cm even with 2-3 drones; *Neural-Fly* (Chapter 5) achieves accurate adaptive tracking with an average error down to 3cm in gusty wind conditions up to 12m/s, which improves baselines by one order of magnitude. All these systems run Deep Neural Networks (DNNs) onboard in the control loop but with safety, stability, and robustness guarantees.

The connection between these robotic capabilities and the unified theoretical and algorithmic framework is two-fold: (1) The unified framework allows tractable, efficient, and safe real-world deployment due to its modularity and end-to-end guarantees. For instance, *Neural-Lander/Fly* only needs 5/12-minute training data. (2) Pushing the boundaries of agile robot control requires and inspires new unified perspectives on control and learning. For example, for agile maneuvers in time-varying wind conditions, we have to extract common representations shared by all conditions, which requires and inspires the novel *meta-adaptive control* framework (Chapters 5 and 9). This thesis systematically introduces the unified framework and illustrates why it is *necessary* and *sufficient* for unlocking new capabilities in

real-world autonomous systems.

### **Theory: Bridge Learning and Control in a Unified Framework**

The theoretical parts of this thesis are driven by the quest for a unified framework to characterize and safeguard learned control systems. This thesis is distinctive in three ways. First, it studies how to integrate advanced concepts such as deep learning and nonlinear control, for which unified learning-theoretic and control-theoretic analysis is still in its infancy. Second, it studies how to reconcile fairly fragmented analyses between the two fields (e.g., what does exponentially stable control imply about online regret?), which will deepen the fundamental connections and enable more efficient translation between the two fields. Third, it allows tractable real-world implementations with novel capabilities.

**From statistical learning with regularized DNNs to nonlinear stability and safety.** From both computational and statistical standpoints, the learning methods must incorporate prior physics in real-world dynamical systems. For example, the *Neural-Control Family* in Part I only learns complex aerodynamics which is hard to model using classic approaches. A central question in this diagram is how to leverage the structure in prior physics and translate statistical learning results to stability and safety results in control. Chapter 3 connects generalization bounds of properly spectrally regularized DNNs with input-to-state stability (ISS) bounds in nonlinear control. The experiment found that poorly regularized DNNs without desired Lipschitz properties can cause drones to crash because small noise may destroy local stability. Chapter 4 and Chapter 5 extend this result to multi-agent and real-time adaptation settings. Chapter 6 then generalizes this idea to sequential control problems, by connecting uncertainty bounds under domain shift with safety bounds. These results not only provide sufficient conditions for stable and safe learning-based nonlinear control but also enable roboticists to reason more cleanly between learning and control performance.

**Meta-adaptive control.** In order to have rapidly adaptable autonomous systems operating in changing environments (e.g., varying wind conditions for drones), it is crucial to extract common representations from all environments. However, existing theoretical results focus on either representation/meta-learning with i.i.d. data (i.e., no dynamics) or adaptive control in a single environment. Therefore, this thesis proposes a novel *meta-adaptive control* framework, where meta-learning optimizes a representation shared by all environments. Then the representation is seamlessly

used as basis functions for adaptive nonlinear control in low-dimensional space (Chapters 5 and 9). *Meta-adaptive control* provides both new theoretical results (the first end-to-end non-asymptotic guarantee for multi-task nonlinear control in Chapter 9) and new capabilities in robotics (Chapter 5), both beyond a simple stack of meta-learning and adaptive control.

**Beyond no-regret: competitive online control.** The recent progress in learning-theoretic analyses for online control begs the question: What learning-theoretic guarantees do we need for real-world systems? Existing results focus on linear systems with classic no-regret guarantees. However, no-regret policies compare with the optimal static controller in a specific class (typically linear policy class), which could be arbitrarily suboptimal in real-world nonlinear or time-varying systems (see examples in Chapter 10). Therefore, this thesis studies *competitive online control*, which uses stronger metrics beyond regret, i.e., competitive ratio and dynamic regret (competitive difference). Those metrics directly compare with the global optimum, thus naturally suitable for real-world systems. This thesis designs competitive policies in time-varying and nonlinear systems, via novel reductions from online optimization to control (Chapters 10 and 11). Moreover, we show new fundamental limits via novel lower bounds, e.g., the impact of delay (Chapter 10).

**Understand MPC from learning perspectives.** Another critical question is begged in online learning and control: Do established control methods have strong learning guarantees? In particular, Model Predictive Control (MPC) has been one of the most successful methods in industrial control since the 1980s. However, many learning theorists are studying RL algorithms, but few are analyzing MPC and why it is so powerful. To close this gap, this thesis proves the first non-asymptotic guarantee for MPC (Chapter 11), showing that MPC is *near-optimal* in the sense of dynamic regret in online LQR control with predictable disturbance. Then Chapter 11 further extends to settings with inexact predictions and LTV systems, in the *competitive online control* framework. These results found common ground for learning and control theory and imply fundamental algorithmic principles.

#### **Algorithm: Reliable Decision Making Methods with Certifiable Guarantees**

The algorithmic goal of this thesis is to translate the improved theoretical understand (in particular, the established unified abstractions for learning and control) into tractable algorithms that inherit both the flexibility and accuracy of learning and the certifiable guarantees of control.

**Stable, robust, and adaptive deep-learning-based nonlinear control.** As a powerful yet obscure black box, deep learning must incorporate principled regularizations for reliable and tractable real-world implementations. First, Part I uses Lipschitz-constrained DNNs to learn the residual dynamics, where the constraint is specified by the established unified framework between regularized learning and nonlinear stability. Such control-theoretic regularizations ensure the stability and robustness of deep-learning-based nonlinear controllers, and also improve generalization to unseen data (Chapters 3 to 5). The second type of regularization for learning is *invariance*. *Neural-Swarm* (Chapter 4) leverages the *permutation-invariance* of swarm dynamics to develop Heterogeneous Deep Sets for learning swarm interactions in a decentralized manner, which enables generalization from 1-3 robots in training to 5-16 in testing. *Neural-Fly* (Chapter 5) proposes the Domain Adversarially Invariant Meta-Learning (DAIML) algorithm, which uses an adversarial regularizer to train a *domain-invariant* representation for online adaptation. These principled regularizations are critical for efficient and safe real-world deployment.

**Safe exploration in nonlinear dynamical systems.** Safety-critical tasks such as space exploration and agile drone landing are challenging because (1) there is no expert collecting data, and (2) there exists non-trivial domain shift (e.g., sim2real, exploring to unseen states). Therefore, we have to quantify uncertainty and design safe learning and exploration methods. Built on the connection between uncertainty bounds and safety bounds in the unified framework, Chapter 6 proposes the first safe exploration algorithm with robust end-to-end learning and control guarantees under domain shift, in both deterministic and stochastic settings. The key idea is to derive uncertainty bounds from distributionally robust learning and use them in robust trajectory optimization.

## References

- Akkaya, Ilge et al. (2019). *Solving Rubik's cube with a robot hand*. In: *arXiv preprint arXiv:1910.07113*.
- Greenblatt, Nathan A. (2016). *Self-driving cars and the law*. In: *IEEE Spectrum* 53.2, pp. 46–51.
- Silver, David et al. (2016). *Mastering the game of Go with deep neural networks and tree search*. In: *Nature* 529.7587, pp. 484–489.
- Vinyals, Oriol et al. (2019). *Grandmaster level in StarCraft II using multi-agent reinforcement learning*. In: *Nature* 575.7782, pp. 350–354.

Wurman, Peter R. et al. (2022). *Outracing champion Gran Turismo drivers with deep reinforcement learning*. In: *Nature* 602.7896, pp. 223–228.

# **Part I**

**Neural-Control Family: Deep-Learning-Based Nonlinear Control with**

**Learned Robotic Agility**

In this part of the thesis, we will build a holistic view on *Neural-Control Family*, which is a family of deep-learning-based nonlinear controllers with theoretical guarantees and learned new capabilities in robotics. The general idea is that we have a mixed dynamical system  $\dot{x} = f_{\text{known}}(x, u) + f_{\text{unknown}}(x, u)$  and our goal is to model  $f_{\text{unknown}}$  using modern deep learning tools and design control laws. We will show the importance and necessity of building a unified framework of control and learning, e.g., incorporating control-theoretic constraints in learning.

In Chapter 2, we will give an overview about the *Neural-Control Family*, which defines the research problem in general robotic systems and specific drone systems. In Chapter 3, Chapter 4, Chapter 5 and Chapter 6, we will introduce four “children” in the family. Finally, in Chapter 7, we will discuss future research directions. This part is mainly based on the following papers:

Liu, Anqi, Guanya Shi, Soon-Jo Chung, Anima Anandkumar, and Yisong Yue (2020). *Robust regression for safe exploration in control*. In: *Learning for Dynamics and Control*. PMLR, pp. 608–619. URL: <https://proceedings.mlr.press/v120/liu20a.html>.

Nakka, Yashwanth Kumar, Anqi Liu, Guanya Shi, Anima Anandkumar, Yisong Yue, and Soon-Jo Chung (2021). *Chance-constrained trajectory optimization for safe exploration and learning of nonlinear Systems*. In: *IEEE Robotics and Automation Letters* 6.2, pp. 389–396. DOI: 10.1109/LRA.2020.3044033.

O’Connell, Michael, Guanya Shi, Xichen Shi, Kamyar Azizzadenesheli, Anima Anandkumar, Yisong Yue, and Soon-Jo Chung (2022). *Neural-Fly enables rapid learning for agile flight in strong winds*. In: *Science Robotics* 7.66, eabm6597. DOI: 10.1126/scirobotics.abm6597.

Shi, Guanya, Wolfgang Hönig, Xichen Shi, Yisong Yue, and Soon-Jo Chung (2022). *Neural-Swarm2: Planning and control of heterogeneous multirotor swarms using learned interactions*. In: *IEEE Transactions on Robotics* 38.2, pp. 1063–1079. DOI: 10.1109/TRO.2021.3098436.

Shi, Guanya, Wolfgang Hönig, Yisong Yue, and Soon-Jo Chung (2020). *Neural-Swarm: Decentralized close-proximity multirotor control using learned interactions*. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3241–3247. DOI: 10.1109/ICRA40945.2020.9196800.

Shi, Guanya, Xichen Shi, Michael O’Connell, Rose Yu, Kamyar Azizzadenesheli, Animashree Anandkumar, Yisong Yue, and Soon-Jo Chung (2019). *Neural Lander: Stable drone landing control using learned dynamics*. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 9784–9790. DOI: 10.1109/ICRA.2019.8794351.

## Chapter 2

### OVERVIEW

**Notations.** In Part I,  $\|\cdot\|$  indicates the 2-norm of a vector or the induced 2-norm of a matrix.  $\lambda_{\min}(\cdot)$  and  $\lambda_{\max}(\cdot)$  denote the minimum or the maximum eigenvalue of a real symmetric matrix.  $a \circ b$  refers to the composition of two functions  $a$  and  $b$ .  $(\cdot)^\dagger$  is the Moore-Penrose inverse of a matrix. Finally,  $\dot{x}$  denotes the time derivative of a variable  $x$ , i.e.,  $\dot{x} = \frac{dx}{dt}$ .

#### 2.1 A Mixed Model for Robot Dynamics and the Nonlinear Control Law

Often times in real-world robotic systems, there are complicated uncertainties in dynamics but we also have some *prior knowledge*. For example, as shown in later chapters, for a quadrotor, it is very easy and standard to model the *rigid body* dynamics in SE(3). However, it is in general much more difficult, if not impossible, to model the aerodynamics such as ground effect, down wash effect, air drag and wind effect (G. Shi, X. Shi, et al., 2019; G. Shi, Hönig, Yue, et al., 2020; G. Shi, Hönig, X. Shi, et al., 2022; O’Connell et al., 2022).

To systematically and simultaneously study the known part and the unknown part in the dynamics, in this part of the thesis, we will consider a mixed model for robot dynamics in the form of the Euler-Lagrange equation. In particular, we consider the following robot dynamics model:

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = Bu + \underbrace{f(q, \dot{q}, u, t)}_{\text{unknown}} \quad (2.1)$$

where  $q, \dot{q}, \ddot{q} \in \mathbb{R}^n$  are the  $n$  dimensional position, velocity, and acceleration vectors,  $M(q)$  is the symmetric, positive definite inertia matrix,  $C(q, \dot{q})$  is the Coriolis matrix,  $g(q)$  is the gravitational force vector,  $u \in \mathbb{R}^m$  is the control force and  $B \in \mathbb{R}^{n \times m}$  is the actuation matrix. Most importantly,  $f(q, \dot{q}, u, t)$  incorporates unmodeled dynamics, and it is potentially *time-variant* (i.e.,  $f$  depends on time  $t$ ).

Note that one important fact about the Euler-Lagrange equation in Eq. (2.1) is that  $\dot{M} - 2C$  is a skew-symmetric matrix. We will heavily use this fact for nonlinear stability analysis in the following chapters.



## Tasks

**Stable and robust trajectory tracking.** In Part I, we consider the *trajectory tracking* task, where we have a *reference* (or desired) trajectory  $q_d(t)$  which we hope the robot can precisely track. In particular, our goal is to guarantee that the controller is (globally) *stable* and *robust*.

We say the closed-loop system is globally asymptotically stable if  $\|q - q_d\| \rightarrow 0$  asymptotically wherever  $q$  starts (i.e., whatever  $q(0)$  is). Furthermore, we say the closed-loop system is *exponentially* stable if  $\|q - q_d\| \rightarrow 0$  exponentially. Note that exponential stability typically implies robustness in the sense of input-to-state stability (ISS) (Khalil, 2002). In the context of robotic trajectory tracking, it means that  $\|q - q_d\|$  will exponentially converge to an error ball  $\epsilon$ , whose size is typically proportional to the size of extra disturbance on top of  $f$  or the *learning error* (the gap between the estimation of  $f$  and the true  $f$ ) (Slotine and Li, 1991; O’Connell et al., 2022). Therefore, having exponential stability is critical for learning-based control because an exponentially stable controller can handle errors from imperfect learning.

**Safe motion planning and control.** At this stage, readers may have the question “where is the reference trajectory  $q_d$  from?” The answer to this question is exactly the *motion planning* problem, whose goal is to find a path  $q_d(t)$  for the robot to track. Typically, we want to make sure such a path is *safe* in the state space. Formally speaking,  $[q_d; \dot{q}_d] \in \mathcal{S}$  where  $\mathcal{S}$  is some safe set. For example, for multi-robot planning we must consider collision and obstacle avoidance.

It is worth noting that safety is not only important in planning, but also in feedback control. Once we have  $q_d$ , the next question is naturally how to design a controller such that the actual trajectory is also safe, i.e.,  $[q; \dot{q}] \in \mathcal{S}$ . Note that it is relatively easy if the tracking is *perfect* (i.e.,  $q = q_d$ ), but much more challenging if there exist uncertainties.

## Remarks and Assumptions

**Remark (nonlinear, time-variant, unknown and non-affine systems).** Controlling the system in Eq. (2.1) is challenging in four folds. (1) The  $f$  term is unknown and could be very complicated. (2) Both the known part and the unknown part ( $f(q, \dot{q}, u, t)$ ) are highly nonlinear. (3) The  $f$  term is in generally time-variant. Note that most of theories developed in Part I still hold if the nominal part (i.e.,  $M, C, g, B$ ) is also time-variant, and we assume they are time-invariant for simplicity

(also for all our experiments the nominal part is indeed time-invariant). (4) The system is non-affine in  $u$ , and in particular, the unknown term  $f(q, \dot{q}, u, t)$  might depend on  $u$ . Controlling a non-affine linear system is notoriously challenging because  $u$  often appears in both sides of a controller equation (e.g.,  $Bu = \pi(x, u)$ ).

**Assumption 2.1** (full actuation). *In Part I, we focus on fully actuated systems, which means  $\text{rank}(B) = n$  and  $m \geq n$ . Generally speaking, guaranteed safe learning and control (especially adaptive control) in nonlinear systems with under actuation is very challenging (Lopez and Slotine, 2021) and left for future directions. We will revisit this point in Chapter 7. In the second part of this thesis (Part II), we will study learning and control in some under-actuated linear systems.*

**Assumption 2.2** (full observation). *In this thesis, we focus on fully observable settings, which means we can directly observe the state of the robot ( $q$  and  $\dot{q}$ ). The fully observable assumption matches robotic experiments in this thesis, because we deploy external motion capture (mocap) systems to localize the robot in SE(3). Nevertheless, note that some of control methods developed in Part I are robust for outdoor experiments without mocap systems, where we first use estimators (e.g., extended Kalman filter) to estimate robot state and then use these estimations for control (e.g., see outdoor experiments in Chapter 5). Having end-to-end guarantees in nonlinear partially observable systems is another interesting future research topic (see more discussions in Chapter 7).*

### The Nonlinear Control Law

In this subsection, we introduce the basic structure of controllers covered in Part I. Note that each child in the family (Neural-Lander/Swarm/Fly) has different downstream control structures but the high-level structures are consistent. We start by defining some notations. The composite velocity tracking error term  $s$  and the reference velocity  $\dot{q}_r$  are defined such that

$$s = \dot{q} - \dot{q}_r = \dot{\tilde{q}} + \Lambda \tilde{q} \quad (2.2)$$

where  $\tilde{q} = q - q_d$  is the position tracking error and  $\Lambda$  is a positive definite gain matrix. Note when  $s$  exponentially converges to an error ball around 0,  $q$  will exponentially converge to a proportionate error ball around the desired trajectory  $q_d(t)$  (see Lemma 2.3). Formulating our control law in terms of the composite velocity error  $s$  simplifies the analysis and gain tuning without loss of rigor.

Our nonlinear control structure is given by

$$Bu = \underbrace{M(q)\ddot{q}_r + C(q, \dot{q})\dot{q}_r + g(q)}_{\text{nonlinear feedforward}} \underbrace{-Ks}_{\text{linear feedback}} \underbrace{-\hat{f}}_{\text{learned feedforward}}, \quad (2.3)$$

where  $K$  is a positive definite control gain matrix and  $\hat{f}$  is a learned term to “compensate” the unknown term  $f$  in real time. Note that this nonlinear controller includes feedback on position and velocity errors and feedforward terms to account for known dynamics and desired acceleration, which allows good tracking of dynamic trajectories in the presence of nonlinearities (e.g.,  $M(q)$  and  $C(q, \dot{q})$  are non-constant in drone attitude control). Moreover, this control law also “predicts” and then compensates the unknown dynamics via the  $\hat{f}$  term. We will have different structures for  $\hat{f}$  in different cases, e.g., a fixed regularized DNN in Neural-Lander (Chapter 3) and a real-time adaptive DNN in Neural-Fly (Chapter 5).

## 2.2 Example: Quadrotor Dynamics and Control

In this section, we will provide a concrete and detailed example for the general robot dynamics model (Eq. (2.1)) in Section 2.1: quadrotor dynamics with unknown aerodynamics. Most of experiments in Part I also focus on drones.

Consider states given by global position,  $p \in \mathbb{R}^3$ , velocity  $v \in \mathbb{R}^3$ , attitude rotation matrix  $R \in \text{SO}(3)$ , and body angular velocity  $\omega \in \mathbb{R}^3$ . Then dynamics of a quadrotor are

$$\dot{p} = v, \quad m\dot{v} = mg + Rf_u + f_a, \quad (2.4a)$$

$$\dot{R} = RS(\omega), \quad J\dot{\omega} = J\omega \times \omega + \tau + \tau_a, \quad (2.4b)$$

where  $m$  is the mass,  $J$  is the inertia matrix of the quadrotor,  $S(\cdot)$  is the skew-symmetric mapping,  $g$  is the gravity vector,  $f_u = [0, 0, T]^\top$  and  $\tau = [\tau_x, \tau_y, \tau_z]^\top$  are the total thrust and body torques from four rotors predicted by the nominal model. In particular, typical quadrotor control input uses squared motor speeds  $u = [n_1^2, n_2^2, n_3^2, n_4^2]^\top$ , and is linearly related to the total thrust and body torques:

$$\underbrace{\begin{bmatrix} T \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix}}_{\eta_o} = \underbrace{\begin{bmatrix} c_T & c_T & c_T & c_T \\ 0 & c_T l_{\text{arm}} & 0 & -c_T l_{\text{arm}} \\ -c_T l_{\text{arm}} & 0 & c_T l_{\text{arm}} & 0 \\ -c_Q & c_Q & -c_Q & c_Q \end{bmatrix}}_{B_0} \cdot \underbrace{\begin{bmatrix} n_1^2 \\ n_2^2 \\ n_3^2 \\ n_4^2 \end{bmatrix}}_u \quad (2.5)$$

where  $c_T$  and  $c_Q$  are rotor force and torque coefficients,  $l_{\text{arm}}$  denotes the length of rotor arm, and  $\eta_o$  is the output wrench. Note that  $B_0$  is an invertible matrix.

**Control task.** In this thesis, we are interested in the position trajectory tracking problem for a quadrotor. Namely, there exists a time-variant desired position trajectory  $p_d(t)$  and our goal is to design a controller such that the tracking error  $\tilde{p} = p - p_d$  is near-zero in the presence of unknown aerodynamic disturbance  $f_a$  and  $\tau_a$ .

**Remark (generalize to other aerial robots).** Generally speaking, the model in Eq. (2.4) and Eq. (2.5) can also model other aerial vehicles with more than four actuators (i.e., to make sure the actuation matrix satisfying  $\text{rank}(B_0) \geq 4$ ), such as hexacopter, octocopter, and some hybrid vehicles (e.g., flying cars in (X. Shi et al., 2018)).

### Residual Aerodynamic Force and Torque

The key difficulty of precise drone control is the influence of unknown disturbance forces  $f_a = [f_{a,x}, f_{a,y}, f_{a,z}]^\top$  and torques  $\tau_a = [\tau_{a,x}, \tau_{a,y}, \tau_{a,z}]^\top$  in Eq. (2.4), which originate from complex aerodynamic effects. For example, ground effect when the drone is flying close to the ground (or some other boundaries) (G. Shi, X. Shi, et al., 2019), downwash effect when one drone is flying below the other ones (G. Shi, Hönig, X. Shi, et al., 2022), and wind effect when the drone is flying in strong wind conditions (O’Connell et al., 2022).

Generally speaking, the unknown force  $f_a$  and torque  $\tau_a$  depend on the drone’s states (position, velocity, attitude, angular velocity), rotor speeds ( $n_1, \dots, n_4$ ), and external environmental conditions (e.g., wind conditions). To summarize,  $f_a$  and  $\tau_a$  are unknown, nonlinear, and potentially time-variant.

In Part I’s experiments, we will use deep learning approaches to model  $f_a$  and  $\tau_a$ . We define the deep learned estimation as  $\hat{f}_a$  and  $\hat{\tau}_a$ , respectively. Different chapters have different learning structures (e.g., a fixed constrained DNN in Chapter 3 and an online adaptive DNN in Chapter 5).

### Hierarchical Control Structure and the Reduction to General Robot Dynamics

Note that Eq. (2.4) cannot be directly converted to Eq. (2.1) because quadrotor is an underactuated system in  $SE(3)$ . However, typical quadrotor control consists of a cascaded or hierarchical control structure which separates the design of the position controller, attitude controller, and thrust mixer (allocation). For both

position controller and attitude controller, we can reduce them to Eq. (2.1).

**Position controller.** When designing the position controller, we assume that the global rotor force vector  $Rf_u$  can be directly controlled. In other words, we can cast the position dynamics in Eq. (2.4a) into the form of Eq. (2.1), by taking  $q = p$ ,  $M(q) = mI$ ,  $C(q, \dot{q}) \equiv 0$ ,  $B = I$ ,  $f = f_a$ , and  $u = Rf_u$ . Corresponding to Eq. (2.3), the nonlinear position controller has the following structure:

$$f_d = \underbrace{m\ddot{p}_r - mg}_{\text{feedforward}} \underbrace{-Ks}_{\text{linear feedback}} \underbrace{-\hat{f}_a}_{\text{learned feedforward}} \quad (2.6)$$

where  $s = \dot{p} - \dot{p}_r = \dot{\tilde{p}} + \Lambda\tilde{p}$  is the composite velocity error (see details in Section 2.1),  $f_d = (Rf_u)_d$  is the desired rotor force and  $\hat{f}_a$  is a learned term to compensate the unknown disturbance force  $f_a$ . Note that this controller includes linear feedback on position and velocity errors and feedforward terms to account for gravity and desired acceleration, which allows agile tracking of dynamic trajectories. Moreover, the  $\hat{f}_a$  term predicts and compensates the unknown disturbance.

**Attitude controller.** With the desired rotor force  $f_d = (Rf_u)_d$ , consequently, desired total thrust  $T_d$  and desired force direction  $\hat{k}_d$  can be computed as

$$T_d = f_d \cdot \hat{k}, \text{ and } \hat{k}_d = f_d / \|f_d\|, \quad (2.7)$$

with  $\hat{k}$  being the unit vector of rotor thrust direction (typically  $z$ -axis in quadrotors). Using  $\hat{k}_d$  and fixing a desired yaw angle, desired attitude  $R_d$  can be deduced (Morgan et al., 2016).

Then we need to design a nonlinear attitude controller which uses the desired torque  $\tau_d$  from rotors to track  $R_d(t)$ . The quadrotor attitude dynamics Eq. (2.4b) is also a special case of the general robot dynamics model Eq. (2.1) (Slotine and Li, 1991; Murray et al., 2017), but it is worth noting that different choices of rotation errors will yield different controllers. For example, one nonlinear attitude controller is given by (X. Shi et al., 2018):

$$\tau_d = \underbrace{J\dot{\omega}_r - J\omega \times \omega_r}_{\text{nonlinear feedforward}} \underbrace{-K_\omega(\omega - \omega_r)}_{\text{linear feedback}} \underbrace{-\hat{\tau}_a}_{\text{learned feedforward}}, \quad (2.8)$$

where the reference angular rate  $\omega_r$  is designed similar to Eq. (2.2), so that when  $\omega \rightarrow \omega_r$ , exponential trajectory tracking of a desired attitude  $R_d(t)$  is guaranteed. Similarly,  $\hat{\tau}_a$  is a learned term to compensate the unknown torque  $\tau_a$ . Similar to the position controller, the attitude controller also includes a nonlinear feedforward term

and a linear feedback term, and a  $\hat{\tau}_a$  term to account for the unknown aerodynamic torque.

**Thrust mixer (allocation).** Finally, with the desired total thrust  $T_d$  and the desired torque  $\tau_d$ , we can straightforwardly solve the rotor speed  $n_1, \dots, n_4$  using Eq. (2.5). Note that if the control limit ( $0 \leq n_i \leq \bar{n}, \forall i$ ) becomes infeasible, a thrust mixer typically compromises some maneuverability (especially yaw) to optimize the flight performance. Some thrust mixers also consider motor delay to some extent, i.e.,  $n_1, \dots, n_4$  cannot immediately change  $T$  and  $\tau$ . We will discuss more about delay compensation in Chapter 4.

**Remark (benefits and limitations of the hierarchical control structure).** The aforementioned hierarchical controller can be summarized as the following steps:

1. The position controller accounts for the unknown aerodynamic force  $f_a$ , and computes the desired rotor force  $f_d$ .
2. Compute the desired total thrust  $T_d$  and the desired attitude  $R_d$ .
3. Given  $R_d$ , the attitude controller accounts for the unknown aerodynamic torque  $\tau_a$ , and computes the desired torque  $\tau_d$ .
4. Given  $T_d$  and  $\tau_d$ , the thrust mixer computes the rotor speed  $n_1, \dots, n_4$ .

This procedure has the following benefits:

- It is supported by rigorous theories, because both position and attitude controllers follow the standard fully-actuated robot dynamics in Eq. (2.1).
- It is relatively easy to implement and debug. In particular, we do not need to customize drones or flight controllers since most commonly-used off-the-shelf controllers (such as PX4) support the “offboard” control mode, which only needs a position control law (i.e.,  $T_d$  and  $R_d$ ) and automatically tracks the reference attitude  $R_d$ . In experiments, the hierarchical control law is robust even flying outdoors.
- In practice, the attitude controller is running at a much higher frequency (e.g., 400Hz) than the position controller (e.g., 50Hz), which matches the key assumption made in the cascading structure (i.e., the attitude controller runs fast enough to track the desired attitude generated from the upstream position controller).

However, this cascading structure implicitly assumes that the cross-error term between position and attitude control is bounded, and the attitude controller can track arbitrary trajectories generated from the upstream controller. Therefore, such a cascading structure may not be proper for acrobatic maneuvers such as flips. Alternatively, we can also use a single optimization-based controller such as model predictive control (MPC) to directly compute motor speed commands from desired trajectories (Tal and Karaman, 2021; Hanover et al., 2021). Compared to the cascading controller structure, it is very challenging to have stable and robustness guarantees because of the nonlinear and underactuated nature. Moreover, the single optimization-based controller requires full custom flight controllers and drones.

### 2.3 Organization of Part I: Unifying Learning and Nonlinear Control

In Part I, we will discuss four cases of Eq. (2.1) with the focus on the unknown term  $f(q, \dot{q}, u, t)$ . Meanwhile, we also introduce different corresponding drone dynamics in Eq. (2.4) and controllers.

*Neural-Control Family* consists of these four different cases. Each case (chapter) will answer the following questions:

1. What is the main challenge with the unknown term  $f$ ?
2. How to design the learning and control strategy with theoretical guarantees?
3. In real-world robotic experiments, how does the proposed learning and control method compare with baselines and state-of-the-art methods?
4. What new capabilities (in terms of robotic agility) does the proposed learning and control method bring? In other words, why is the proposed learning and control method crucial?

Answers to these questions can be summarized in Table 2.1.

Before diving into details, here we highlight how we unify learning and nonlinear control in each chapter:

**Neural-Lander (Chapter 3)** considers the model  $f = f(q, \dot{q}, u)$ . We use a deep neural network  $\hat{f}(q, \dot{q}, u)$  to approximate  $f$ . In this case, the challenge is to guarantee that the controller in Eq. (2.3) is stable and robust with a DNN term  $\hat{f}$ . To that end, we propose a control-theoretic Lipschitz-constrained deep learning framework

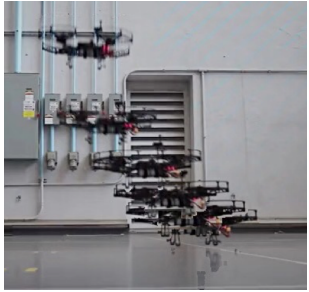
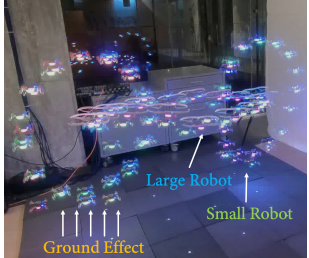
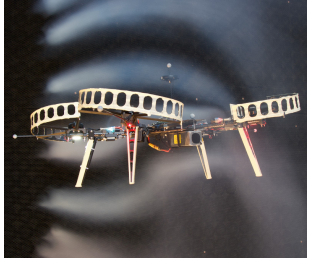
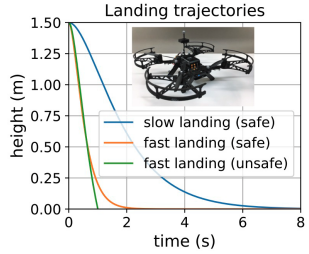
	Challenge	Solution	New capabilities
Neural-Lander (Chapter 3)	$f$ depends on $u$	Control-theoretic Lipschitz constraints for stability guarantees	 <p>Agile maneuvers very close to the ground</p>
Neural-Swarm (Chapter 4)	$f$ depends on heterogeneous neighbors in a swarm	Encoding permutation invariance in learning and control	 <p>Close-proximity heterogeneous swarm control</p>
Neural-Fly (Chapter 5)	$f$ depends on time-variant environmental conditions	Invariant meta-learning and composite adaptation	 <p>Precise flight in time-variant strong winds</p>
Safe Exploration (Chapter 6)	No pre-collected data	Uncertainty quantification under domain shift	 <p>Aggressive maneuvers with end-to-end safety guarantees</p>

Table 2.1: The summary of Part I.



to guarantee the nonlinear stability and robustness. This learning framework also improves sampling efficiency and generalizability.

**Neural-Swarm (Chapter 4)** considers a multi-agent interactive setting, i.e.,  $f = f(q, \dot{q}, \text{neighbors' states})$ . In this case, we aim to design a decentralized learning and control method which can generalize to different numbers of agents. With this goal, we leveraged the permutation invariance property in the swarm system, and modeled the multi-agent interaction using heterogeneous deep sets. We also developed an interaction-aware motion planner and a delay compensator.

**Neural-Fly (Chapter 5)** considers  $f = f(q, \dot{q}, w(t))$  where  $w$  describes potentially time-variant environmental conditions (e.g., wind conditions for a drone). Our goal is to develop an efficient online adaptation strategy such that the controller can adapt in changing conditions in real time. Hence, we use  $\phi(q, \dot{q})a$  to approximate  $f$ , where  $\phi$  is a pre-meta-trained representation shared by all conditions and  $a$  is a condition-dependent linear coefficient which will be adapted by adaptive control in an online manner.

In Neural-Lander/Swarm/Fly, we all need pre-collected data to train or pre-train our deep learning models. In **Safe Exploration (Chapter 6)**, we have to learn and control from scratch without pre-collect data or human in the loop. The key challenge is to trade-off between exploration and safety such that we can gradually achieve an aggressive control goal with safety guarantees in the whole process.

## 2.4 Preliminaries on Deep Learning

In this section, we will introduce some preliminaries which are heavily used in Part I.

### ReLU Deep Neural Networks

In Part I, we often learn the unknown dynamics term  $f$  in Eq. (2.1) using a DNN with Rectified Linear Units (ReLU) activation. In general, DNNs equipped with ReLU converge faster during training, demonstrate more robust behavior with respect to changes in hyperparameters, and have fewer vanishing gradient problems compared to other activation functions such as *sigmoid* (Krizhevsky et al., 2012).

A ReLU deep neural network represents the functional mapping from the input  $x$  to the output  $g(x, \theta)$ , parameterized by the DNN weights  $\theta = W_1, \dots, W_{L+1}$ :

$$g(x, \theta) = W_{L+1}\sigma(W_L(\sigma(W_{L-1}(\dots\sigma(W_1x)\dots))), \quad (2.9)$$

where the activation function  $\sigma(\cdot) = \max(\cdot, 0)$  is called the element-wise ReLU

function. We omit the bias term of each layer for simplicity. Note that omitting the bias term does not change the Lipschitz property of the DNN so it will not influence our theoretical analysis. ReLU is less computationally expensive than *tanh* and *sigmoid* because it involves simpler mathematical operations. However, deep neural networks are usually trained by first-order gradient based optimization, which is highly sensitive on the curvature of the training objective and can be unstable (Salimans and Kingma, 2016). To alleviate this issue, we apply the spectral normalization technique (Miyato et al., 2018) and introduce the details as follows.

### Spectral Normalization and Constrained Training

Spectral normalization stabilizes DNN training by constraining the Lipschitz constant of the objective function  $g(x, \theta)$ . Spectrally normalized DNNs have also been shown to generalize well (Bartlett et al., 2017), which is an indication of stability in machine learning. Mathematically, the Lipschitz constant of a function  $\|g\|_{\text{Lip}}$  is defined as the smallest value such that

$$\forall x, x' : \frac{\|g(x) - g(x')\|_2}{\|x - x'\|_2} \leq \|g\|_{\text{Lip}}. \quad (2.10)$$

It is well known that the Lipschitz constant of a general differentiable function  $g$  is the maximum spectral norm (i.e., induced 2-norm or maximum singular value) of its gradient over its domain:  $\|g\|_{\text{Lip}} = \sup_x \|\nabla g(x)\|_2$ . In particular, if  $g$  is an affine function  $g(x) = Wx + b$ , then  $\|g\|_{\text{Lip}} = \|W\|_2$ .

The ReLU DNN in Eq. (2.9) is a composition of functions. Thus we can bound the Lipschitz constant of the network by constraining the spectral norm of each layer. Using the fact that the Lipschitz norm of ReLU activation function  $\sigma(\cdot)$  is equal to 1, with the inequality  $\|g_a \circ g_b\|_{\text{Lip}} \leq \|g_a\|_{\text{Lip}} \cdot \|g_b\|_{\text{Lip}}$  for two functions  $g_a$  and  $g_b$ , we can find the following bound on  $\|g\|_{\text{Lip}}$ :

$$\|g\|_{\text{Lip}} \leq \|W_{L+1}\|_2 \cdot \|\sigma\|_{\text{Lip}} \cdots \|W_1\|_2 = \prod_{l=1}^{L+1} \|W_l\|_2. \quad (2.11)$$

Given the upper bound in Eq. (2.11), we propose the stochastic gradient descent (SGD) with spectral normalization algorithm (Algorithm 2.1). In Algorithm 2.1,  $\gamma > 0$  is the intended upper bounded for the Lipschitz constant of the DNN  $g(x, \theta)$ ,  $\eta > 0$  is the learning rate, and  $\ell(g(x), y)$  could be any differentiable loss function, e.g., squared loss  $\|g(x) - y\|^2$ .  $\mathcal{D}$  is a dataset comprising  $N$  input ( $x$ ) and label ( $y$ ) pairs. Essentially, Algorithm 2.1 enforces the spectral norm of each layer matrix upper bounded by  $\gamma^{\frac{1}{L+1}}$ . The following lemma formally bounds  $\|g\|_{\text{Lip}}$ :

---

**Algorithm 2.1:** SGD with Spectral Normalization

---

**Hyperparameter:** Normalization constant  $\gamma > 0$ , learning rate  $\eta > 0$ **Input:** Dataset  $\mathcal{D} = \{x_i, y_i\}_{i=1}^N$  and a loss function  $\ell(g(x), y)$ **Initialize:** Neural network parameter  $\theta = W_1, \dots, W_{L+1}$ 1 **repeat**2     Randomly sample a batch  $D$  from the dataset  $\mathcal{D}$ 3     Gradient descent on  $D$ :

4

$$\theta \leftarrow \theta - \eta \cdot \sum_{(x,y) \in D} \nabla_{\theta} \ell(g(x, \theta), y) \quad (2.12)$$

Spectral normalization of each layer:

5     **for**  $1 \leq l \leq L + 1$  **do**6         **if**  $\|W_l\|_2 > \gamma^{\frac{1}{L+1}}$  **then**

7

$$W_l \leftarrow \frac{W_l}{\|W_l\|_2} \cdot \gamma^{\frac{1}{L+1}}, \quad (2.13)$$

8 **until convergence**

---

**Lemma 2.1.** *Algorithm 2.1 guarantees that*

$$\|g\|_{\text{Lip}} \leq \gamma. \quad (2.14)$$

*Proof.* As in Eq. (2.11), we have  $\|g\|_{\text{Lip}} \leq \prod_{l=1}^{L+1} \|W_l\|_2 \leq \prod_{l=1}^{L+1} \gamma^{\frac{1}{L+1}} = \gamma$ .  $\square$

**Remark (spectral norm computation).** Note that Algorithm 2.1 needs to compute the spectral norm of  $W_l, \forall l$  in each training epoch. Computing the largest singular value of a large matrix could be computationally heavy. However, one can use the power iteration method to estimate  $\|W_l\|_2$  (Miyato et al., 2018). Nevertheless, in this thesis we focus on learning the dynamics so the DNN is in general small (within 100K parameters) so we directly use `numpy.linalg.norm(W, 2)` to compute the spectral norm.

**Remark (optimization and back propagation).** In practice, we use the Adam optimizer to train the neural network  $g(x, \theta)$ . Note that the normalization step  $W_l \leftarrow \frac{W_l}{\|W_l\|_2} \cdot \gamma^{\frac{1}{L+1}}$  is not backward propagated, which is different from the original implementation of spectral normalization in (Miyato et al., 2018). Miyato et al. (2018) normalizes each layer’s spectral norm to exactly 1 in each step via back propagation. Instead, after each optimization step, we “project” each layer to have a  $\gamma^{\frac{1}{L+1}}$  spectral norm if its norm is bigger than  $\gamma^{\frac{1}{L+1}}$ . Alternatively, we can also

directly deploy the standard spectral normalization implementation in PyTorch with a scaling factor  $\gamma^{\frac{1}{L+1}}$ . In practice, we found our implementation is easier to tune and have slightly better performance.

**Remark (looseness of Eq. (2.11)).** It is not hard to notice that the lower bound given in Eq. (2.11) could be loose, due to the nature of the composition inequality  $\|g_a \circ g_b\|_{\text{Lip}} \leq \|g_a\|_{\text{Lip}} \cdot \|g_b\|_{\text{Lip}}$ . For instance, in the linear case  $\|W_1 W_2\|_2 \leq \|W_1\|_2 \|W_2\|_2$  is tight only if the first right-singular vector of  $W_1$  aligns with the first left-singular vector of  $W_2$ .

In general, computing the exact global Lipschitz constant of a DNN is NP-hard, and there is a well-known trade-off between tightness and computational efficiency when computing a Lipschitz upper bound (Fazlyab et al., 2019). For example, Fazlyab et al. (2019) gives a tighter estimation based on SDP (semidefinite programming). However, those optimization-based methods are in general much more computationally heavy than spectral normalization and very hard to be incorporated within the training process. Moreover, since the DNNs implemented in this thesis are in general small, the looseness from spectral normalization is acceptable.

**Remark (robustness from spectral normalization).** Although our motivation of introducing spectral normalization is to guarantee control stability, it is worth noting that another benefit from spectral normalization is that spectrally normalized DNN is robust to input disturbance. Namely, since  $g$  is  $\gamma$ -Lipschitz continuous,  $\|g(x + \Delta x) - g(x)\|$  is always bounded by  $\gamma\|\Delta x\|$ .

## 2.5 Preliminaries on Nonlinear Control Theory

### Comparison Lemma

In Lyapunov stability analysis, we often use the following comparison lemma to bound a solution of an ODE (ordinary differential equation).

**Lemma 2.2.** *[adopted from (Khalil, 2002)] Suppose that a continuously differentiable function  $a(t) \in \mathbb{R}_{\geq 0} \mapsto \mathbb{R}$  satisfies the following inequality:*

$$\dot{a}(t) \leq -b \cdot a(t) + c, \quad \forall t \geq 0, \quad a(0) = a_0$$

where  $b > 0$ . Then we have

$$a(t) \leq a_0 e^{-bt} + \frac{c}{b}(1 - e^{-bt}), \quad \forall t \geq 0.$$

The first application of Lemma 2.2 will be analyzing the convergence of the position tracking error  $\tilde{q} = q - q_d$  when the composite velocity error  $s = \dot{\tilde{q}} + \Lambda \tilde{q}$  converges.

**Lemma 2.3.** *Suppose  $\|s(t)\| \leq a(t)$ . Then we have*

$$\|\tilde{q}(t)\| \leq \|\tilde{q}(0)\|e^{-\lambda_{\min}(\Lambda)t} + \frac{a(t)}{\lambda_{\min}(\Lambda)}(1 - e^{-\lambda_{\min}(\Lambda)t}).$$

*Proof.* Define a Lyapunov function as  $V(t) = \frac{1}{2}\|\tilde{q}\|^2$ . Then we have

$$\dot{V} = -\tilde{q}^\top \Lambda \tilde{q} + \tilde{q}^\top s \leq -\lambda_{\min}(\Lambda)\|\tilde{q}\|^2 + a\|\tilde{q}\| = -2\lambda_{\min}(\Lambda)V + a\sqrt{2V}.$$

Define  $W = \sqrt{V}$  and  $\dot{W} = \frac{\dot{V}}{2\sqrt{V}}$ . Then we have

$$\dot{W} \leq -\lambda_{\min}(\Lambda)W + \frac{a}{\sqrt{2}}.$$

Applying Lemma 2.2 we have

$$\|\tilde{q}(t)\| \leq \|\tilde{q}(0)\|e^{-\lambda_{\min}(\Lambda)t} + \frac{a(t)}{\lambda_{\min}(\Lambda)}(1 - e^{-\lambda_{\min}(\Lambda)t}).$$

□

## References

- Bartlett, Peter L., Dylan J. Foster, and Matus J. Telgarsky (2017). *Spectrally-normalized margin bounds for neural networks*. In: *Advances in Neural Information Processing Systems*, pp. 6240–6249.
- Fazlyab, Mahyar, Alexander Robey, Hamed Hassani, Manfred Morari, and George Pappas (2019). *Efficient and accurate estimation of Lipschitz constants for deep neural networks*. In: *Advances in Neural Information Processing Systems* 32.
- Hanover, Drew, Philipp Foehn, Sihao Sun, Elia Kaufmann, and Davide Scaramuzza (2021). *Performance, precision, and payloads: Adaptive nonlinear MPC for quadrotors*. In: *IEEE Robotics and Automation Letters* 7.2, pp. 690–697.
- Khalil, Hassan K. (2002). *Nonlinear systems*. Pearson Education. Prentice Hall. ISBN: 9780130673893.
- Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2012). *Imagenet classification with deep convolutional neural networks*. In: *Advances in Neural Information Processing Systems* 25.
- Lopez, Brett T. and Jean-Jacques E. Slotine (2021). *Universal adaptive control of nonlinear systems*. In: *IEEE Control Systems Letters* 6, pp. 1826–1830.
- Miyato, Takeru, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida (2018). *Spectral normalization for generative adversarial networks*. In: *arXiv preprint arXiv:1802.05957*.

- Morgan, Daniel, Giri P. Subramanian, Soon-Jo Chung, and Fred Y. Hadaegh (2016). *Swarm assignment and trajectory optimization using variable-swarm, distributed auction assignment and sequential convex programming*. In: *The International Journal of Robotics Research* 35.10, pp. 1261–1285.
- Murray, Richard M., Zexiang Li, and S. Shankar Sastry (Dec. 2017). *A mathematical introduction to robotic manipulation*. 1st ed. CRC Press. ISBN: 978-1-315-13637-0. DOI: 10.1201/9781315136370. URL: <https://www.taylorfrancis.com/books/9781351469791>.
- O’Connell, Michael, Guanya Shi, Xichen Shi, Kamyar Azizzadenesheli, Anima Anandkumar, Yisong Yue, and Soon-Jo Chung (2022). *Neural-Fly enables rapid learning for agile flight in strong winds*. In: *Science Robotics* 7.66, eabm6597. DOI: 10.1126/scirobotics.abm6597.
- Salimans, Tim and Diederik P. Kingma (2016). *Weight normalization: A simple reparameterization to accelerate training of deep neural networks*. In: *Advances in Neural Information Processing Systems*, pp. 901–909.
- Shi, Guanya, Wolfgang Hönig, Xichen Shi, Yisong Yue, and Soon-Jo Chung (2022). *Neural-Swarm2: Planning and control of heterogeneous multirotor swarms using learned interactions*. In: *IEEE Transactions on Robotics* 38.2, pp. 1063–1079. DOI: 10.1109/TRO.2021.3098436.
- Shi, Guanya, Wolfgang Hönig, Yisong Yue, and Soon-Jo Chung (2020). *Neural-Swarm: Decentralized close-proximity multirotor control using learned interactions*. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3241–3247. DOI: 10.1109/ICRA40945.2020.9196800.
- Shi, Guanya, Xichen Shi, Michael O’Connell, Rose Yu, Kamyar Azizzadenesheli, Animashree Anandkumar, Yisong Yue, and Soon-Jo Chung (2019). *Neural Lander: Stable drone landing control using learned dynamics*. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 9784–9790. DOI: 10.1109/ICRA.2019.8794351.
- Shi, Xichen, Kyunam Kim, Salar Rahili, and Soon-Jo Chung (2018). *Nonlinear control of autonomous flying cars with wings and distributed electric propulsion*. In: *2018 IEEE Conference on Decision and Control (CDC)*. IEEE, pp. 5326–5333.
- Slotine, Jean-Jacques E. and Weiping Li (1991). *Applied nonlinear control*. Vol. 199. 1. Prentice Hall, Englewood Cliffs, NJ.
- Tal, Ezra and Sertac Karaman (May 2021). *Accurate tracking of aggressive quadrotor trajectories using incremental nonlinear dynamic inversion and differential flatness*. In: *IEEE Transactions on Control Systems Technology* 29.3, pp. 1203–1218. ISSN: 1558-0865. DOI: 10.1109/TCST.2020.3001117.

## Chapter 3

## NEURAL-LANDER

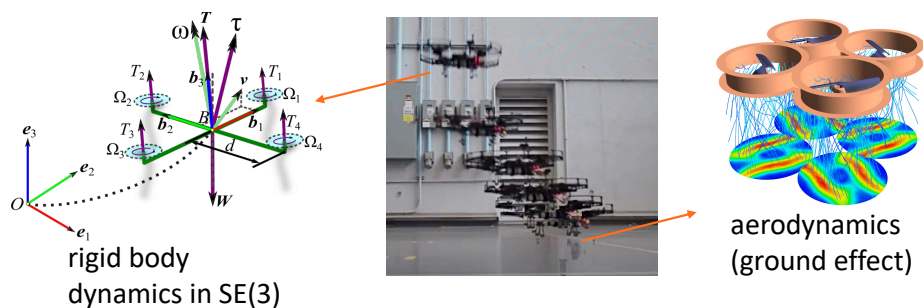


Figure 3.1: Neural-Lander enables agile flight maneuver very close to the ground.

In Chapter 2, we introduced a general mixed robot dynamics model  $M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = Bu + f$  where  $f$  is the unknown dynamics. We also introduced the corresponding drone dynamics model in Eq. (2.4). In this chapter, we focus on the case  $f = f(q, \dot{q}, u)$ , and the main challenge is to ensure nonlinear stability and robustness when using a DNN to approximate  $f$ . In drone experiments, the nominal part refers to the standard rigid body dynamics in SE(3) and  $f$  refers to the unknown aerodynamics such as the ground effect when the drone is near-ground (see Fig. 3.1). This chapter is mainly based on the following paper<sup>1 2</sup>:

Shi, Guanya, Xichen Shi, Michael O’Connell, Rose Yu, Kamyar Azizzadenesheli, Animashree Anandkumar, Yisong Yue, and Soon-Jo Chung (2019). *Neural Lander: Stable drone landing control using learned dynamics*. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 9784–9790. DOI: 10.1109/ICRA.2019.8794351.

---

**Abstract.** Precise near-ground trajectory control is difficult for multi-rotor drones, due to the complex aerodynamic effects caused by interactions between multi-rotor airflow and the environment. Conventional control methods often fail to properly account for these complex effects and fall short in accomplishing smooth landing.

<sup>1</sup>Summary video: <https://youtu.be/FLLsG0S78ik>

<sup>2</sup>Simulator code: [https://github.com/GuanyaShi/neural\\_lander\\_sim\\_1d](https://github.com/GuanyaShi/neural_lander_sim_1d)

In this chapter, we present a novel deep-learning-based robust nonlinear controller (*Neural-Lander*) that improves control performance of a quadrotor during landing. Our approach combines a nominal dynamics model with a Deep Neural Network (DNN) that learns high-order interactions. We apply spectral normalization (SN) to constrain the Lipschitz constant of the DNN. Leveraging this Lipschitz property, we design a nonlinear feedback linearization controller using the learned model and prove system stability with disturbance rejection. To the best of our knowledge, this is the first DNN-based nonlinear feedback controller with stability guarantees that can utilize arbitrarily large neural nets. Experimental results demonstrate that the proposed controller significantly outperforms a Baseline Nonlinear Tracking Controller in both landing and cross-table trajectory tracking cases. We also empirically show that the DNN generalizes well to unseen data outside the training domain.

### 3.1 Introduction and Related Work

Unmanned Aerial Vehicles (UAVs) require high precision control of aircraft positions, especially during landing and take-off. This problem is challenging largely due to complex interactions of rotor and wing airflows with the ground. The aerospace community has long identified such ground effect that can cause an increased lift force and a reduced aerodynamic drag. These effects can be both helpful and disruptive in flight stability (Cheeseman and Bennett, 1955), and the complications are exacerbated with multiple rotors. Therefore, performing automatic landing of UAVs is risk-prone, and requires expensive high-precision sensors as well as carefully designed controllers.

Compensating for ground effect is a long-standing problem in the aerial robotics community. Prior work has largely focused on mathematical modeling (e.g. Nonaka and Sugizaki (2011)) as part of system identification (ID). These models are later used to approximate aerodynamics forces during flights close to the ground and combined with controller design for feed-forward cancellation (e.g. Danjun et al. (2015)). However, existing theoretical ground effect models are derived based on steady-flow conditions, whereas most practical cases exhibit unsteady flow. Alternative approaches, such as integral or adaptive control methods, often suffer from slow response and delayed feedback. Berkenkamp, Schoellig, et al. (2016) employs Bayesian Optimization for open-air control but not for take-off/landing. Given these limitations, the precision of existing fully automated systems for UAVs are still insufficient for landing and take-off, thereby necessitating the guidance of a human UAV operator during those phases.



To capture complex aerodynamic interactions without overly-constrained by conventional modeling assumptions, we take a machine-learning (ML) approach to build a black-box ground effect model using Deep Neural Networks (DNNs). However, incorporating such models into a UAV controller faces three key challenges. First, it is challenging to collect sufficient real-world training data, as DNNs are notoriously data-hungry. Second, due to high-dimensionality, DNNs can be unstable and generate unpredictable output, which makes the system susceptible to instability in the feedback control loop. Third, DNNs are often difficult to analyze, which makes it difficult to design provably stable DNN-based controllers.

The aforementioned challenges pervade previous works using DNNs to capture high-order non-stationary dynamics. For example, Abbeel et al. (2010) and Punjani and Abbeel (2015) use DNNs to improve system ID of helicopter aerodynamics, but not for controller design. Other approaches aim to generate reference inputs or trajectories from DNNs (Bansal et al., 2016; Li et al., 2017; Zhou et al., 2017; Sánchez-Sánchez and Izzo, 2018). However, these approaches can lead to challenging optimization problems (Bansal et al., 2016), or heavily rely on well-designed closed-loop controller and require a large number of labeled training data (Li et al., 2017; Zhou et al., 2017; Sánchez-Sánchez and Izzo, 2018). A more classical approach of using DNNs is direct inverse control (Frye and Provence, 2014; Suprijono and Kusumoputro, 2017) but the non-parametric nature of a DNN controller also makes it challenging to guarantee stability and robustness to noise. Berkenkamp, Turchetta, et al. (2017) proposes a provably stable model-based Reinforcement Learning method based on Lyapunov analysis, but it requires a potentially expensive discretization step and relies on the native Lipschitz constant of the DNN.

### **Contributions**

In this chapter, we propose a learning-based controller, *Neural-Lander*, to improve the precision of quadrotor landing with guaranteed stability. Our approach directly learns the ground effect on coupled unsteady aerodynamics and vehicular dynamics. We use deep learning for system ID of residual dynamics and then integrate it with nonlinear feedback linearization control.

We train DNNs with layer-wise spectrally normalized weight matrices. We prove that the resulting controller is globally exponentially stable under bounded learning errors. This is achieved by exploiting the Lipschitz bound of spectrally normalized DNNs. It has earlier been shown that spectral normalization of DNNs leads to good

generalization, i.e. stability in a learning-theoretic sense (Miyato et al., 2018). It is intriguing that spectral normalization simultaneously guarantees stability both in a learning-theoretic and a control-theoretic sense.

We evaluate *Neural-Lander* on trajectory tracking of quadrotor during take-off, landing and cross-table maneuvers. *Neural-Lander* is able to land a quadrotor much more accurately than a Baseline Nonlinear Tracking Controller with a pre-identified system. In particular, we show that compared to the baseline, *Neural-Lander* can decrease error in  $z$  axis from 0.13 m to 0, mitigate  $x$  and  $y$  drifts by as much as 90%, in the landing case. Meanwhile, *Neural-Lander* can decrease  $z$  error from 0.153 m to 0.027 m, in the cross-table trajectory tracking task. We also demonstrate that the learned model can handle temporal dependency, and is an improvement over the steady-state theoretical models.

### 3.2 Problem Statement

#### General Robot Dynamics and Drone Dynamics Modelling

As introduced in Chapter 2, we consider the following mixed robot dynamics model:

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = Bu + \underbrace{f(q, \dot{q}, u)}_{\text{unknown}}. \quad (3.1)$$

For the specific drone dynamics, recall Eq. (2.4) in Chapter 2:

$$\begin{aligned} \dot{p} &= v, & m\dot{v} &= mg + Rf_u + f_a, \\ \dot{R} &= RS(\omega), & J\dot{\omega} &= J\omega \times \omega + \tau + \tau_a. \end{aligned}$$

As discussed in Chapter 2, we can reduce the drone dynamics model to Eq. (3.1) using the cascading control structure. Therefore, for theoretical analysis we will focus on the general model in Eq. (3.1) for generality and rotational simplicity. Moreover, in drone experiments, as we mainly focus on landing and take-off tasks, the attitude dynamics is limited and the aerodynamic disturbance torque  $\tau_a$  is negligible. Thus position dynamics and the aerodynamic force  $f_a$  will our primary concern.

#### Learning and Control Goals

We first collect a data set  $\mathcal{D}$  containing  $N$  input and label pairs:

$$x_i = [q_i; \dot{q}_i; u_i], \quad \mathcal{D} = \{x_i, y_i = f(x_i) + \epsilon_i\}_{i=1}^N, \quad (3.3)$$

where we define  $x = [q; \dot{q}; u]$  for simplicity, and  $y_i$  is a noisy measurement of  $f(x_i)$  with  $\epsilon_i$  an extra disturbance.

With  $\mathcal{D}$ , we aim to train a ReLU DNN  $\hat{f}(x, \theta)$  to approximate  $f$ . Then, our goal is to design a stable and robust controller such that the tracking error  $\|q - q_d\|$  is small. Training is done off-line, and the learned dynamics  $\hat{f}$  is applied in the on-board controller in real-time to achieve smooth landing and take-off.

### 3.3 Dynamics Learning and Controller Design

We apply Algorithm 2.1 to train the ReLU DNN  $\hat{f}(x, \theta)$  with a bounded Lipschitz constant  $\gamma$ . We select  $\gamma$  such that  $\gamma \cdot \|B^\dagger\| < 1$  where  $B^\dagger$  is the Moore-Penrose inverse of  $B$ . Thus from Lemma 2.1, we have

$$\|\hat{f}\|_{\text{Lip}} \leq \gamma, \quad \gamma \cdot \|B^\dagger\| < 1.$$

With  $\hat{f}$ , the controller structure is given as Eq. (2.3):

$$Bu = \underbrace{M(q)\ddot{q}_r + C(q, \dot{q})\dot{q}_r + g(q)}_{\text{nonlinear feedforward}} \underbrace{-Ks}_{\text{linear feedback}} \underbrace{-\hat{f}(q, \dot{q}, u)}_{\text{learned feedforward}}. \quad (3.4)$$

Because of the dependency of  $\hat{f}$  on  $u$ , the control synthesis problem in Eq. (3.4) is non-affine. Therefore, we propose the following fixed-point iteration method for solving Eq. (3.4):

$$u_k = B^\dagger \left( M(q)\ddot{q}_r + C(q, \dot{q})\dot{q}_r + g(q) - Ks - \hat{f}(q, \dot{q}, u_{k-1}) \right), \quad (3.5)$$

where  $u_k$  and  $u_{k-1}$  are the control input for current and previous time step in the discrete-time controller. In the next section, we prove the stability of the system and convergence of the control inputs in Eq. (3.5).

### 3.4 Nonlinear Stability Analysis

In this section, the closed-loop tracking error analysis provides a direct correlation on how to tune the neural network  $\hat{f}$  and controller parameter to improve control performance and robustness.

#### Control Allocation as Contraction Mapping

We first show that the control input  $u_k$  converges to the solution of Eq. (3.4) when all states are fixed.

**Lemma 3.1.** *Define mapping  $u_k = \mathcal{F}(u_{k-1})$  based on Eq. (3.5) and fix all current states. If  $\hat{f}(x)$  is  $\gamma$ -Lipschitz continuous, and  $\|B^\dagger\|_2 \cdot \gamma < 1$ ; then  $\mathcal{F}(\cdot)$  is a contraction mapping, and  $u_k$  converges to unique solution of  $u^* = \mathcal{F}(u^*)$ .*

*Proof.*  $\forall u_1, u_2 \in \mathbb{R}^m$  and given fixed states as  $q, \dot{q}, \dot{q}_r, \ddot{q}_r, s$ , then:

$$\|\mathcal{F}(u_1) - \mathcal{F}(u_2)\|_2 = \left\| B^\dagger \left( \hat{f}(q, \dot{q}, u_1) - \hat{f}(q, \dot{q}, u_2) \right) \right\|_2 \leq \|B^\dagger\|_2 \cdot \gamma \|u_1 - u_2\|_2.$$

Thus,  $\|B^\dagger\|_2 \cdot \gamma < 1$  so  $\mathcal{F}(\cdot)$  is a contraction mapping.  $\square$

### Stability of Learning-based Nonlinear Controller

Before continuing to prove the stability of the full system, we make the following assumptions.

**Assumption 3.1** (boundedness of the desired trajectory). *The desired states along the position trajectory  $q_d(t)$ ,  $\dot{q}_d(t)$ , and  $\ddot{q}_d(t)$  are bounded.*

**Assumption 3.2.** *One-step difference of control signal satisfies  $\|u_k - u_{k-1}\| \leq \rho \|s\|$  with a small positive  $\rho$ .*

Namely, we assume the control signal is changing smoothly. Here we provide the intuition behind this assumption. From Eq. (3.5), we can derive the following approximate relation with  $\Delta(\cdot)_k = \|(\cdot)_k - (\cdot)_{k-1}\|$ :

$$\begin{aligned} \Delta u_k \leq \|B^\dagger\| & \left( \gamma \Delta u_{k-1} + \gamma \Delta q_k + \gamma \Delta \dot{q}_k + \|K\| \Delta s_k \right. \\ & \left. + \Delta(M(q)\ddot{q}_r)_k + \Delta(C(q, \dot{q})\dot{q}_r)_k + \Delta(g(q))_k \right). \end{aligned}$$

Note that for drone position controller,  $C = 0$  and  $M$  and  $g$  are both constant (see more details in Chapter 2). Moreover, because update rate of the attitude controller ( $> 200$  Hz) and motor speed control ( $> 5$  kHz) are much higher than that of the position controller ( $\approx 50$  Hz) and the desired trajectory  $q_d(t)$  is not changing too fast, in practice, we can safely neglect  $\Delta s_k$ ,  $\Delta q_k$ ,  $\Delta \dot{q}_k$  and  $\Delta \ddot{q}_r$  in one update (Theorem 11.1 Khalil (2002)). It leads to:

$$\Delta u_k \leq \|B^\dagger\| (\gamma \Delta u_{k-1} + c),$$

with  $c$  being a small constant and  $\|B^\dagger\| \cdot \gamma < 1$ , we can deduce that  $\Delta u$  rapidly converges to a small ultimate bound between each position controller update.

**Assumption 3.3** (bounded learning error). *The learning error of  $\hat{f}(x)$  over the compact sets  $x \in \mathcal{X}$  is upper bounded by  $\epsilon_m = \sup_{x \in \mathcal{X}} \|f(x) - \hat{f}(x)\|$ .*

DNNs have been shown to generalize well to the set of unseen events that are from almost the same distribution as training set (Zhang et al., 2016; He et al.,

2016). This empirical observation is also theoretically studied in order to shed more light toward an understanding of the complexity of these models (Neyshabur, Bhojanapalli, McAllester, and Nathan Srebro, 2017; Bartlett et al., 2017; Dziugaite and Roy, 2017; Neyshabur, Bhojanapalli, McAllester, and Nati Srebro, 2017).

Based on the above assumptions, we can now present our overall stability and robustness result.

**Theorem 3.1.** *Under Assumptions 3.1 to 3.3, for a time-varying  $q_d(t)$ , the controller defined in Eq. (3.5) with  $\lambda_{\min}(K) > \gamma\rho$  and  $\gamma\|B^\dagger\| < 1$  achieves the exponential convergence of the tracking error  $\|\tilde{q}\| = \|q - q_d\|$  to the following error ball*

$$\frac{\lambda_{\max}(M)}{\lambda_{\min}(M)\lambda_{\min}(\Lambda)(\lambda_{\min}(K) - \gamma\rho)} \cdot \epsilon_m.$$

*Proof.* Plugging Eq. (3.5) into Eq. (3.1), we have the following closed-loop dynamics

$$M\dot{s} + Cs + Ks = f(q, \dot{q}, u_k) - \hat{f}(q, \dot{q}, u_{k-1}).$$

Note that this is a hybrid system in the sense that  $u$  evolves in a discrete manner but other variables changes continuously. We define the Lyapunov function as  $V(s) = \frac{1}{2}s^\top Ms$ , and after taking the derivative we have

$$\begin{aligned} \dot{V} &= \frac{1}{2}s^\top \dot{M}s + s^\top \left( -Cs - Ks + f(q, \dot{q}, u_k) - \hat{f}(q, \dot{q}, u_{k-1}) \right) \\ &\stackrel{(1)}{=} -s^\top Ks + s^\top \left( f(q, \dot{q}, u_k) - \hat{f}(q, \dot{q}, u_k) + \hat{f}(q, \dot{q}, u_k) - \hat{f}(q, \dot{q}, u_{k-1}) \right) \\ &\stackrel{(2)}{\leq} -s^\top Ks + \|s\| (\epsilon_m + \gamma\|u_k - u_{k-1}\|) \\ &\stackrel{(3)}{\leq} -s^\top Ks + \gamma\rho\|s\|^2 + \epsilon_m\|s\| \\ &\leq -(\lambda_{\min}(K) - \gamma\rho)\|s\|^2 + \epsilon_m\|s\| \end{aligned}$$

where (1) is from the fact  $\dot{M} - 2C$  is skew-symmetric, (2) is from Assumption 3.3 and the Lipschitz property of  $\hat{f}$ , and (3) is from Assumption 3.2. Note that  $\frac{1}{2}\lambda_{\max}(M)\|s\|^2 \geq V \geq \frac{1}{2}\lambda_{\min}(M)\|s\|^2$ , we have

$$\dot{V} \leq -(\lambda_{\min}(K) - \gamma\rho)\frac{2V}{\lambda_{\max}(M)} + \epsilon_m\sqrt{\frac{2V}{\lambda_{\min}(M)}}.$$

Then define  $W(t) = \sqrt{V(t)}$ . Note that  $\dot{W} = \frac{\dot{V}}{2\sqrt{V}}$ . Therefore we have

$$\dot{W} \leq -\frac{\lambda_{\min}(K) - \gamma\rho}{\lambda_{\max}(M)}W + \epsilon_m\sqrt{\frac{1}{2\lambda_{\min}(M)}}.$$

Using the comparison lemma (Lemma 2.2), we have

$$W \leq \exp\left(-\frac{\lambda_{\min}(K) - \gamma\rho}{\lambda_{\max}(M)}t\right)W(0) + \frac{\epsilon_m \lambda_{\max}(M)}{\lambda_{\min}(K) - \gamma\rho} \sqrt{\frac{1}{2\lambda_{\min}(M)}}.$$

Recall that  $W = \sqrt{V} \geq \|s\| \sqrt{\frac{\lambda_{\min}(M)}{2}}$ , so we finally have

$$\|s\| \leq \exp\left(-\frac{\lambda_{\min}(K) - \gamma\rho}{\lambda_{\max}(M)}t\right) \sqrt{\frac{\lambda_{\max}(M)}{\lambda_{\min}(M)}} \|s(0)\| + \frac{\lambda_{\max}(M)}{\lambda_{\min}(M)(\lambda_{\min}(K) - \gamma\rho)} \cdot \epsilon_m$$

It can be shown that this leads to finite-gain  $\mathcal{L}_p$  stability and input-to-state stability (ISS) (Chung et al., 2013). Further, with Lemma 2.3, we have  $\|\tilde{q}\|$  exponentially converges to the following error ball

$$\frac{\lambda_{\max}(M)}{\lambda_{\min}(M)\lambda_{\min}(\Lambda)(\lambda_{\min}(K) - \gamma\rho)} \cdot \epsilon_m.$$

□

### Interpretation of Theorem 3.1

**Potential trade-off when tuning  $\gamma$ .** Note that Theorem 3.1 has two conditions on the Lipschitz constant of the DNN  $\hat{f}$ :  $\lambda_{\min}(K) > \gamma\rho$  and  $\gamma\|B^\dagger\| < 1$ . The latter one is to make sure the fixed-point iteration method will converge, and the former one guarantees the closed-loop stability. They both suggest  $\gamma = \|\hat{f}\|_{\text{Lip}}$  should be small and bigger  $\gamma$  will influence the controller robustness against big  $\rho$  (i.e., bigger delay). However, the learning error  $\epsilon_m$  might increase if  $\gamma$  gets too small, especially when the ground-truth function  $f$  has bad Lipschitz property. In that case, one should look for higher-order methods than the fixed-point iteration (e.g., Newton method or something like MPC) but those methods are in general much more involved and it is unclear how they influence the closed-loop stability analysis. Nevertheless, in our experiments we strictly followed  $\gamma\|B^\dagger\| < 1$  and the learning performance is even better than the baseline method without Lipschitz constraints.

**An alternative assumption on  $\|u_k - u_{k-1}\|$ .** Assumption 3.2 assumes  $\|u_k - u_{k-1}\| \leq \rho\|s\|$ . Another assumption we can consider is  $\|u_k - u_{k-1}\| \leq \bar{\rho}$  where  $\bar{\rho}$  is a small constant. In this case, the Lyapunov analysis will change to

$$\dot{V} \leq -\lambda_{\min}(K)\|s\|^2 + (\epsilon_m + \gamma\bar{\rho})\|s\|$$

and the error ball for  $\tilde{q}$  will be changed to

$$\frac{\lambda_{\max}(M)}{\lambda_{\min}(M)\lambda_{\min}(\Lambda)\lambda_{\min}(K)} \cdot (\epsilon_m + \gamma\bar{\rho}).$$

With this alternative assumption,  $\gamma\bar{\rho}$  can be viewed as an add-on term on the learning error. Similarly, bigger  $\gamma$  amplifies the delay constant  $\bar{\rho}$ . Moreover, it is worth noting that with this alternative assumption we no longer need to require the gain condition about  $\lambda_{\min}(K)$ .

### 3.5 Experiments

In our experiments, we evaluate both the generalization performance of our DNN as well as the overall control performance of *Neural-Lander*. The experimental setup is composed of a motion capture system with 17 cameras, a WiFi router for communication, and an Intel Aero drone, weighing 1.47 kg with an onboard Linux computer (2.56 GHz Intel Atom x7 processor, 4 GB DDR3 RAM). We retrofitted the drone with eight reflective infrared markers for accurate position, attitude and velocity estimation at 100Hz. The Intel Aero drone and the test space are shown in Fig. 3.1.

In this section, we will focus on the drone dynamics given in Eq. (2.4) and Eq. (2.5). We apply the proposed nonlinear learning-based controller Eq. (3.5) to the quadrotor position dynamics Eq. (2.4a) via the reduction in Eq. (2.6). In particular,  $\hat{f}_a$  in Eq. (2.6) is a ReLU DNN depending on the rotor control signal  $u$ . We call this drone position controller *Neural-Lander*. See the original paper (Shi et al., 2019) for the detailed structure and analysis of *Neural-Lander* in quadrotor dynamics.

#### Bench Test

To identify a good nominal model, we first measured the mass  $m$ , diameter of the rotor  $D_r$ , the air density  $\rho_a$ , and the gravity  $g$ . Then we performed bench test to determine the thrust constant,  $c_T$ , as well as the non-dimensional thrust coefficient  $C_T = \frac{c_T}{\rho_a D_r^4}$ . Note that  $C_T$  is a function of propeller speed  $n$ , and here we picked a nominal value at  $n = 2000$  RPM.

#### Real-World Flying Data and Preprocessing

To estimate the disturbance force  $f_a$ , an expert pilot manually flew the drone at different heights, and we collected training data consisting of sequences of state estimates and control inputs  $\{(p, v, R, u), y\}$  where  $u = [n_1^2; n_2^2; n_3^2; n_4^2]$  and  $y$  is the observed value of  $f_a$ .

We utilized the relation  $f_a = m\dot{v} - mg - Rf_u$  from Eq. (2.4a) to calculate  $f_a$ , where  $f_u$  is calculated based on the nominal  $c_T$  from the bench test. For the acceleration  $\dot{v}$ , we used the fifth-order numerical differentiation method. Our training set is a

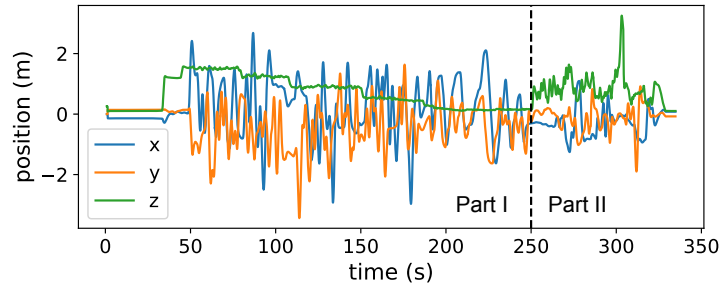


Figure 3.2: Training data trajectory. Part I (0 to 250 s) contains maneuvers at different heights (0.05 m to 1.50 m). Part II (250 s to 350 s) includes random  $x$ ,  $y$ , and  $z$  motions for maximum state-space coverage.

single continuous trajectory with varying heights and velocities. The trajectory has two parts shown in Fig. 3.2. We aim to learn the ground effect through Part I of the training set, and other aerodynamics forces such as air drag through Part II.

### DNN Prediction Performance

We train a deep ReLU network  $\hat{f}_a(z, v, R, u)$ , with  $z, v, R, u$  corresponding to global height, global velocity, attitude, and control input. We build the ReLU network using PyTorch. Our ReLU network consists of four fully-connected hidden layers, with input and the output dimensions 12 and 3, respectively. We use spectral normalization (see Algorithm 2.1) to constrain the Lipschitz constant of the DNN.

We compare the near-ground estimation accuracy our DNN model with existing 1D steady ground effect model (Cheeseman and Bennett, 1955; Danjun et al., 2015):

$$T(n, z) = \frac{n^2}{1 - \mu \left(\frac{D_r}{8z}\right)^2} c_T(n) = n^2 c_T(n_0) + \bar{f}_{a,z}, \quad (3.6)$$

where  $T$  is the thrust generated by propellers,  $n$  is the rotation speed,  $n_0$  is the idle RPM, and  $\mu$  depends on the number and the arrangement of propellers ( $\mu = 1$  for a single propeller, but must be tuned for multiple propellers). Note that  $c_T$  is a function of  $n$ . Thus, we can derive  $\bar{f}_{a,z}(n, z)$  from  $T(n, z)$ .

Figure 3.3(a) shows the comparison between the estimation of  $f_a$  from the DNN  $\hat{f}_a$  and the theoretical ground effect model Eq. (3.6) at different  $z$  (assuming  $T = mg$  when  $z = \infty$ ). We can see that our DNN can achieve much better estimates than the theoretical ground effect model. We further investigate the trend of  $\bar{f}_{a,z}$  with respect to the rotation speed  $n$ . Fig. 3.3(b) shows the learned  $\hat{f}_{a,z}$  over the rotation speed  $n$  at a given height, in comparison with the  $C_T$  measured from the bench test. We



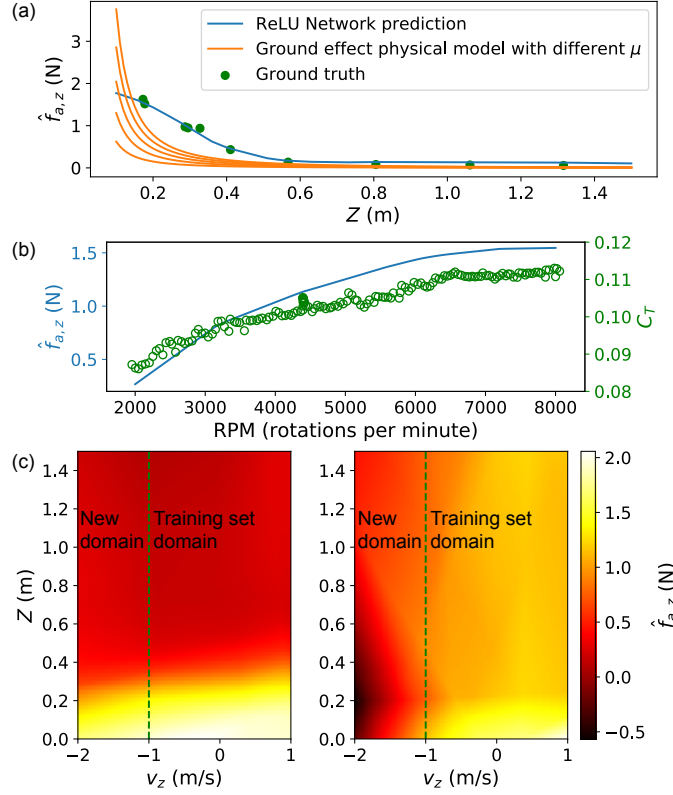


Figure 3.3: DNN prediction performance. (a) Learned  $f_{a,z}$  compared to the ground effect model with respect to height  $z$ , with  $v_z = v_x = v_y = 0$  m/s,  $R = I$ ,  $u = 6400$  RPM. Ground truth points are from hovering data at different heights. (b) Learned  $f_{a,z}$  with respect to rotation speed  $n$  ( $z = 0.2$  m,  $v_z = 0$  m/s), compared to  $C_T$  measured in the bench test. (c) Heatmaps of learned  $f_{a,z}$  versus  $z$  and  $v_z$ . (Left) ReLU network with spectral normalization. (Right) ReLU network without spectral normalization.

observe that the increasing trend of the estimates  $\hat{f}_{a,z}$  is consistent with bench test results for  $C_T$ .

To understand the benefits of SN, we compared  $\hat{f}_{a,z}$  predicted by the DNNs trained both with and without SN as shown in Fig. 3.3(c). Note that  $v_z$  from  $-1$  m/s to  $1$  m/s is covered in our training set, but  $-2$  m/s to  $-1$  m/s is not. We observe the following differences:

1. Ground effect:  $\hat{f}_{a,z}$  increases as  $z$  decreases, which is also shown in Fig. 3.3(a).
2. Air drag:  $\hat{f}_{a,z}$  increases as the drone goes down ( $v_z < 0$ ) and it decreases as the drone goes up ( $v_z > 0$ ).

3. Generalization: the spectral normalized DNN is much smoother and can also generalize to new input domains not contained in the training set.

In Bartlett et al. (2017), the authors theoretically show that spectral normalization can provide tighter generalization guarantees on unseen data, which is consistent with our empirical observation. We will connect generalization theory more tightly with our robustness guarantees in the future.

### Baseline Controller

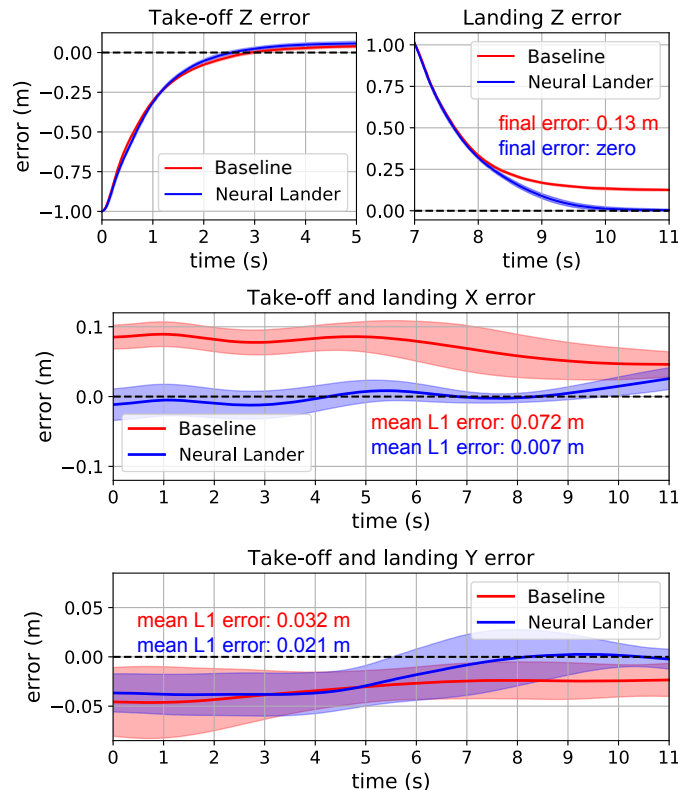


Figure 3.4: Baseline Controller and *Neural-Lander* performance in take-off and landing. Means (solid curves) and standard deviations (shaded areas) of 10 trajectories.

We compared the *Neural-Lander* with a Baseline Nonlinear Tracking Controller. We implemented both a Baseline Controller similar to Eq. (2.6) with  $\hat{f}_a \equiv 0$ , as well as an integral controller variant with  $s = \dot{\tilde{p}} + 2\Lambda\tilde{p} + \Lambda^2 \int_0^t \tilde{p}(\tau)d\tau$ . Though an integral gain can cancel steady-state error during set-point regulation, our flight results showed that the performance can be sensitive to the integral gain, especially during trajectory tracking. This can be seen in the demo video<sup>3</sup>.

<sup>3</sup>Demo videos: <https://youtu.be/FLLsG0S78ik>

## Setpoint Regulation Performance

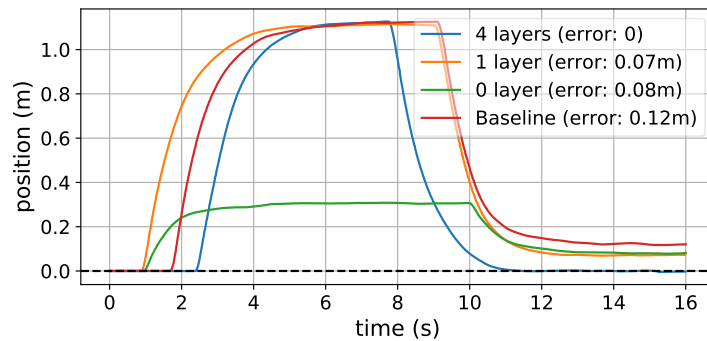


Figure 3.5: *Neural-Lander* performance in take-off and landing with different DNN capacities. 1 layer means a linear mapping  $\hat{f}_a = Ax + b$ ; 0 layer means  $\hat{f}_a = b$ ; Baseline means  $\hat{f}_a \equiv 0$ .

First, we tested the two controllers’ performance in take-off/landing, by commanding position setpoint  $p_d$ , from  $(0, 0, 0)$ , to  $(0, 0, 1)$ , then back to  $(0, 0, 0)$ , with  $\dot{p}_d \equiv 0$ . From Fig. 3.4, we can conclude that there are two main benefits of our *Neural-Lander*. (a) *Neural-Lander* can control the drone to precisely and smoothly land on the ground surface while the Baseline Controller struggles to achieve 0 terminal height due to the ground effect. (b) *Neural-Lander* can mitigate drifts in  $x - y$  plane, as it also learned about additional aerodynamics such as air drag.

Second, we tested *Neural-Lander* performance with different DNN capacities. Fig. 3.5 shows that compared to the baseline ( $\hat{f}_a \equiv 0$ ), 1 layer model could decrease  $z$  error but it is not enough to land the drone. 0 layer model generated significant error during take-off.

In experiments, we observed the *Neural-Lander* without spectral normalization can even result in unexpected controller outputs leading to crash, which empirically implies the necessity of SN in training the DNN and designing the controller.

## Trajectory Tracking Performance

To show that our algorithm can handle more complicated environments where physics-based modelling of dynamics would be substantially more difficult, we devise a task of tracking an elliptic trajectory very close to a table with a period of 10 seconds shown in Fig. 3.6. The trajectory is partially over the table with significant ground effects, and a sharp transition to free space at the edge of the table. We compared the performance of both *Neural-Lander* and Baseline Controller on this test.

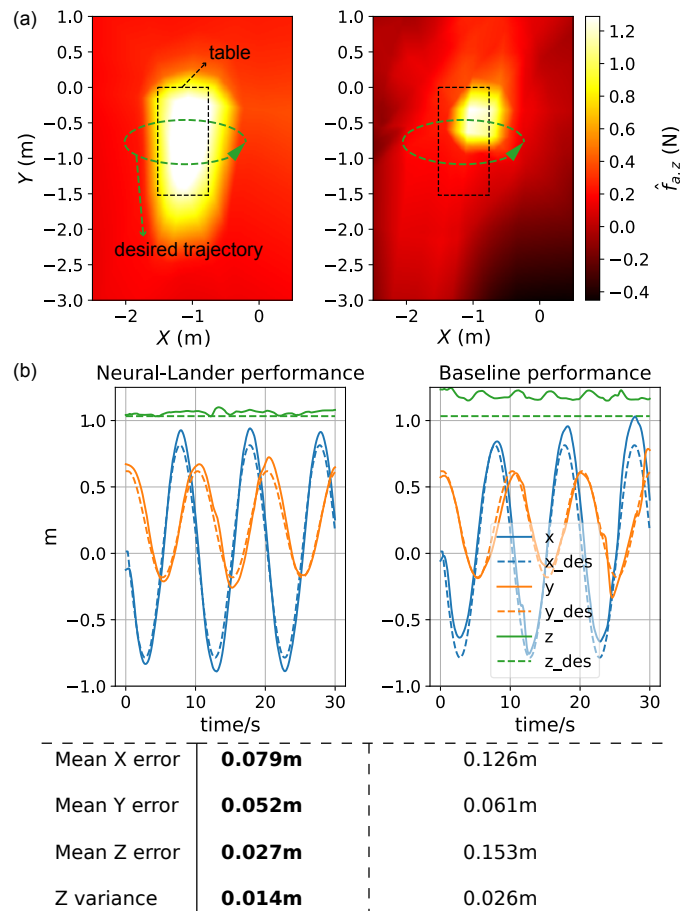


Figure 3.6: Heatmaps of learned  $\hat{f}_{a,z}$  and trajectory tracking performance. (a) Heatmaps of learned  $\hat{f}_{a,z}$  versus  $x$  and  $y$ , with other inputs fixed. (Left) ReLU network with spectral normalization. (Right) ReLU network without spectral normalization. (b) Tracking performance and statistics.

In order to model the complex dynamics near the table, we manually flew the drone in the space close to the table to collect another data set. We trained a new ReLU DNN model with  $x$ - $y$  positions as additional input features:  $\hat{f}_a(p, v, R, u)$ . Similar to the setpoint experiment, the benefit of spectral normalization can be seen in Fig. 3.6(a), where only the spectrally-normalized DNN exhibits a clear table boundary.

Fig. 3.6(b) shows that *Neural-Lander* outperformed the Baseline Controller for tracking the desired position trajectory in all  $x$ ,  $y$ , and  $z$  axes. Additionally, *Neural-Lander* showed a lower variance in height, even at the edge of the table, as the controller captured the changes in ground effects when the drone flew over the table.

In summary, the experimental results with multiple ground interaction scenarios show that much smaller tracking errors are obtained by *Neural-Lander*, which

is essentially the nonlinear tracking controller with feedforward cancellation of a spectrally-normalized DNN.

## References

- Abbeel, Pieter, Adam Coates, and Andrew Y. Ng (2010). *Autonomous helicopter aerobatics through apprenticeship learning*. In: *The International Journal of Robotics Research* 29.13, pp. 1608–1639.
- Bansal, Somil, Anayo K. Akametalu, Frank J. Jiang, Forrest Laine, and Claire J. Tomlin (2016). *Learning quadrotor dynamics using neural network for flight control*. In: *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE, pp. 4653–4660.
- Bartlett, Peter L., Dylan J. Foster, and Matus J. Telgarsky (2017). *Spectrally-normalized margin bounds for neural networks*. In: *Advances in Neural Information Processing Systems*, pp. 6240–6249.
- Berkenkamp, Felix, Angela P. Schoellig, and Andreas Krause (2016). *Safe controller optimization for quadrotors with Gaussian Processes*. In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 493–496. URL: <https://arxiv.org/abs/1509.01066>.
- Berkenkamp, Felix, Matteo Turchetta, Angela P. Schoellig, and Andreas Krause (2017). *Safe model-based reinforcement learning with stability guarantees*. In: *Neural Information Processing Systems (NIPS)*. URL: <https://arxiv.org/abs/1705.08551>.
- Cheeseman, I. C. and W. E. Bennett (1955). *The effect of ground on a helicopter rotor in forward flight*. In: *Citeseer*.
- Chung, Soon-Jo, Saptarshi Bandyopadhyay, Insu Chang, and Fred Y. Hadaegh (2013). *Phase synchronization control of complex networks of Lagrangian systems on adaptive digraphs*. In: *Automatica* 49.5, pp. 1148–1161.
- Danjun, Li, Zhou Yan, Shi Zongying, and Lu Geng (2015). *Autonomous landing of quadrotor based on ground effect modelling*. In: *2015 34th Chinese Control Conference (CCC)*. IEEE, pp. 5647–5652.
- Dziugaite, Gintare Karolina and Daniel M. Roy (2017). *Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data*. In: *arXiv preprint arXiv:1703.11008*.
- Frye, Michael T. and Robert S. Provence (2014). *Direct inverse control using an artificial neural network for the autonomous hover of a helicopter*. In: *2014 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. IEEE, pp. 4121–4122.

- He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). *Deep residual learning for image recognition*. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778.
- Khalil, Hassan K. (2002). *Nonlinear systems*. Pearson Education. Prentice Hall. ISBN: 9780130673893.
- Li, Qiyang, Jingxing Qian, Zining Zhu, Xuchan Bao, Mohamed K. Helwa, and Angela P. Schoellig (2017). *Deep neural networks for improved, impromptu trajectory tracking of quadrotors*. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 5183–5189.
- Miyato, Takeru, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida (2018). *Spectral normalization for generative adversarial networks*. In: *arXiv preprint arXiv:1802.05957*.
- Neyshabur, Behnam, Srinadh Bhojanapalli, David McAllester, and Nathan Srebro (2017). *A PAC-Bayesian approach to spectrally-normalized margin bounds for neural networks*. In: *arXiv preprint arXiv:1707.09564*.
- Neyshabur, Behnam, Srinadh Bhojanapalli, David McAllester, and Nati Srebro (2017). *Exploring generalization in deep learning*. In: *Advances in Neural Information Processing Systems*, pp. 5947–5956.
- Nonaka, Kenichiro and Hirokazu Sugizaki (2011). *Integral sliding mode altitude control for a small model helicopter with ground effect compensation*. In: *American Control Conference (ACC), 2011*. IEEE, pp. 202–207.
- Punjani, Ali and Pieter Abbeel (2015). *Deep learning helicopter dynamics models*. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3223–3230.
- Sánchez-Sánchez, Carlos and Dario Izzo (2018). *Real-time optimal control via Deep Neural Networks: Study on landing problems*. In: *Journal of Guidance, Control, and Dynamics* 41.5, pp. 1122–1135.
- Shi, Guanya, Xichen Shi, Michael O’Connell, Rose Yu, Kamyar Azizzadenesheli, Animashree Anandkumar, Yisong Yue, and Soon-Jo Chung (2019). *Neural Lander: Stable drone landing control using learned dynamics*. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 9784–9790. doi: 10.1109/ICRA.2019.8794351.
- Suprijono, Herwin and Benyamin Kusumoputro (2017). *Direct inverse control based on neural network for unmanned small helicopter attitude and altitude control*. In: *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)* 9.2-2, pp. 99–102.
- Zhang, Chiyuan, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals (2016). *Understanding deep learning requires rethinking generalization*. In: *arXiv preprint arXiv:1611.03530*.

Zhou, Siqu, Mohamed K. Helwa, and Angela P. Schoellig (2017). *Design of deep neural networks as add-on blocks for improving impromptu trajectory tracking*. In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, pp. 5201–5207.

## Chapter 4

## NEURAL-SWARM

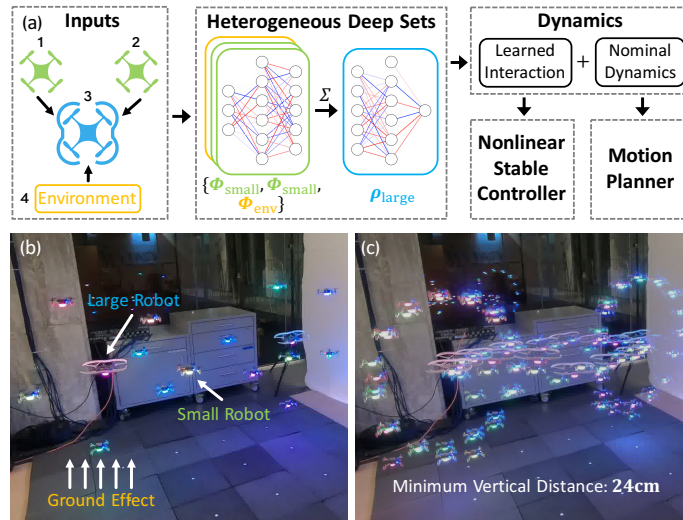


Figure 4.1: Overview of *Neural-Swarm*. We learn complex interaction between multirotors using heterogeneous deep sets and design an interaction-aware nonlinear stable controller and a multi-robot motion planner (a). Our approach enables close-proximity flight (minimum vertical distance 24 cm) of heterogeneous aerial teams (16 robots) with significant lower tracking error compared to solutions that do not consider the interaction forces (b,c).

In Chapter 2, we introduced a general mixed robot dynamics model  $M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = Bu + f$  where  $f$  is the unknown dynamics. We also introduced the corresponding drone dynamics model in Eq. (2.4). In Chapter 3, we discussed the case  $f = f(q, \dot{q}, u)$  where the main challenge is to ensure nonlinear stability and robustness when using a DNN to approximate  $f$ . In this chapter, we will study a multi-agent interactive setting, where

$$f = f(q, \dot{q}, \text{neighbors' states})$$

and the main challenge is to design a scalable learning and control method which can generalize to different numbers of agents. For example, in multirotor swarm experiments in Fig. 4.1,  $f$  refers to complex aerodynamic interactions between multirotors. In addition to learning-based control methods, we will also discuss



how to deal with motor delays and design learning-based motion planners. This chapter is mainly based on the following papers<sup>1 2</sup>:

Shi, Guanya, Wolfgang Hönig, Xichen Shi, Yisong Yue, and Soon-Jo Chung (2022). *Neural-Swarm2: Planning and control of heterogeneous multirotor swarms using learned interactions*. In: *IEEE Transactions on Robotics* 38.2, pp. 1063–1079. DOI: [10.1109/TR0.2021.3098436](https://doi.org/10.1109/TR0.2021.3098436).

Shi, Guanya, Wolfgang Hönig, Yisong Yue, and Soon-Jo Chung (2020). *Neural-Swarm: Decentralized close-proximity multirotor control using learned interactions*. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3241–3247. DOI: [10.1109/ICRA40945.2020.9196800](https://doi.org/10.1109/ICRA40945.2020.9196800).

---

**Abstract.** We present *Neural-Swarm*, a learning-based method for motion planning and control that allows heterogeneous multirotors in a swarm to safely fly in close proximity. Such operation for drones is challenging due to complex aerodynamic interaction forces, such as downwash generated by nearby drones and ground effect. Conventional planning and control methods neglect capturing these interaction forces, resulting in sparse swarm configuration during flight. Our approach combines a physics-based nominal dynamics model with learned Deep Neural Networks (DNNs) with strong Lipschitz properties. We make use of two techniques to accurately predict the aerodynamic interactions between heterogeneous multirotors: i) spectral normalization for stability and generalization guarantees of unseen data and ii) heterogeneous deep sets for supporting any number of heterogeneous neighbors in a permutation-invariant manner without reducing expressiveness. The learned residual dynamics benefit both the proposed interaction-aware multi-robot motion planning and the nonlinear tracking control design because the learned interaction forces reduce the modelling errors. Experimental results demonstrate that *Neural-Swarm* is able to generalize to larger swarms beyond training cases and significantly outperforms a baseline nonlinear tracking controller with up to three times reduction in worst-case tracking errors.

## 4.1 Introduction

The ongoing commoditization of unmanned aerial vehicles (UAVs) requires robots to fly in much closer proximity to each other than before, which necessitates ad-

<sup>1</sup>Summary video: <https://youtu.be/Y02juH6BDxo>

<sup>2</sup>Data and code: <https://github.com/aerorobotics/neural-swarm>

vanced planning and control methods for large aerial swarms (Chung et al., 2018; Morgan, Subramanian, et al., 2016). For example, consider a search-and-rescue mission where an aerial swarm must enter and search a collapsed building. In such scenarios, close-proximity flight enables the swarm to navigate the building much faster compared to swarms that must maintain large distances from each other. Other important applications of close-proximity flight include manipulation, search, surveillance, and mapping. In many scenarios, heterogeneous teams with robots of different sizes and sensing or manipulation capabilities are beneficial due to their significantly higher adaptability. For example, in a search-and-rescue mission, larger UAVs can be used for manipulation tasks or to transport goods, while smaller ones are more suited for exploration and navigation.

A major challenge of close-proximity control and planning is that small distances between UAVs create complex aerodynamic interactions. For instance, one multirotor flying above another causes the so-called downwash effect on the lower one, which is difficult to model using conventional model-based approaches (Jain et al., 2019). Without accurate downwash interaction modeling, a large safety distance between vehicles is necessary, thereby preventing a compact 3-D formation shape, e.g., 60 cm for the small Crazyflie 2.0 quadrotor (9 cm rotor-to-rotor) (Hönig et al., 2018). Moreover, the downwash is sometimes avoided by restricting the relative position between robots in the 2-D horizontal plane (Du et al., 2019). For heterogeneous teams, even larger and asymmetric safety distances are required (Debord et al., 2018). However, the downwash for two small Crazyflie quadrotors hovering 30 cm on top of each other is only 9 g, which is well within their thrust capabilities, and suggests that proper modeling of downwash and other interaction effects can lead to more precise motion planning and dense formation control.

In this chapter, we present a learning-based approach, *Neural-Swarm*, which enhances the precision, safety, and density of close-proximity motion planning and control of heterogeneous multirotor swarms. In the example shown in Fig. 4.1, we safely operate the same drones with vertical distances less than half of those of prior work (Hönig et al., 2018). In particular, we train deep neural networks (DNNs) to predict the residual interaction forces that are not captured by the nominal models of free-space aerodynamics. To the best of our knowledge, this is the first model for aerodynamic interactions between two or more multirotors in flight. Our DNN architecture supports heterogeneous inputs in a permutation-invariant manner without reducing the expressiveness. The DNN only requires relative positions and veloci-

ties of neighboring multirotors as inputs, similar to the existing collision-avoidance techniques (Berg et al., 2009), which enables fully-decentralized computation. We use the predicted interaction forces to augment the nominal dynamics and derive novel methods to directly consider them during motion planning and as part of the multirotors' controller.

From a learning perspective, we leverage and extend two state-of-the-art tools to derive effective DNN models. First, we extend deep sets (Zaheer et al., 2017) to the heterogeneous case and prove its representation power. Our novel encoding is used to model interactions between heterogeneous vehicle types in an index-free or permutation-invariant manner, enabling better generalization to new formations and a varying number of vehicles. The second is spectral normalization (see details in Chapter 2 and Algorithm 2.1), which ensures the DNN is Lipschitz continuous and helps the DNN generalize well on test examples that lie outside the training set. We demonstrate that the interaction forces can be computationally efficiently and accurately learned such that a small 32-bit microcontroller can predict such forces in real-time.

From a planning and control perspective, we derive novel methods that directly consider the predicted interaction forces. For motion planning, we use a two-stage approach. In the first stage, we extend an existing kinodynamic sampling-based planner for a single robot to the interaction-aware multi-robot case. In the second stage, we adopt an optimization-based planner to refine the solutions of the first stage. Empirically, we demonstrate that our interaction-aware motion planner both avoids dangerous robot configurations that would saturate the multirotors' motors and reduces the tracking error significantly. For the nonlinear control we leverage the Lipschitz continuity of our learned interaction forces to derive stability guarantees similar to Chapter 3. The controller can be used to reduce the tracking error of arbitrary desired trajectories, including ones that were not planned with an interaction-aware planner.

We validate our approach using two to sixteen quadrotors of two different sizes, and we also integrate ground effect and other unmodeled dynamics into our model, by viewing the physical environment as a special robot. To our knowledge, our approach is the first that models interactions between two or more multirotor vehicles and demonstrates how to use such a model effectively and efficiently for motion planning and control of aerial teams.

## 4.2 Related Work

The aerodynamic interaction force applied to a single UAV flying near the ground (ground effect), has been discussed in Chapter 3. The interaction between two rotor blades of a single multirotor has been studied in a lab setting to optimize the placement of rotors on the vehicle (Shukla and Komerath, 2018). However, it remains an open question how this influences the flight of two or more multirotors in close proximity. Interactions between two multirotors can be estimated using a propeller velocity field model (Jain et al., 2019). Unfortunately, this method is hard to generalize to the multi-robot or heterogeneous case and it only considers the stationary case, which is inaccurate for real flights.

The use of DNNs to learn higher-order residual dynamics or control actions is gaining attention in the areas of control and reinforcement learning settings (Le et al., 2016; Taylor et al., 2019; Cheng et al., 2019; McKinnon and Schoellig, 2019; Saveriano et al., 2017; Johannink et al., 2019). For swarms, a common encoding approach is to discretize the whole space and employ convolutional neural networks (CNNs), which yields a permutation-invariant encoding. Another common encoding for robot swarms is a Graphical Neural Network (GNN) (Scarselli et al., 2008; Tolstaya et al., 2020). GNNs have been extended to heterogeneous graphs (Zhang et al., 2019), but it remains an open research question how such a structure would apply to heterogeneous robot teams. We extend a different architecture, which is less frequently used in robotics applications, called deep sets (Zaheer et al., 2017). Deep sets enable distributed computation without communication requirements. Compared to CNNs, our approach: i) requires less training data and computation; ii) is not restricted to a pre-determined resolution and input domain; and iii) directly supports the heterogeneous swarm. Compared to GNNs, we do not require any direct communication between robots. Deep sets have been used in robotics for heterogeneous (Rivi re et al., 2020) teams. Our heterogeneous deep set extension has a more compact encoding and we prove its representation power.

For motion planning, empirical models have been used to avoid harmful interactions (Morgan, Subramanian, et al., 2016; Morgan, Chung, et al., 2014; H nig et al., 2018; Debord et al., 2018; Mellinger et al., 2012). Typical safe boundaries along multi-vehicle motions form ellipsoids (H nig et al., 2018) or cylinders (Debord et al., 2018) along the motion trajectories. Estimating such shapes experimentally would potentially lead to many collisions and dangerous flight tests and those collision-free regions are in general conservative. In contrast, we use deep learning to estimate

the interaction forces accurately in heterogeneous multi-robot teams. This model allows us to directly control the magnitude of the interaction forces to accurately and explicitly control the risk, removing the necessity of conservative collision shapes.

### 4.3 Problem Statement

Similar to *Neural-Lander* in Chapter 3, *Neural-Swarm* can generally apply to any robotic system and we will focus on multirotor UAVs in this chapter for notational simplicity. In this section, we first recap single multirotor dynamics given in Eq. (2.4) with an extra first-order motor delay model. Then, we generalize these dynamics for a swarm of multirotors. Finally, we formulate our objective as a variant of an optimal control problem and introduce our performance metric.

#### Single Multirotor Dynamics

We will keep using the multirotor model given in Eq. (2.4) but with extra delay modelling:

$$\dot{p} = v, \quad m\dot{v} = mg + Rf_u + f_a, \quad (4.1a)$$

$$\dot{R} = RS(\omega), \quad J\dot{\omega} = J\omega \times \omega + \tau + \tau_a, \quad (4.1b)$$

$$\underbrace{[T; \tau_x; \tau_y; \tau_z]}_{\eta_o} = B_0 u, \quad \dot{u} = -\lambda u + \lambda u_c \quad (4.1c)$$

where  $\eta_o$  is the output wrench,  $u$  is the control input to the system (i.e., squared motor speeds),  $u_c$  is the actual command signal we can control and  $\lambda > 0$  is the time constant of the delay model.

Compared to Eq. (2.4) in Chapter 2, in this chapter we consider a first-order delay model in Eq. (4.1c). Moreover, the aerodynamic residual terms  $f_a$  and  $\tau_a$  are mainly from the interaction between other multirotors and the environment.

#### Heterogeneous Swarm Dynamics

We now consider  $N$  multirotor robots. We use  $x^{(i)} = [p^{(i)}; v^{(i)}; R^{(i)}; \omega^{(i)}]$  to denote the state of the  $i^{\text{th}}$  multirotor. We use  $x^{(ij)}$  to denote the *relative* state component between robot  $i$  and  $j$ , e.g.,  $x^{(ij)} = [p^{(j)} - p^{(i)}; v^{(j)} - v^{(i)}; R^{(i)}R^{(j)\top}]$ .

We use  $\mathcal{I}(i)$  to denote the type of the  $i^{\text{th}}$  robot, where robots with identical physical parameters such as  $m$ ,  $J$ , and  $B_0$  are considered to be of the same type. We assume there are  $K \leq N$  types of robots, i.e.,  $\mathcal{I}(\cdot)$  is a surjective mapping from  $\{1, \dots, N\}$  to  $\{\text{type}_1, \dots, \text{type}_K\}$ . Let  $\mathbf{r}_{\text{type}_k}^{(i)}$  be the set of the relative states of the  $\text{type}_k$  neighbors

of robot  $i$ :

$$\mathbf{r}_{\text{type}_k}^{(i)} = \{x^{(ij)} \mid j \in \text{neighbor}(i) \text{ and } \mathcal{I}(j) = \text{type}_k\}. \quad (4.2)$$

The neighboring function  $\text{neighbor}(i)$  is defined by an interaction volume function  $\mathcal{V}$ . Formally,  $j \in \text{neighbor}(i)$  if  $p^{(j)} \in \mathcal{V}(p^{(i)}, \mathcal{I}(i), \mathcal{I}(j))$ , i.e., robot  $j$  is a neighbor of  $i$  if the position of  $j$  is within the interaction volume of  $i$ . In this chapter, we design  $\mathcal{V}$  as a cuboid based on observed interactions in experiments. The ordered sequence of all relative states grouped by robot type is

$$\mathbf{r}_{\mathcal{I}}^{(i)} = \left( \mathbf{r}_{\text{type}_1}^{(i)}, \mathbf{r}_{\text{type}_2}^{(i)}, \dots, \mathbf{r}_{\text{type}_K}^{(i)} \right). \quad (4.3)$$

The dynamics of the  $i^{\text{th}}$  multirotor can be written in compact form:

$$\dot{x}^{(i)} = \Phi^{(i)}(x^{(i)}, u^{(i)}) + \begin{bmatrix} 0 \\ f_a^{(i)}(\mathbf{r}_{\mathcal{I}}^{(i)}) \\ 0 \\ \tau_a^{(i)}(\mathbf{r}_{\mathcal{I}}^{(i)}) \end{bmatrix}, \quad (4.4)$$

where  $\Phi^{(i)}(x^{(i)}, u^{(i)})$  denotes the nominal dynamics of robot  $i$ , and  $f_a^{(i)}(\cdot)$  and  $\tau_a^{(i)}(\cdot)$  are the unmodeled force and torque of the  $i^{\text{th}}$  robot that are caused by interactions with neighboring robots or the environment (e.g., ground effect and air drag).

Robots with the same type have the same nominal dynamics and unmodeled force and torque:

$$\Phi^{(i)}(\cdot) = \Phi^{\mathcal{I}(i)}(\cdot), f_a^{(i)}(\cdot) = f_a^{\mathcal{I}(i)}(\cdot), \tau_a^{(i)}(\cdot) = \tau_a^{\mathcal{I}(i)}(\cdot) \forall i. \quad (4.5)$$

Note that the homogeneous case is a special case where  $K = 1$ , i.e.,  $\Phi^{(i)}(\cdot) = \Phi(\cdot)$ ,  $f_a^{(i)}(\cdot) = f_a(\cdot)$ , and  $\tau_a^{(i)}(\cdot) = \tau_a(\cdot) \forall i$ .

Our system is heterogeneous in three ways: i) different robot types have heterogeneous nominal dynamics  $\Phi^{\mathcal{I}(i)}$ ; ii) different robot types have different unmodeled  $f_a^{\mathcal{I}(i)}$  and  $\tau_a^{\mathcal{I}(i)}$ ; and iii) the neighbors of each robot belong to  $K$  different sets.

We highlight that our heterogeneous model not only captures different types of robot, but also different types of environmental interactions, e.g., ground effect (G. Shi, X. Shi, et al., 2019) and air drag. This is achieved in a straightforward manner by viewing the physical environment as a special robot type. We illustrate this generalization in the following example.

**Example 4.1** (small and large robots, and the environment). *We consider a heterogeneous system as depicted in Fig. 4.1(a). Robot 3 (large robot) has three neighbors: robot 1 (small), robot 2 (small) and environment 4. For robot 3, we have*

$$\begin{aligned} f_a^{(3)} &= f_a^{\text{large}}(\mathbf{r}_{\mathcal{I}}^{(3)}) = f_a^{\text{large}}(\mathbf{r}_{\text{small}}^{(3)}, \mathbf{r}_{\text{large}}^{(3)}, \mathbf{r}_{\text{env}}^{(3)}), \\ \mathbf{r}_{\text{small}}^{(3)} &= \{x^{(31)}, x^{(32)}\}, \quad \mathbf{r}_{\text{large}}^{(3)} = \emptyset, \quad \mathbf{r}_{\text{env}}^{(3)} = \{x^{(34)}\} \end{aligned}$$

and a similar expression for  $\tau_a^{(3)}$ .

### Interaction-Aware Motion Planning & Control

Our goal is to move the heterogeneous team of robots from their start states to goal states, which can be framed as the following optimal control problem:

$$\begin{aligned} \min_{u^{(i)}, x^{(i)}, t_f} \sum_{i=1}^N \int_0^{t_f} \|u^{(i)}(t)\| dt & \quad (4.6) \\ \text{s.t.} \begin{cases} \text{robot dynamics (4.4)} & i \in [1, N] \\ u^{(i)}(t) \in \mathcal{U}^{\mathcal{I}(i)}; x^{(i)}(t) \in \mathcal{X}^{\mathcal{I}(i)} & i \in [1, N] \\ \|p^{(ij)}\| \geq r^{(\mathcal{I}(i)\mathcal{I}(j))} & i < j, j \in [2, N] \\ \|f_a^{(i)}\| \leq f_{a,\max}^{\mathcal{I}(i)}; \|\tau_a^{(i)}\| \leq \tau_{a,\max}^{\mathcal{I}(i)} & i \in [1, N] \\ x^{(i)}(0) = x_s^{(i)}; x^{(i)}(t_f) = x_f^{(i)} & i \in [1, N] \end{cases} \end{aligned}$$

where  $\mathcal{U}^{(k)}$  is the control space for type<sub>k</sub> robots,  $\mathcal{X}^{(k)}$  is the free space for type<sub>k</sub> robots,  $r^{(lk)}$  is the minimum safety distance between type<sub>l</sub> and type<sub>k</sub> robots,  $f_{a,\max}^{(k)}$  is the maximum endurable interaction force for type<sub>k</sub> robots,  $\tau_{a,\max}^{(k)}$  is the maximum endurable interaction torque for type<sub>k</sub> robots,  $x_s^{(i)}$  is the start state of robot  $i$ , and  $x_f^{(i)}$  is the desired state of robot  $i$ . In contrast with the existing literature (Debord et al., 2018), we assume a tight spherical collision model and bound the interaction forces directly, eliminating the need of manually defining virtual collision shapes. For instance, larger  $f_{a,\max}^{\mathcal{I}(i)}$  and  $\tau_{a,\max}^{\mathcal{I}(i)}$  will yield denser and more aggressive trajectories. Also note that the time horizon  $t_f$  is a decision variable.

Solving Eq. (4.6) in real-time in a distributed fashion is intractable due to the exponential growth of the decision space with respect to the number of robots. Thus, we focus on solving two common subproblems instead. First, we approximately solve Eq. (4.6) offline as an interaction-aware motion planning problem. Second, we formulate an interaction-aware controller that minimizes the tracking error online. This controller can use both predefined trajectories and planned trajectories from the interaction-aware motion planner.

Since interaction between robots might only occur for a short time period with respect to the overall flight duration but can cause significant deviation from the nominal trajectory, we consider the worst tracking error of any robot in the team as a success metric:

$$\max_{i,t} \|p^{(i)}(t) - p_d^{(i)}(t)\|, \quad (4.7)$$

where  $p_d^{(i)}(t)$  is the desired trajectory for robot  $i$ . Note that this metric reflects the worst error out of *all* robots, because different robots in a team have various levels of task difficulty. For example, the two-drone swapping task is very challenging for the bottom drone due to the downwash effect, but relatively easier for the top drone. Minimizing Eq. (4.7) implies improved tracking performance and safety of a multirotor swarm during tight formation flight.

#### 4.4 Learning of Swarm Aerodynamic Interaction

We employ state-of-the-art deep learning methods to capture the unknown (or residual) dynamics caused by interactions of heterogeneous robot teams. In order to use the learned functions effectively for motion planning and control, we require that the DNNs have strong Lipschitz properties (for stability analysis), can generalize well to new test cases, and use compact encodings to achieve high computational and statistical efficiency. To that end, we introduce *heterogeneous deep sets*, a generalization of regular deep sets (Zaheer et al., 2017), and employ spectral normalization for strong Lipschitz properties (see more details in Chapter 2).

In this section, we will first review the homogeneous learning architecture covered in prior work (Zaheer et al., 2017). Then we generalize them to the heterogeneous case with representation guarantees. Finally, we introduce our data collection procedures.

##### Homogeneous Permutation-Invariant Neural Networks

Recall that in the homogeneous case, all robots are with the same type ( $\text{type}_1$ ). Therefore, the input to functions  $f_a$  or  $\tau_a$  is a single set. The permutation-invariant aspect of  $f_a$  or  $\tau_a$  can be characterized as:

$$f_a(\mathbf{r}_{\text{type}_1}^{(i)}) = f_a(\pi(\mathbf{r}_{\text{type}_1}^{(i)})), \quad \tau_a(\mathbf{r}_{\text{type}_1}^{(i)}) = \tau_a(\pi(\mathbf{r}_{\text{type}_1}^{(i)}))$$

for any permutation  $\pi$ . Since the aim is to learn the function  $f_a$  and  $\tau_a$  using DNNs, we need to guarantee that the learned DNN is permutation-invariant. Therefore, we consider the following “deep sets” (Zaheer et al., 2017) architecture to approximate homogeneous  $f_a$  and  $\tau_a$ :



$$\begin{bmatrix} f_a(\mathbf{r}_{\text{type}_1}^{(i)}) \\ \tau_a(\mathbf{r}_{\text{type}_1}^{(i)}) \end{bmatrix} \approx \boldsymbol{\rho} \left( \sum_{x^{(ij)} \in \mathbf{r}_{\text{type}_1}^{(i)}} \phi(x^{(ij)}) \right) := \begin{bmatrix} \hat{f}_a^{(i)} \\ \hat{\tau}_a^{(i)} \end{bmatrix}, \quad (4.8)$$

where  $\phi(\cdot)$  and  $\boldsymbol{\rho}(\cdot)$  are two DNNs. The output of  $\phi$  is a hidden state to represent ‘‘contributions’’ from each neighbor, and  $\boldsymbol{\rho}$  is a nonlinear mapping from the summation of these hidden states to the total effect.

Obviously the network architecture in Eq. (4.8) is permutation-invariant due to the inner sum operation. We now show that this architecture is able to approximate any continuous permutation-invariant function. The following Lemma 4.1 and Theorem 4.1 are adopted from (Zaheer et al., 2017) and will be used and extended in the next section for the heterogeneous case.

**Lemma 4.1.** *Define  $\bar{\phi}(z) = [1; z; \dots; z^M] \in \mathbb{R}^{M+1}$  as a mapping from  $\mathbb{R}$  to  $\mathbb{R}^{M+1}$ , and  $\mathcal{X} = \{[x_1; \dots; x_M] \in [0, 1]^M | x_1 \leq \dots \leq x_M\}$  as a subset of  $[0, 1]^M$ . For  $\mathbf{x} = [x_1; \dots; x_M] \in \mathcal{X}$ , define  $\mathbf{q}(\mathbf{x}) = \sum_{m=1}^M \bar{\phi}(x_m)$ . Then  $\mathbf{q}(\mathbf{x}) : \mathcal{X} \rightarrow \mathbb{R}^{M+1}$  is a homeomorphism.*

*Proof.* The proof builds on the Newton-Girard formulae, which connect the moments of a sample set (sum-of-power) to the elementary symmetric polynomials (see Zaheer et al. (2017)).  $\square$

**Theorem 4.1.** *Suppose  $h(\mathbf{x}) : [0, 1]^M \rightarrow \mathbb{R}$  is a permutation-invariant continuous function, i.e.,  $h(\mathbf{x}) = h(x_1, \dots, x_M) = h(\pi(x_1, \dots, x_M))$  for any permutation  $\pi$ . Then there exist continuous functions  $\bar{\rho} : \mathbb{R}^{M+1} \rightarrow \mathbb{R}$  and  $\bar{\phi} : \mathbb{R} \rightarrow \mathbb{R}^{M+1}$  such that*

$$h(\mathbf{x}) = \bar{\rho} \left( \sum_{m=1}^M \bar{\phi}(x_m) \right), \quad \forall \mathbf{x} \in [0, 1]^M.$$

*Proof.* We choose  $\bar{\phi}(z) = [1; z; \dots; z^M]$  and  $\bar{\rho}(\cdot) = h(\mathbf{q}^{-1}(\cdot))$ , where  $\mathbf{q}(\cdot)$  is defined in Lemma 4.1. Note that since  $\mathbf{q}(\cdot)$  is a homeomorphism,  $\mathbf{q}^{-1}(\cdot)$  exists and it is a continuous function from  $\mathbb{R}^{M+1}$  to  $\mathcal{X}$ . Therefore,  $\bar{\rho}$  is also a continuous function from  $\mathbb{R}^{M+1}$  to  $\mathbb{R}$ , and  $\bar{\rho} \left( \sum_{m=1}^M \bar{\phi}(x_m) \right) = \bar{\rho}(\mathbf{q}(\mathbf{x})) = h(\mathbf{q}^{-1}(\mathbf{q}(\mathbf{x}))) = h(\mathbf{x})$  for  $\mathbf{x} \in \mathcal{X}$ . Finally, note that for any  $\mathbf{x} \in [0, 1]^M$ , there exists some permutation  $\pi$  such that  $\pi(\mathbf{x}) \in \mathcal{X}$ . Then because both  $\bar{\rho}(\mathbf{q}(\mathbf{x}))$  and  $h(\mathbf{x})$  are permutation-invariant, we have  $\bar{\rho}(\mathbf{q}(\mathbf{x})) = \bar{\rho}(\mathbf{q}(\pi(\mathbf{x}))) = h(\pi(\mathbf{x})) = h(\mathbf{x})$  for all  $\mathbf{x} \in [0, 1]^M$ .  $\square$

Theorem 4.1 focuses on scalar valued permutation-invariant continuous functions with hypercubic input space  $[0, 1]^M$ , i.e., each element in the input set is a scalar. In contrast, our learning target function  $[f_a; \tau_a]$  in Eq. (4.8) is a vector valued function with a bounded input space, and each element in the input set is also a vector. However, Theorem 4.1 can be generalized in a straightforward manner by the following corollary.

**Corollary 4.1.** *Suppose  $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(M)}$  are  $M$  bounded vectors in  $\mathbb{R}^{D_1}$ , and  $h(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(M)})$  is a continuous permutation-invariant function from  $\mathbb{R}^{M \times D_1}$  to  $\mathbb{R}^{D_2}$ , i.e.,  $h(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(M)}) = h(\mathbf{x}^{\pi(1)}, \dots, \mathbf{x}^{\pi(M)})$  for any permutation  $\pi$ . Then  $h(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(M)})$  can be approximated arbitrarily close in the proposed architecture in Eq. (4.8).*

*Proof.* First, there exists a bijection from the bounded vector space in  $\mathbb{R}^{D_1}$  to  $[0, 1]$  after discretization, with finite but arbitrary precision. Thus, Theorem 4.1 is applicable. Second, we apply Theorem 4.1  $D_2$  times and stack  $D_2$  scalar-valued functions to represent the vector-valued function with output space  $\mathbb{R}^{D_2}$ . Finally, because DNNs are universal approximators for continuous functions (Csáji et al., 2001), the proposed architecture in Eq. (4.8) can approximate any  $h(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(M)})$  arbitrarily close.  $\square$

### Heterogeneous $K$ -Group Permutation-Invariant DNN

Different from the homogeneous setting, the inputs to functions  $f_a^{I(i)}$  and  $\tau_a^{I(i)}$  in Eq. (4.5) are  $K$  different sets. First, we define *permutation-invariance* in the heterogeneous case. Intuitively, we expect that the following equality holds:

$$f_a^{I(i)}(\mathbf{r}_{\text{type}_1}^{(i)}, \dots, \mathbf{r}_{\text{type}_K}^{(i)}) = f_a^{I(i)}(\pi_1(\mathbf{r}_{\text{type}_1}^{(i)}), \dots, \pi_K(\mathbf{r}_{\text{type}_K}^{(i)}))$$

for any permutations  $\pi_1, \dots, \pi_K$  (similarly for  $\tau_a^{I(i)}$ ). Formally, we define  $K$ -group permutation invariance as follows.

**Definition 4.1** ( $K$ -group permutation invariance). *Let  $\mathbf{x}^{(k)} = [x_1^{(k)}; \dots; x_{M_k}^{(k)}] \in [0, 1]^{M_k}$  for  $1 \leq k \leq K$ , and  $\mathbf{x} = [\mathbf{x}^{(1)}; \dots; \mathbf{x}^{(K)}] \in [0, 1]^{M_K}$ , where  $M_K = \sum_{k=1}^K M_k$ .  $h(\mathbf{x}) : \mathbb{R}^{M_K} \rightarrow \mathbb{R}$  is  $K$ -group permutation-invariant if*

$$h([\mathbf{x}^{(1)}; \dots; \mathbf{x}^{(K)}]) = h([\pi_1(\mathbf{x}^{(1)}); \dots; \pi_K(\mathbf{x}^{(K)})])$$

for any permutations  $\pi_1, \pi_2, \dots, \pi_K$ .

For example,  $h(x_1, x_2, y_1, y_2) = \max\{x_1, x_2\} + 2 \cdot \max\{y_1, y_2\}$  is a 2-group permutation-invariant function, because we can swap  $x_1$  and  $x_2$  or swap  $y_1$  and  $y_2$ , but if we interchange  $x_1$  and  $y_1$  the function value may vary. In addition, the  $f_a^{\text{large}}$  function in Example 4.1 is a 3-group permutation-invariant function.

Similar to Lemma 4.1, in order to handle ambiguity due to permutation, we define  $\mathcal{X}_{M_k} = \{[x_1; \dots; x_{M_k}] \in [0, 1]^{M_k} | x_1 \leq \dots \leq x_{M_k}\}$  and

$$\mathcal{X}_K = \{[\mathbf{x}^{(1)}; \dots; \mathbf{x}^{(K)}] \in [0, 1]^{M_K} | \mathbf{x}^{(k)} \in \mathcal{X}_{M_k}, \forall k\}.$$

Finally, we show how a  $K$ -group permutation-invariant function can be approximated via the following theorem.

**Theorem 4.2.**  $h(\mathbf{x}) : [0, 1]^{M_K} \rightarrow \mathbb{R}$  is a  $K$ -group permutation-invariant continuous function if and only if it has the representation

$$h(\mathbf{x}) = \bar{\rho} \left( \sum_{m=1}^{M_1} \bar{\phi}_1(x_m^{(1)}) + \dots + \sum_{m=1}^{M_K} \bar{\phi}_K(x_m^{(K)}) \right) = \bar{\rho} \left( \sum_{k=1}^K \sum_{m=1}^{M_k} \bar{\phi}_k(x_m^{(k)}) \right), \forall \mathbf{x} \in [0, 1]^{M_K}$$

for some continuous outer and inner functions  $\bar{\rho} : \mathbb{R}^{K+M_K} \rightarrow \mathbb{R}$  and  $\bar{\phi}_k : \mathbb{R} \rightarrow \mathbb{R}^{K+M_K}$  for  $1 \leq k \leq K$ .

*Proof.* The sufficiency follows from that  $h(\mathbf{x})$  is  $K$ -group permutation-invariant by construction. For the necessary condition, we need to find continuous functions  $\bar{\rho}$  and  $\{\bar{\phi}_k\}_{k=1}^K$  given  $h$ . We define  $\bar{\phi}_k(x) : \mathbb{R} \rightarrow \mathbb{R}^{K+M_K}$  as

$$\bar{\phi}_k(x) = [\mathbf{0}_{M_1}; \dots; \mathbf{0}_{M_{k-1}}; \begin{bmatrix} 1 \\ x \\ \vdots \\ x^{M_k} \end{bmatrix}; \mathbf{0}_{M_{k+1}}; \dots; \mathbf{0}_{M_K}]$$

where  $\mathbf{0}_{M_k} = [0; \dots; 0] \in \mathbb{R}^{M_k+1}$ . Then

$$\mathbf{q}_K(\mathbf{x}) = \sum_{k=1}^K \sum_{m=1}^{M_k} \bar{\phi}_k(x_m^{(k)})$$

is a homeomorphism from  $\mathcal{X}_K \subseteq \mathbb{R}^{M_K}$  to  $\mathbb{R}^{K+M_K}$  from Lemma 4.1. We choose  $\bar{\rho} : \mathbb{R}^{K+M_K} \rightarrow \mathbb{R}$  as  $\bar{\rho}(\cdot) = h(\mathbf{q}_K^{-1}(\cdot))$  which is continuous, because both  $\mathbf{q}_K^{-1}$  and  $h$  are continuous. Then  $\bar{\rho}(\mathbf{q}_K(\mathbf{x})) = h(\mathbf{x})$  for  $\mathbf{x} \in \mathcal{X}_K$ . Finally, because i) for all  $\mathbf{x} = [\mathbf{x}^{(1)}; \dots; \mathbf{x}^{(K)}]$  in  $[0, 1]^{M_K}$  there exist permutations  $\pi_1, \dots, \pi_K$  such that  $[\pi_1(\mathbf{x}^{(1)}); \dots; \pi_K(\mathbf{x}^{(K)})] \in \mathcal{X}_K$ ; and ii) both  $\bar{\rho}(\mathbf{q}_K(\mathbf{x}))$  and  $h(\mathbf{x})$  are  $K$ -group permutation-invariant, we have  $\bar{\rho}(\mathbf{q}_K(\mathbf{x})) = h(\mathbf{x})$  for  $\mathbf{x} \in [0, 1]^{M_K}$ .  $\square$

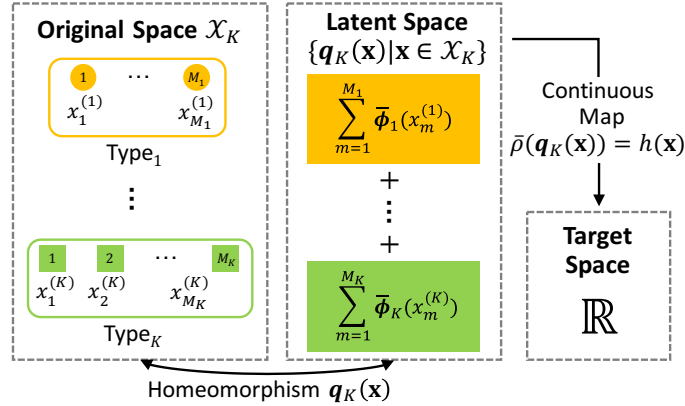


Figure 4.2: Illustration of Theorem 4.2. We first find a homeomorphism  $\mathbf{q}_K(\cdot)$  between the original space and the latent space, and then find a continuous function  $\bar{\rho}(\cdot)$  such that  $\bar{\rho}(\mathbf{q}_K(\cdot)) = h(\cdot)$ .

Figure 4.2 depicts the key idea of Theorem 4.2. Moreover, we provide a 2-group permutation-invariant function example to highlight the roles of  $\phi$  and  $\rho$  in the heterogeneous case.

**Example 4.2** (2-group permutation-invariant function). Consider  $h(x_1, x_2, y_1, y_2) = \max\{x_1, x_2\} + 2 \cdot \max\{y_1, y_2\}$ , which is 2-group permutation-invariant. Then we define  $\phi_x(x) = [e^{\alpha x}; x e^{\alpha x}; 0; 0]$ ,  $\phi_y(y) = [0; 0; e^{\alpha y}; y e^{\alpha y}]$  and  $\rho([a; b; c; d]) = b/a + 2 \cdot d/c$ . Note that

$$\rho(\phi_x(x_1) + \phi_x(x_2) + \phi_y(y_1) + \phi_y(y_2)) = \frac{x_1 e^{\alpha x_1} + x_2 e^{\alpha x_2}}{e^{\alpha x_1} + e^{\alpha x_2}} + 2 \cdot \frac{y_1 e^{\alpha y_1} + y_2 e^{\alpha y_2}}{e^{\alpha y_1} + e^{\alpha y_2}},$$

which is asymptotically equal to  $\max\{x_1, x_2\} + 2 \cdot \max\{y_1, y_2\}$  as  $\alpha \rightarrow +\infty$ .

Similar to the homogeneous case, Theorem 4.2 can generalize to vector-output functions with a bounded input space by applying the same argument as in Corollary 4.1. We propose the following *heterogeneous deep set* structure to model the heterogeneous functions  $f_a^{I(i)}$  and  $\tau_a^{I(i)}$ :

$$\begin{bmatrix} f_a^{I(i)}(\mathbf{r}_{\text{type}_1}^{(i)}, \dots, \mathbf{r}_{\text{type}_K}^{(i)}) \\ \tau_a^{I(i)}(\mathbf{r}_{\text{type}_1}^{(i)}, \dots, \mathbf{r}_{\text{type}_K}^{(i)}) \end{bmatrix} \approx \rho_{I(i)} \left( \sum_{k=1}^K \sum_{x^{(ij)} \in \mathbf{r}_{\text{type}_k}^{(i)}} \phi_{I(j)}(x^{(ij)}) \right) := \begin{bmatrix} \hat{f}_a^{(i)} \\ \hat{\tau}_a^{(i)} \end{bmatrix}. \quad (4.9)$$

**Example 4.3** (Use of 3-group permutation-invariant function for multirotors). For example, in the heterogeneous system provided by Example 4.1 (as depicted in Fig. 4.1(a)), we have

$$\begin{bmatrix} f_a^{(3)} \\ \tau_a^{(3)} \end{bmatrix} = \begin{bmatrix} f_a^{\text{large}}(\mathbf{r}_{\text{small}}^{(3)}, \mathbf{r}_{\text{large}}^{(3)}, \mathbf{r}_{\text{env}}^{(3)}) \\ \tau_a^{\text{large}}(\mathbf{r}_{\text{small}}^{(3)}, \mathbf{r}_{\text{large}}^{(3)}, \mathbf{r}_{\text{env}}^{(3)}) \end{bmatrix} \approx \rho_{\text{large}} \left( \phi_{\text{small}}(x^{(31)}) + \phi_{\text{small}}(x^{(32)}) + \phi_{\text{env}}(x^{(34)}) \right),$$

for the large robot 3, where  $\phi_{\text{small}}$  captures the interaction with the small robot 1 and 2, and  $\phi_{\text{env}}$  captures the interaction with the environment 4, e.g., ground effect and air drag.

The structure in Eq. (4.9) has many valuable properties:

- **Representation ability.** Since Theorem 4.2 is necessary and sufficient, we do not lose approximation power by using this constrained framework, i.e., any  $K$ -group permutation-invariant function can be learned by Eq. (4.9). We demonstrate strong empirical performance using relatively compact DNNs for  $\rho_{I(i)}$  and  $\phi_{I(j)}$ .
- **Computational and sampling efficiency and scalability.** Since the input dimension of  $\phi_{I(j)}$  is always the same as the single vehicle case, the feed-forward computational complexity of Eq. (4.9) grows linearly with the number of neighboring vehicles. Moreover, the number of neural networks ( $\rho_{I(i)}$  and  $\phi_{I(j)}$ ) we need is  $2K$ , which grows linearly with the number of robot types. In practice, we found that one hour flight data is sufficient to accurately learn interactions between two to five multirotors.
- **Generalization to varying swarm size.** Given learned  $\phi_{I(j)}$  and  $\rho_{I(i)}$  functions, Eq. (4.9) can be used to predict interactions for any swarm size. In other words, we can accurately model swarm sizes (slightly) larger than those used for training. In practice, we found that our model can give good predictions for five multirotor swarms, despite only being trained on one to three multirotor swarms. Theoretical analysis on this generalizability is an interesting future research direction.

### Curriculum Learning

Training DNNs in Eq. (4.9) to approximate  $f_a^{I(i)}$  and  $\tau_a^{I(i)}$  requires collecting close formation flight data. However, the downwash effect causes the nominally controlled multirotors (without compensation for the interaction forces) to move apart from each other. Thus, we use a curriculum/cumulative learning approach: first, we collect data for two multirotors without a DNN and learn a model. Second, we repeat the data collection using our learned model as a feed-forward term in our controller, which allows closer-proximity flight of the two vehicles. Third, we repeat the procedure with increasing number of vehicles, using the current best model.

Note that our data collection and learning are independent of the controller used and independent of the  $f_a^{I(i)}$  or  $\tau_a^{I(i)}$  compensation. In particular, if we actively compensate for the learned  $f_a^{I(i)}$  or  $\tau_a^{I(i)}$ , this will only affect  $\eta_o$  in Eq. (4.1) and not the observed  $f_a^{I(i)}$  or  $\tau_a^{I(i)}$ .

#### 4.5 Interaction-Aware Multi-Robot Planning

We approximately solve Eq. (4.6) offline by using two simplifications: i) we plan sequentially for each robot, treating other robots as dynamic obstacles with known trajectories, and ii) we use double-integrator dynamics plus learned interactions. Both simplifications are common for multi-robot motion planning with applications to multirotors (Morgan, Subramanian, et al., 2016; Luis and Schoellig, 2019). Such a motion planning approach can be easily distributed and is complete for planning instances that fulfill the *well-formed infrastructure* property (Čáp et al., 2015). However, the interaction forces Eq. (4.9) complicate motion planning significantly, because the interactions are highly nonlinear and robot dynamics are not independent from each other anymore.

For example, consider a three-robot team with two small and one large robot as in Fig. 4.1(a). Assume that we already have valid trajectories for the two small robots and now plan a motion for the large robot. The resulting trajectory might result in a significant downwash force for the small robots if the large robot flies directly above the small ones. This strong interaction might invalidate the previous trajectories of the small robots or even violate their interaction force limits  $f_{a,\max}^{\text{small}}$  and  $\tau_{a,\max}^{\text{small}}$ . Furthermore, the interaction force is asymmetric and thus it is not sufficient to only consider the interaction force placed on the large robot. We solve this challenge by directly limiting the change of the interaction forces placed on all neighbors when we plan for a robot. This concept is similar to trust regions in sequential optimization (Foust et al., 2020).

The simplified state is  $x^{(i)} = [p^{(i)}; v^{(i)}; \hat{f}_a^{(i)}]$  and the simplified dynamics Eq. (4.4) become:

$$\dot{x}^{(i)} = f^{(i)}(x^{(i)}, u^{(i)}) = \begin{bmatrix} v^{(i)} \\ u^{(i)} + \hat{f}_a^{(i)} \\ \hat{\dot{f}}_a^{(i)} \end{bmatrix}. \quad (4.10)$$

These dynamics are still complex and nonlinear because of  $\hat{f}_a^{(i)}$ , which is the learned interaction force represented by DNNs in Eq. (4.9). We include  $\hat{f}_a^{(i)}$  in our state space to simplify the enforcement of the bound on the interaction force in Eq. (4.6).

We propose a novel hybrid two-stage planning algorithm, see Algorithm 4.1, leveraging the existing approaches while still highlighting the importance of considering interactive forces/torques in the planning. The portions of the pseudo-code in Algorithm 4.1 that significantly differ from the existing methods to our approach are highlighted. In Stage 1, we find initial feasible trajectories using a kinodynamic sampling-based motion planner. Note that any kinodynamic planner can be used for Stage 1. In Stage 2, we use sequential convex programming (SCP) (Morgan, Subramanian, et al., 2016; Foust et al., 2020; Dinh and Diehl, 2010) to refine the initial solution to reach the desired states exactly and to minimize our energy objective defined in Eq. (4.6). Intuitively, Stage 1 identifies the homeomorphism class of the solution trajectories and fixes  $t_f$ , while Stage 2 finds the optimal trajectories to the goal within that homeomorphism class. Both stages differ from similar methods in the literature (Morgan, Subramanian, et al., 2016), because they need to reason over the coupling of the robots caused by interaction forces  $\hat{f}_a^{(i)}$ .

### Stage 1: Sampling-Based Planning using Interaction Forces

For Stage 1, any kinodynamic single-robot motion planner can be extended. For the coupled multi-robot setting in this chapter, we modify AO-RRT (Hauser and Zhou, 2016), which is a meta-algorithm that uses the rapidly-exploring random tree (RRT) algorithm as a subroutine.

**Sampling-based planner.** Our adaption of RRT (Lines 3 to 15 in Algorithm 4.1) works as follows. First, a random state  $x_{\text{rand}}$  is uniformly sampled from the state space (Line 6) and the closest state  $x_{\text{near}}$  that is already in the search tree  $\mathcal{T}$  is found (Line 7). This search can be done efficiently in logarithmic time by employing a specialized data structure such as a kd-tree (Bentley, 1975) and requires the user to define a distance function on the state space. Second, an action is uniformly sampled from the action space (Line 8) and the dynamics Eq. (4.4) are forward propagated for a fixed time period  $\Delta t$  using  $x_{\text{near}}$  as the initial condition, e.g., by using the Runge-Kutta method (Line 9). Note that this forward propagation directly considers the learned dynamics  $\hat{f}_a^{(i)}$ . Third, the new state  $x_{\text{new}}$  is checked for validity with respect to i) the state space (which includes  $\hat{f}_a^{(i)}$ ), ii) collisions with other robots, and iii) change and bound of the neighbor’s interaction forces (Line 10). The first validity check ensures that the interaction force of the robot itself is bounded, while the third check is a trust region and upper bound for the neighbor’s interaction forces. If  $x_{\text{new}}$  is valid, it is added as a child node of  $x_{\text{near}}$  in the search tree  $\mathcal{T}$  (Line 11).

**Algorithm 4.1:** Interaction-aware motion planning

---

**Input:**  $x_0^{(i)}, x_f^{(i)}, \Delta t$

**Result:**  $\mathcal{X}_{\text{sol}}^{(i)} = (x_0^{(i)}, x_1^{(i)}, x_2^{(i)}, \dots, x_{T^{(i)}}^{(i)})$ ,  $\mathcal{U}_{\text{sol}}^{(i)} = (u_0^{(i)}, u_1^{(i)}, u_2^{(i)}, \dots, u_{T^{(i)}-1}^{(i)})$

▷ Stage 1: Find  $t_f$  and initial trajectories that are close to the goal state

- 1  $c^{(i)} \leftarrow \infty, \mathcal{X}_{\text{sol}}^{(i)} \leftarrow (), \mathcal{U}_{\text{sol}}^{(i)} \leftarrow () \forall i \in \{1, \dots, N\}$
- 2 **repeat**
- 3     **foreach**  $i \in \text{RandomShuffle}(\{1, \dots, N\})$  **do**
- 4          $\mathcal{T} = (\{x_0^{(i)}\}, \emptyset)$
- 5         **repeat**
- 6              $x_{\text{rand}} \leftarrow \text{UniformSample}(\mathcal{X}^{I(i)})$
- 7              $x_{\text{near}} \leftarrow \text{FindClosest}(\mathcal{T}, x_{\text{rand}})$
- 8              $u_{\text{rand}} \leftarrow \text{UniformSample}(\mathcal{U}^{I(i)})$
- 9              $x_{\text{new}}, c \leftarrow \text{Propagate}(x_{\text{near}}, u_{\text{rand}}, \Delta t, \{\mathcal{X}_{\text{sol}}^{(j)} \mid j \neq i\})$
- 10            **if**  $\text{StateValid}(x_{\text{new}}, \{\mathcal{X}_{\text{sol}}^{(j)} \mid j \neq i\})$  **and**  $c \leq c^{(i)}$  **then**
- 11                 **Add** $(\mathcal{T}, x_{\text{near}} \rightarrow x_{\text{new}})$
- 12                 **if**  $\|x_{\text{new}} - x_f^{(i)}\| \leq \varepsilon$  **then**
- 13                      $c^{(i)} \leftarrow c$
- 14                      $\mathcal{X}_{\text{sol}}^{(i)}, \mathcal{U}_{\text{sol}}^{(i)} \leftarrow \text{ExtractSolution}(\mathcal{T}, x_{\text{new}})$
- 15                     **break**
- 16         **until**  $\text{TerminationCondition1}()$
- 17      $\mathcal{X}_{\text{sol}}^{(i)}, \mathcal{U}_{\text{sol}}^{(i)} \leftarrow \text{PostProcess}(\mathcal{X}_{\text{sol}}^{(i)}, \mathcal{U}_{\text{sol}}^{(i)})$
- 18     ▷ Stage 2: Refine trajectories sequentially; Based on SCP
- 19     **repeat**
- 20         **foreach**  $i \in \text{RandomShuffle}(\{1, \dots, N\})$  **do**
- 21              $\mathcal{X}_{\text{sol}}^{(i)}, \mathcal{U}_{\text{sol}}^{(i)} \leftarrow \text{SolveCP}(\text{Eq. (4.14)}, \{\mathcal{X}_{\text{sol}}^{(i)} \mid \forall i\}, \{\mathcal{U}_{\text{sol}}^{(i)} \mid \forall i\})$
- 22     **until**  $\text{Converged}()$

---

Finally, if  $x_{\text{new}}$  is within an  $\varepsilon$ -distance to the goal  $x_f^{(i)}$ , the solution can be extracted by following the parent pointers of each tree node starting from  $x_{\text{new}}$  until the root node  $x_0^{(i)}$  is reached (Line 15).

We note that our RRT steering method departs from ones in the literature which either sample  $\Delta t$ , use a best-control approximation of the steer method in RRT, or use a combination of both  $\Delta t$ -sampling and best-control approximation (Hauser and Zhou, 2016). We are interested in a constant  $\Delta t$  for our optimization formulation in Stage 2. In this case, a best-control approximation would lead to a probabilistic incomplete planner (Kunz and Stilman, 2015). We adopt a technique of goal biasing where we pick the goal state rather than  $x_{\text{rand}}$  in fixed intervals, in order to improve



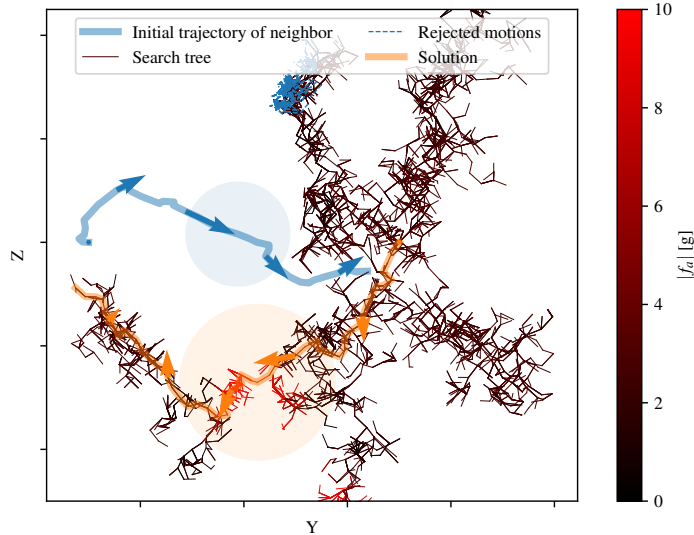


Figure 4.3: Example for Stage 1 of our motion planning with learned dynamics. Here, we have an initial solution for a small (blue) robot and plan for a large (orange) robot. The created search tree of the large robot is color-coded by the magnitude of the interaction force on the orange robot. During the search, we reject states that would cause a significant change in the interaction force for the blue robot (edges in blue).

the convergence speed.

While RRT is probabilistically complete, it also almost surely converges to a suboptimal solution (Karaman and Frazzoli, 2011). AO-RRT remedies this shortcoming by planning in a state-cost space and using RRTs sequentially with a monotonically decreasing cost bound. The cost bound  $c^{(i)}$  is initially set to infinity (Line 1) and the tree can only grow with states that have a lower cost associated with them (Line 10). Once a solution is found, the cost bound is decreased accordingly (Line 13) and the search is repeated using the new cost bound (Line 2). This approach is asymptotically optimal, but in practice the algorithm is terminated based on some condition, e.g., a timeout or a fixed number of iterations without improvements (Line 16).

**Modification of sampling-based planner.** We extend AO-RRT to a sequential interaction-aware multi-robot planner by adding  $\hat{f}_a^{(i)}$  and time to our state space and treating the other robots as dynamic obstacles. As cost, we use a discrete approximation of the objective in Eq. (4.6). For each AO-RRT outer-loop iteration with a fixed cost bound, we compute trajectories sequentially using a random permutation of the robots (Line 3). When we check the state for validity (Line 10), we also enforce that the new state is not in collision with the trajectories of the other robots and that their

interaction forces are bounded and within a trust region compared to their previous value, see Fig. 4.3 for visualization. Here, the red edges show motions that cause large ( $\approx 10$  g) but admissible interaction forces on the orange robot, because the blue robot flies directly above it. The blue edges are candidate edges as computed in Line 9 and are not added to the search tree, because their motion would cause a violation of the interaction force trust region of the blue robot (condition in Line 10). Once the search tree contains a path to the goal region, a solution is returned (orange path).

The output of the sequential planner (Line 15) is a sequence of states  $\mathcal{X}_{\text{sol}}^{(i)}$  and actions  $\mathcal{U}_{\text{sol}}^{(i)}$ , each to be applied for a duration of  $\Delta t$ . Note that the sequences might have different lengths for each robot. Implicitly, the sequences also define  $t_f$ . Furthermore, the first element of each sequence is the robots' start state and the last element is within a  $\varepsilon$ -distance of the robots' goal state. We postprocess this sequence of states to make it an appropriate input for the optimization, e.g., for uniform length (Line 17). In practice, we found that repeating the last state and adding null actions, or (virtual) tracking of the computed trajectories using a controller are efficient and effective postprocessing techniques.

Other sampling-based methods can be used as foundation of the first stage as well, with similar changes in sequential planning, state-augmentation to include the interaction forces, state-validity checking, and postprocessing.

## Stage 2: Optimization-Based Motion Planning

We employ sequential convex programming (SCP) for optimization. SCP is a local optimization method for nonconvex problems that leverages convex optimization. The key concept is to convexify the nonconvex portions of the optimization problem by linearizing around a prior solution. The resulting convex problem instance is solved and a new solution obtained. The procedure can be repeated until convergence criteria are met. Because of the local nature of this procedure, a good initial guess is crucial for high-dimensional and highly nonlinear system dynamics. In our case, we use the searched trajectories from Stage 1 in Section 4.5 as the initial guess.

We first adopt a simple zero-order hold temporal discretization of the dynamics Eq. (4.10) using Euler integration:

$$x_{k+1}^{(i)} = x_k^{(i)} + \dot{x}_k^{(i)} \Delta t. \quad (4.11)$$

Second, we linearize  $\dot{x}_k^{(i)}$  around prior states  $\bar{x}_k^{(i)}$  and actions  $\bar{u}_k^{(i)}$ :

$$\dot{x}_k^{(i)} \approx A_k(x_k^{(i)} - \bar{x}_k^{(i)}) + B_k(u_k^{(i)} - \bar{u}_k^{(i)}) + f^{(i)}(\bar{x}_k^{(i)}, \bar{u}_k^{(i)}), \quad (4.12)$$

where  $A_k$  and  $B_k$  are the partial derivative matrices of  $f^{(i)}$  with respect to  $x_k^{(i)}$  and  $u_k^{(i)}$  evaluated at  $\bar{x}_k^{(i)}, \bar{u}_k^{(i)}$ . Because we encode  $\hat{f}_a^{(i)}$  using fully-differentiable DNNs, the partial derivatives can be efficiently computed analytically, e.g., by using autograd in PyTorch (Paszke et al., 2019).

Third, we linearize  $\hat{f}_a^{(j)}$  around our prior states  $\bar{x}_k^{(i)}$  for all neighboring robots  $j \in \text{neighbor}(i)$ :

$$\hat{f}_a^{(j)} \approx C_k^{(j)}(x_k^{(i)} - \bar{x}_k^{(i)}) + \hat{f}_a^{(j)}(\mathbf{r}_{\mathcal{I}}^{(i)}(\bar{x}_k^{(i)})), \quad (4.13)$$

where  $C_k^{(j)}$  is the derivative matrix of  $\hat{f}_a^{(j)}$  (the learned interaction function of robot  $j$ , represented by DNNs) with respect to  $x_k^{(i)}$  evaluated at  $\bar{x}_k^{(i)}$ ; and  $\mathbf{r}_{\mathcal{I}}^{(i)}(\bar{x}_k^{(i)})$  is the ordered sequence of relative states as defined in Eq. (4.3) but using the fixed prior state  $\bar{x}_k^{(i)}$  rather than decision variable  $x_k^{(i)}$  in Eq. (4.2).

We now formulate a convex program, one per robot:

$$\min_{\mathcal{X}_{\text{sol}}^{(i)}, \mathcal{U}_{\text{sol}}^{(i)}} \sum_{t=0}^T \|u_k^{(i)}\|^2 + \lambda_1 \|x_T^{(i)} - x_f^{(i)}\|_{\infty} + \lambda_2 \delta \quad (4.14)$$

subject to:

$$\left\{ \begin{array}{ll} \text{robot dynamics Eq. (4.11) and Eq. (4.12)} & i \in [1, N] \\ u_k^{(i)} \in \mathcal{U}^{\mathcal{I}(i)} & i \in [1, N] \\ x_k^{(i)} \in \mathcal{X}_{\delta}^{\mathcal{I}(i)} & i \in [1, N], \delta \geq 0 \\ \langle \bar{p}_k^{(ij)}, p_k^{(i)} - \bar{p}_k^{(i)} \rangle \geq r^{(\mathcal{I}(i)\mathcal{I}(j))} \|\bar{p}_k^{(ij)}\|_2 & i < j, j \in [2, N] \\ x_0^{(i)} = x_s^{(i)} & i \in [1, N] \\ |C_k^{(j)}(x_k^{(i)} - \bar{x}_k^{(i)})| \leq b_{fa} & i < j, j \in [2, N] \\ |x_k^{(i)} - \bar{x}_k^{(i)}| \leq b_x; |u_k^{(i)} - \bar{u}_k^{(i)}| \leq b_u & i \in [1, N] \end{array} \right.$$

where  $\mathcal{X}_{\delta}^{\mathcal{I}(i)}$  is the state space increased by  $\delta$  in each direction, the linearized robot dynamics are similar to (Foust et al., 2020; Nakka et al., 2021), and the convexified inter-robot collision constraint is from (Morgan, Subramanian, et al., 2016). We use soft constraints for reaching the goal (with weight  $\lambda_1$ ) and the state space (with weight  $\lambda_2$ ), and trust regions around  $\bar{x}_k^{(i)}, \bar{u}_k^{(i)}$ , and the neighbors' interaction forces for numerical stability. Interaction forces are constrained in (4.14) because  $\hat{f}_a^{(i)}$  is part of the state space  $\mathcal{X}^{\mathcal{I}(i)}$ .

We solve these convex programs sequentially and they converge to a locally optimal solution (Morgan, Subramanian, et al., 2016). For the first iteration, we linearize around the trajectory computed during Stage 1 of our motion planner while subsequent iterations linearize around the solution of the previous iteration (Lines 18 to 21 in Algorithm 4.1). It is possible to implement Algorithm 4.1 in a distributed fashion similar to prior work (Morgan, Subramanian, et al., 2016).

#### 4.6 Interaction-Aware Tracking Controller

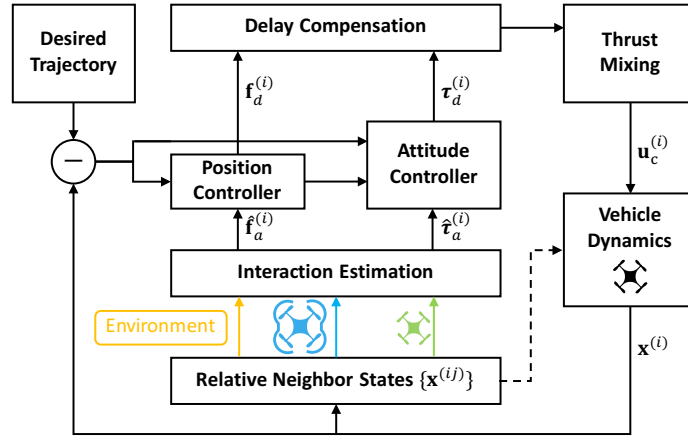


Figure 4.4: Hierarchy of control and planning blocks with information flow for commands and sensor feedback. We use different colors to represent heterogeneous neighbors. Note that the neighbors will influence the vehicle dynamics (dashed arrow).

Given arbitrary desired trajectories, including ones that have not been computed using the method presented in Section 4.5, we augment the hierarchical multirotor controller discussed in Chapter 2 with delay compensation. The overall control structure is shown in Fig. 4.4.

First, we recap a multi-agent integral variant of the hierarchical control structure given in Chapter 2. For the  $i^{\text{th}}$  agent, we have the following control law:

$$f_d^{(i)} = m^{(i)} \ddot{p}_r^{(i)} - m^{(i)} g - K^{(i)} s^{(i)} - \hat{f}_a^{(i)}$$

$$\tau_d^{(i)} = J^{(i)} \dot{\omega}_r^{(i)} - J^{(i)} \omega^{(i)} \times \omega_r^{(i)} - K_\omega^{(i)} (\omega^{(i)} - \omega_r^{(i)}) - \Gamma_\omega^{(i)} \int (\omega^{(i)} - \omega_r^{(i)}) dt - \hat{\tau}_a^{(i)}.$$

Since most of theoretical analyses are covered in Chapter 2 and Chapter 3. Here we highlight three main differences of the above controller from previous chapters:

**Decentralized, heterogeneous, and interaction-aware.** Note that in this chapter, we consider a *heterogeneous multi-agent* setting, where the nominal dynamics (e.g.,

$m^{(i)}, J^{(i)}, K^{(i)}$  depend on the agent type  $\mathcal{I}(i)$ . Moreover, the learned interaction term  $\hat{f}_a^{(i)}$  and  $\hat{\tau}_a^{(i)}$  are computed in real time in a decentralized manner:

$$\begin{bmatrix} \hat{f}_a^{(i)} \\ \hat{\tau}_a^{(i)} \end{bmatrix} = \rho_{\mathcal{I}(i)} \left( \sum_{k=1}^K \sum_{x^{(ij)} \in \mathbf{r}_{\text{type}_k}^{(i)}} \phi_{\mathcal{I}(j)}(x^{(ij)}) \right).$$

**Integral control.** We also enhance the controller in Chapter 2 by two integral terms. In the position control law, the composite variable  $s^{(i)}$  contains an integral term such that  $\dot{\tilde{p}}^{(i)} + 2\Lambda^{(i)}\tilde{p}^{(i)} + \Lambda^{(i)2} \int \tilde{p}^{(i)}$ . The attitude control law also contains an integral term  $\Gamma_{\omega}^{(i)} \int (\omega^{(i)} - \omega_r^{(i)}) dt$ . Adding integral terms makes the tracking error bound depend on the maximum time-derivative of the approximation error (i.e.,  $\frac{d}{dt}(\hat{f}_a^{(i)} - f_a^{(i)})$  or  $\frac{d}{dt}(\hat{\tau}_a^{(i)} - \tau_a^{(i)})$ ). See more details in G. Shi, Hönig, et al. (2022).

**Delay compensation.** With the desired output wrench  $\eta_{o,d}^{(i)} = [T_d^{(i)}; \tau_d^{(i)}]$  from the position and attitude controller, the next step is to compute the control command  $u_c^{(i)}$ . Note that we can not directly control the motor speed  $u^{(i)}$  because of the first order delay model in Eq. (4.1c). Therefore, we apply the following delay compensation method:

$$u_c^{(i)} = B_0^{(i)\dagger} \left( \eta_{o,d}^{(i)} + \frac{\dot{\eta}_{o,d}^{(i)}}{\lambda^{(i)}} \right). \quad (4.15)$$



The key idea of Eq. (4.15) is to “predict” the future of  $\eta_{o,d}^{(i)}$  and account for it according to the delay constant  $\lambda^{(i)}$ . If the delay constant is accurate, one can show that the actual output wrench  $\eta_o^{(i)}$  will exponentially converge to the desired one (X. Shi et al., 2020). With small modelling errors on  $\lambda^{(i)}$ , it can robustly cancel out some effects from delays and improve tracking performance in practice. Furthermore, it can handle not only first-order motor delay, but also signal transport delays (X. Shi et al., 2020). In case of the small quadrotors used in our experiments, such delays are on the same order of magnitude as the motor delay, thus making Eq. (4.15) essential for improving the control performance.

## 4.7 Experiments

We use quadrotors based on Bitcraze Crazyflie 2.0/2.1 (CF). Our small quadrotors are Crazyflie 2.X, which are small (9 cm rotor-to-rotor) and lightweight (34 g) products that are commercially available. Our large quadrotors use the Crazyflie 2.1 as control board on a larger frame with brushed motors (model: Parrot Mini Drone), see Table 4.1 for a summary of physical parameters. We use the Crazyswarm (Preiss

et al., 2017) package to control multiple Crazyflies simultaneously. Each quadrotor is equipped with a single reflective marker for position tracking at 100 Hz using a motion capture system. The nonlinear controller, extended Kalman filter, and neural network evaluation are running on-board the STM32 microcontroller.

Table 4.1: System identification of the used quadrotors.

	Small	Large
		
Weight	34 g	67 g
Max Thrust	65 g	145 g
Diameter	12 cm	19 cm
$\lambda$	16	16
$\psi_1(\hat{p}, \hat{v})$	$11.09 - 39.08\hat{p} - 9.53\hat{v} + 20.57\hat{p}^2 + 38.43\hat{p}\hat{v}$	$44.10 - 122.51\hat{p} - 36.18\hat{v} + 53.11\hat{p}^2 + 107.68\hat{p}\hat{v}$
$\psi_2(\hat{f}, \hat{v})$	$0.5 + 0.12\hat{f} - 0.41\hat{v} - 0.002\hat{f}^2 - 0.043\hat{f}\hat{v}$	$0.56 + 0.06\hat{f} - 0.6\hat{v} - 0.0007\hat{f}^2 - 0.015\hat{f}\hat{v}$
$\psi_3(\hat{p}, \hat{v})$	$-9.86 + 3.02\hat{p} + 26.72\hat{v}$	$-29.91 + 8.1\hat{p} + 65.2\hat{v}$

For the controller, we implement the delay compensation Eq. (4.15) by numerical differentiation. The baseline controller is identical (including the chosen gains) to our proposed controller except that the interaction force for the baseline is set to zero. The baseline controller is much more robust and efficient than the well-tuned nonlinear controller in the CrazySwarm package, which cannot safely execute the close-proximity flight shown in Fig. 4.1(c) and requires at least 60 cm safety distance (Hönig et al., 2018).

For data collection, we use the  $\mu$ SD card extension board and store binary encoded data roughly every 10 ms. Each dataset is timestamped using the on-board microsecond timer and the clocks are synchronized before takeoff using broadcast radio packets. The drift of the clocks of different Crazyflies can be ignored for our short flight times (less than 2 min).

### Calibration and System Identification of Different Robots

Prior to learning the residual terms  $f_a^{(i)}$  and  $\tau_a^{(i)}$ , we first calibrate the nominal dynamics model  $\Phi^{(i)}(x, u)$ . We found that existing motor thrust models (Bitcraze, 2015; Förster, 2015) are not very accurate, because they only consider a single motor

and ignore the effect of the battery state of charge. We calibrate each Crazyflie by mounting the whole quadrotor upside-down on a load cell (model TAL221) which is directly connected to the Crazyflie via a custom extension board using a 24-bit ADC (model HX711). The upside-down mounting avoids contamination of our measurements with downwash-related forces. We use a 100 g capacity load cell for the small quadrotor and a 500 g capacity load cell for the large quadrotor. We randomly generate desired PWM motor signals (identical for all 4 motors) and collect the current battery voltage, PWM signals, and measured force. We use this data to find three polynomial functions:  $\psi_1$ ,  $\psi_2$ , and  $\psi_3$ . The first  $\hat{f} = \psi_1(\hat{p}, \hat{v})$  computes the force of a single rotor given the normalized PWM signal  $\hat{p}$  and the normalized battery voltage  $\hat{v}$ . This function is only required for the data collection preparation in order to compute  $f_a^{(i)}$ . The second  $\hat{p} = \psi_2(\hat{f}, \hat{v})$  computes the required PWM signal  $\hat{p}$  given the desired force  $\hat{f}$  and current battery voltage  $\hat{v}$ . Finally,  $\hat{f}_{\max} = \psi_3(\hat{p}, \hat{v})$  computes the maximum achievable force  $\hat{f}_{\max}$ , given the current PWM signal  $\hat{p}$  and battery voltage  $\hat{v}$ . The last two functions are important at runtime for outputting the correct force as well as for thrust mixing when motors are saturated (Faessler et al., 2017).

We use the same measurement setup with the load cell to establish the delay model of  $T_d^{(i)}$  with a square wave PWM signal. While the delay model is slightly asymmetric in practice, we found that our symmetric model Eq. (4.1c) is a good approximation. All results are summarized in Table 4.1. We use the remaining parameters ( $J$ , thrust-to-torque ratio) from the existing literature (Förster, 2015).

## Data Collection

Table 4.2: 12 scenarios for data collection.

Scenario	S	S $\rightarrow$ S	L $\rightarrow$ S	{S, S} $\rightarrow$ S
Model	$\rho_{\text{small}}(\phi_{\text{env}})$	$\rho_{\text{small}}(\phi_{\text{env}} + \phi_{\text{small}})$	$\rho_{\text{small}}(\phi_{\text{env}} + \phi_{\text{large}})$	$\rho_{\text{small}}(\phi_{\text{env}} + \phi_{\text{small}} + \phi_{\text{small}})$
Scenario	{S, L} $\rightarrow$ S	{L, L} $\rightarrow$ S	L	S $\rightarrow$ L
Model	$\rho_{\text{small}}(\phi_{\text{env}} + \phi_{\text{small}} + \phi_{\text{large}})$	$\rho_{\text{small}}(\phi_{\text{env}} + \phi_{\text{large}} + \phi_{\text{large}})$	$\rho_{\text{large}}(\phi_{\text{env}})$	$\rho_{\text{large}}(\phi_{\text{env}} + \phi_{\text{small}})$
Scenario	L $\rightarrow$ L	{S, S} $\rightarrow$ L	{S, L} $\rightarrow$ L	{L, L} $\rightarrow$ S
Model	$\rho_{\text{large}}(\phi_{\text{env}} + \phi_{\text{large}})$	$\rho_{\text{large}}(\phi_{\text{env}} + \phi_{\text{small}} + \phi_{\text{small}})$	$\rho_{\text{large}}(\phi_{\text{env}} + \phi_{\text{small}} + \phi_{\text{large}})$	$\rho_{\text{large}}(\phi_{\text{env}} + \phi_{\text{large}} + \phi_{\text{large}})$

Recall that in Eq. (4.9), we need to learn  $2K$  neural networks for  $K$  types of robots. In our experiments, we consider two types of quadrotors (small and large) and also the environment (mainly ground effect and air drag), as shown in Example 4.1.

Therefore, we have 5 neural networks to be learned:

$$\rho_{\text{small}}, \rho_{\text{large}}, \phi_{\text{small}}, \phi_{\text{large}}, \phi_{\text{env}}, \quad (4.16)$$

where we do not have  $\rho_{\text{env}}$  because the aerodynamical force acting on the environment is not interesting for our purpose. To learn these 5 neural networks, we fly the heterogeneous swarm in 12 different scenarios (see Table 4.2) to collect labeled  $f_a^{(i)}$  and  $\tau_a^{(i)}$  data for each robot. For instance, Example 4.1 (as depicted in Fig. 3.1(a)) corresponds to the “{S, S} → L” scenario in Table 4.2, where the large robot has two small robots and the environment as its neighbors.

We utilize two types of data collection tasks: random walk and swapping. For random walk, we implement a simple reactive collision avoidance approach based on artificial potentials on-board each Crazyfly (Khatib, 1985). The host computer randomly selects new goal points within a cube for each vehicle at a fixed frequency. These goal points are used as an attractive force, while neighboring drones contribute a repulsive force. For swapping, the drones are placed in different horizontal planes on a cylinder and tasked to move to the opposite side. All the drones are vertically aligned for one time instance, causing a large interaction force. The random walk data helps us to explore the whole space quickly, while the swapping data ensures that we have data for a specific task of interest. Note that for both random walk and swapping, the drones also move close to the ground, to collect sufficient data for learning the ground effect. The collected data covers drone flying speeds from 0 to 2 m/s, where 7% are with relatively high speeds ( $\geq 0.5$  m/s) to learn the aerodynamic drag. For both task types, we varied the scenarios listed in Table 4.2.

To learn the 5 DNNs in Eq. (4.16), for each robot  $i$  in each scenario, we collect the timestamped states  $x^{(i)} = [p^{(i)}; v^{(i)}; R^{(i)}; \omega^{(i)}]$ . We then compute  $y^{(i)}$  as the observed value of  $f_a^{(i)}$  and  $\tau_a^{(i)}$ . We compute  $f_a^{(i)}$  and  $\tau_a^{(i)}$  using Eq. (4.4), where the nominal dynamics  $\Phi^{(i)}$  is calculated based on our system identification. With  $\Phi^{(i)}$ ,  $y^{(i)}$  is computed by  $\dot{x}^{(i)} - \Phi^{(i)}$ , where  $\dot{x}^{(i)}$  is estimated by the five-point numerical differentiation method. Note that the control delay  $\lambda^{(i)}$  is also considered when we compute  $f_a^{(i)}$  and  $\tau_a^{(i)}$ . Our training data consists of sequences of  $(\{\mathbf{r}_{\text{type}_1}^{(i)}, \dots, \mathbf{r}_{\text{type}_K}^{(i)}\}, y^{(i)})$  pairs, where  $\mathbf{r}_{\text{type}_k}^{(i)} = \{x^{(ij)} | j \in \text{neighbor}(i) \text{ and } \mathcal{I}(j) = \text{type}_k\}$  is the set of the relative states of the type- $k$  neighbors of  $i$ . We have the following loss function for robot  $i$  in each scenario (see Table 4.2 for the detailed



model structure in each scenario):

$$\left\| \rho_{\mathcal{I}(i)} \left( \sum_{k=1}^K \sum_{x^{(ij)} \in \mathbf{r}_{\text{type}_k}^{(i)}} \phi_{\mathcal{I}(j)}(x^{(ij)}) \right) - y^{(i)} \right\|_2^2, \quad (4.17)$$

and we stack all the robots' data in all scenarios and train on them together. There are 1.4 million pairs in the full dataset.

In practice, we found the unmodeled torque  $\|\tau_a^{(i)}\|$  is very small (smaller by two orders of magnitude than the feedback term in the attitude controller), so we only learn  $f_a^{(i)}$ . We compute the relative states from our collected data as  $x^{(ij)} = [p^{(j)} - p^{(i)}; v^{(j)} - v^{(i)}] \in \mathbb{R}^6$  (i.e., relative position and relative velocity both in the world frame), since the attitude information  $R$  and  $\omega$  are not dominant for  $f_a^{(i)}$ . If the type of neighbor  $j$  is “environment”, we set  $p^{(j)} = 0$  and  $v^{(j)} = 0$ . In this work, we only learn the  $z$ -component of  $f_a^{(i)}$  since we found the other two components,  $x$  and  $y$ , are much smaller and less varied, and do not significantly alter the nominal dynamics. In data collection, the rooted means and standard deviations of the squared values of  $f_{a,x}$ ,  $f_{a,y}$ , and  $f_{a,z}$  are  $1.6 \pm 2.5$ ,  $1.2 \pm 2.2$ ,  $5.0 \pm 8.9$  grams, respectively (for reference, the weights of the small and large drones are 34 g and 67 g). Therefore, the output of our learning model in Eq. (4.9) is a scalar to approximate the  $z$ -component of the unmodeled force function  $f_a^{(i)}$ .

## Learning Results and Ablation Analysis

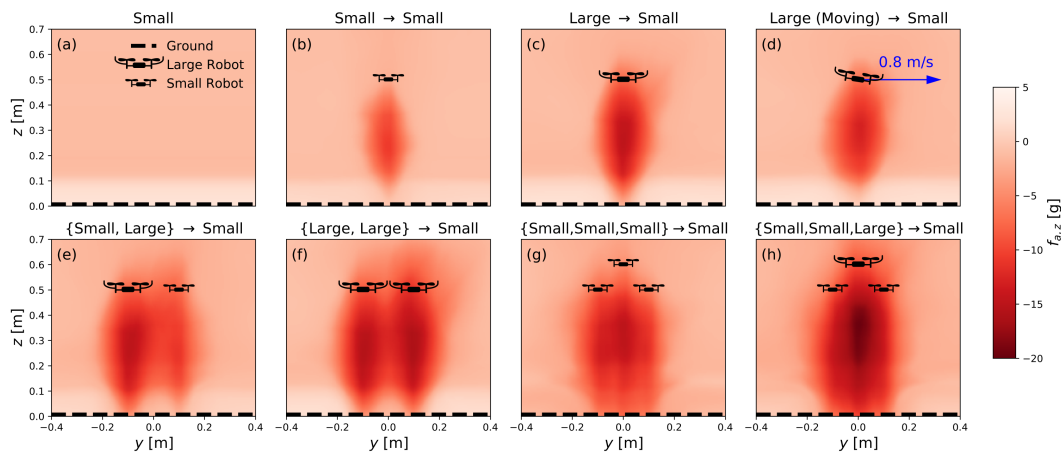


Figure 4.5:  $f_{a,z}$  prediction from the trained  $\{\rho_{\text{small}}, \rho_{\text{large}}, \phi_{\text{small}}, \phi_{\text{large}}, \phi_{\text{env}}\}$  networks. Each heatmap gives the prediction of  $f_{a,z}$  of a vehicle in different horizontal and vertical (global) positions. The (global) position of neighboring drones are represented by drone icons.

Each scenario uses a trajectory with a duration around 1000 seconds. For each scenario, we equally split the total trajectory into 50 shorter pieces, where each one is about 20 seconds. Then we randomly choose 80% of these 50 trajectories for training and 20% for validation.

Our DNN functions of  $\phi$  ( $\phi_{\text{small}}, \phi_{\text{large}}, \phi_{\text{env}}$ ) have four layers with architecture  $6 \rightarrow 25 \rightarrow 40 \rightarrow 40 \rightarrow H$ , and our  $\rho$  DNNs ( $\rho_{\text{small}}, \rho_{\text{large}}$ ) also have  $L = 4$  layers, with architecture  $H \rightarrow 40 \rightarrow 40 \rightarrow 40 \rightarrow 1$ . We use the ReLU function as the activation operator, and we use PyTorch (Paszke et al., 2019) for training and implementation of spectral normalization (see Chapter 2) of all these five DNNs. During training we iterate all the data 20 times for error convergence.

Note that  $H$  is the dimension of the hidden state. To study the effect of  $H$  on learning performance, we use three different values of  $H$  and the mean validation errors for each  $H$  are shown in Table 4.3. Meanwhile, we also study the influence of the number of layers by fixing  $H = 20$  and changing  $L$ , which is the number of layers of all  $\rho$  nets and  $\phi$  nets. For  $L = 3$  or  $L = 5$ , we delete or add a  $40 \rightarrow 40$  layer for all  $\rho$  nets and  $\phi$  nets, before their last layers. We repeat all experiments three times to get mean and standard deviation. As depicted in Table 4.3, we found that the average learning performance (mean validation error) is not sensitive to  $H$ , but larger  $H$  results in higher variance, possibly because using a bigger hidden space (larger  $H$ ) leads to a more flexible encoding that is harder to train reliably. In terms of the number of layers, four layers are significantly better than five (which tends to overfit data), and slighter better than three. To optimize performance, we finally choose  $H = 20$  and use four-layer neural networks, which can be efficiently evaluated on-board. We notice that  $H$  and  $L$  are the most important parameters, and the learning performance is not sensitive to other parameters such as the number of weights in intermediate layers.

Figure 4.5 depicts the prediction of  $f_{a,z}$ , trained with flight data from the 12 scenarios listed in Table 4.2. The color encodes the magnitude of  $\hat{f}_{a,z}$  for a single small multirotor positioned at different global  $(y, z)$  coordinates. The big/small black drone icons indicate the (global) coordinates of neighboring big/small multirotors, and the dashed line located at  $z = 0$  represents the ground. All quadrotors are in the same  $x$ -plane. For example, in Fig. 4.5(e), one large quadrotor is hovering at  $(y = -0.1, z = 0.5)$  and one small quadrotor is hovering at  $(y = 0.1, z = 0.5)$ . If we place a third small quadrotor at  $(y = 0, z = 0.3)$ , it would estimate  $\hat{f}_{a,z} = -10$  g as indicated by the red color in that part of the heatmap. Similarly, in Fig. 4.5(a)

the small multirotor only has the environment as a special neighbor. If the small multirotor is hovering at  $(y = 0, z = 0.05)$ , it would estimate  $\hat{f}_{a,z} = 5$  g, which is mainly from the ground effect. All quadrotors are assumed to be stationary except for Fig. 4.5(d), where the one neighbor is moving at 0.8 m/s.

Table 4.3: Ablation analysis. Top:  $L = 4$  and  $H$  varies. Bottom:  $H = 20$  and  $L$  varies. The error is the mean squared error (MSE) between  $f_{a,z}$  prediction and the ground truth.

$H$	10	20	40
Validation Error	6.70±0.05	6.42±0.18	6.63±0.35
$L$	3	4	5
Validation Error	6.52±0.17	6.42±0.18	7.21±0.28

We observe that the interaction between quadrotors is non-stationary and sensitive to relative velocity. In Fig. 4.5(d), the vehicle’s neighbor is moving, and the prediction becomes significantly different from Fig. 4.5(c), where the neighbor is just hovering. To further understand the importance of relative velocity, we retrain neural networks neglecting relative velocity and the mean squared validation error degrades by 18%, from 6.42 to 7.60. We can also observe that the interactions are not a simple superposition of different pairs. For instance, Fig. 4.5(g) is significantly more complex than a simple superposition of Fig. 4.5(a) plus three (b), i.e.,  $\rho_{\text{small}}(\phi_{\text{env}}) + \rho_{\text{small}}(\phi_{\text{small}}) + \rho_{\text{small}}(\phi_{\text{small}}) + \rho_{\text{small}}(\phi_{\text{small}})$ . The maximum gap between Fig. 4.5(g) and the superposition version is 11.4 g. Moreover, we find that the ground effect and the downwash effect from a neighboring multirotor interact in an intriguing way. For instance, in Fig. 4.5(b), the downwash effect is “mitigated” as the vehicle gets closer to the ground. Finally, we observe that the large quadrotors cause significantly higher interaction forces than the small ones (see Fig. 4.5(e)), which further emphasizes the importance of our heterogeneous modeling.

Note that in training we only have data from 1-3 vehicles (see Table 4.2). Our approach can generalize well to a larger swarm system. In Fig. 4.5, predictions for a 4-vehicle team (as shown in Fig. 4.5(g,h)) are still reliable. Moreover, our models work well in real flight tests with 5 vehicles (see Fig. 4.9) and even 16 vehicles (see Fig. 4.1).

### Motion Planning with Aerodynamics Coupling

We implement Algorithm 4.1 in Python using PyTorch 1.5 (Paszke et al., 2019) for automatic gradient computation, CVXPY 1.0 (Diamond and Boyd, 2016) for

convex optimization, and GUROBI 9.0 (Gurobi Optimization, 2020) as underlying solver. To simulate the tracking performance of the planned trajectories, we also implement a nonlinear controller, which uses the planned controls as feed-forward term. We compare trajectories that were planned with a learned model of  $f_{a,z}$  with trajectories without such a model (i.e.,  $f_{a,z} = 0$ ) using Algorithm 4.1 with identical parameters. At test time, we track the planned trajectories with our controller, and forward propagate the dynamics with our learned model of  $f_{a,z}$ .

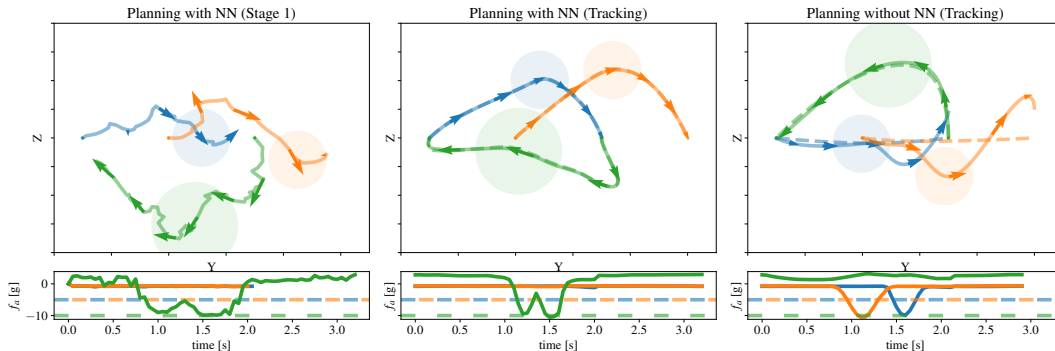


Figure 4.6: Example motion planning result for a three-robot swapping task in 2D (blue and orange: small robots; green: large robot). Top row:  $yz$ -state space plot, where the arrows indicate the velocities every second, and the circles show the robot collision boundary shape at the middle of the task. Bottom row: interaction force for each robot over time (dashed: desired limit per robot). Left: Sampling-based motion planning with neural network to compute trajectories where the large robots moves below the small robots. Middle: Refined trajectories using SCP (dashed) and tracked trajectories (solid). Right: Planned trajectories when ignoring interaction forces (dashed) and tracked trajectories (solid). In this case, a dangerous configuration is chosen where the large robot flies on top of the small robots, exceeding their disturbance limits of 5 g.

We visualize an example in Fig. 4.6, where two small and one large robots are tasked with exchanging positions. We focus on the 2D case in the  $yz$ -plane to create significant interaction forces between the robots. The first stage of Algorithm 4.1 uses sampling-based motion planning to identify the best homeomorphism class where the small multirotors fly on top of the large multirotor (the interaction forces would require more total energy the other way around). However, the robots do not reach their goal state exactly and motions are jerky (Fig. 4.6, left). The second stage uses SCP to refine the motion plan such that robots reach their goal and minimize the total control effort (Fig. 4.6, middle). The planned trajectory can be tracked without significant error and the interaction forces are very small for the two small quadrotors and within the chosen bound of 10 g for the large quadrotor. We compare

this solution to one where we do not consider the interaction forces between robots by setting  $f_{a,z} = 0$  in Algorithm 4.1. The planned trajectories tend to be shorter (Fig. 4.6, right, dashed lines) in that case. However, when tracking those trajectories, significant tracking errors occur and the interaction forces are outside their chosen bounds of 5 g for the small multirotors.

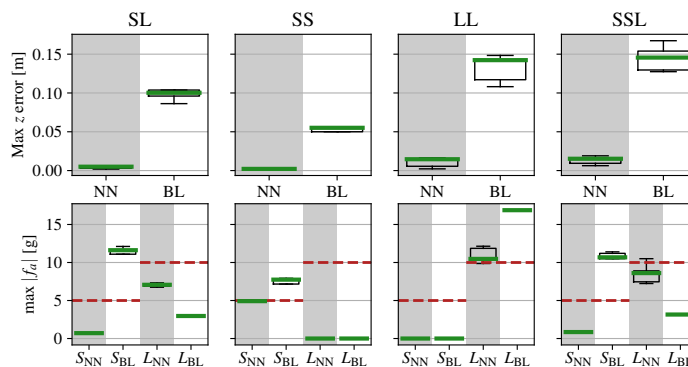


Figure 4.7: Motion planning results for different scenarios (e.g., SSL refers to two small robots and one large robot) comparing planning without neural network (BL) and planning with neural network (NN) over 5 trials. Top: Worst-case tracking error. Ignoring the interaction force can result in errors of over 10 cm. Bottom: Worst-case interaction force for small and large quadrotors. The baseline has significant violations of the interaction force bounds, e.g., the SL case might create interaction forces greater than 10 g for the small quadrotor.

We empirically evaluated the effect of planning with and without considering interaction forces in several scenarios, see Fig. 4.7. We found that ignoring the interaction forces results in significant tracking errors in all cases (top row). While this tracking error could be reduced when using our interaction-aware control law, the interaction forces are in some cases significantly over their desired limit. For example, in the small/large, small/small/large, and large/large cases, the worst-case interaction forces were consistently nearly double the limit (red line, bottom row). In practice, such large disturbances can cause instabilities or even a total loss of control, justifying the use of an interaction-aware motion planner.

### Control Performance in Flight Tests

We study the flight performance improvements on swapping tasks with varying number of quadrotors. For each case, robots are initially arranged in a circle when viewed from above but at different  $z$ -planes and are tasked with moving linearly to the opposite side of the circle in their plane. During the swap, all vehicles align vertically at one point in time with vertical distances of 0.2 m to 0.3 m between neighbors. The tasks are similar, but not identical to the randomized swapping tasks

used in data collection because different parameters (locations, transition times) are used.

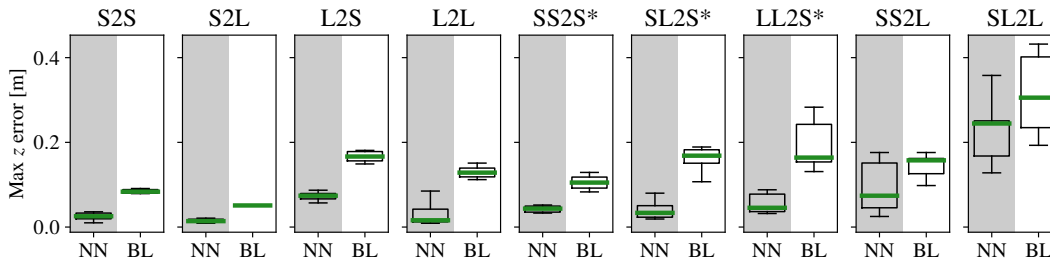


Figure 4.8: Flight test results comparing our solution with learned interaction compensation (NN) with the baseline (BL) in different scenarios. For each case, robots are initially arranged in a circle when viewed from above but at different  $z$ -planes and are tasked with moving linearly to the opposite side of the circle in their plane. For each swap, we compute the worst-case  $z$ -error of the lowest quadrotor and plot the data over six swaps.

Our results are summarized in Fig. 4.8 for various combinations of two and three multirotors, where we use “XY2Z” to denote the swap task with robots of type X and Y at the top and a robot of type Z at the bottom. We compute a box plot with median (green line) and first/third quartile (box) of the maximum  $z$ -error (repeated over 6 swaps). In some cases, the downwash force was so large that we upgraded the motors of the small quadrotor to improve the best-case thrust-to-weight ratio to 2.6. Such modified quadrotors are indicated as “S\*”. We also verified that the  $x$ - and  $y$ -error distributions are similar across the different controllers and omit those numbers for brevity.

Our controller improves the median  $z$ -error in all cases and in most cases this improvement is statistically significant. For example, in the “L2S” case, where a large multirotor is on top of a small multirotor for a short period of time, the median  $z$ -error is reduced from 17 cm to 7 cm.

To estimate the limits of our learning generalization, we test our approach on larger teams. First, we consider a team of five robots, where two large robots move on a circle in the horizontal plane and three small robots move on circle in the vertical plane such that the two circles form intertwined rings. In this case, the  $f_{a,z}$  prediction is accurate and the maximum  $z$ -error can be reduced significantly using our neural network prediction, see Fig. 4.9 for an example. Second, we consider a team of 16 robots moving on three intertwined rings as shown in Fig. 4.1(b,c). Here, two large and four small robots move on an ellipsoid in the horizontal plane, and five robots move on circles in different vertical planes. In this case, robots can

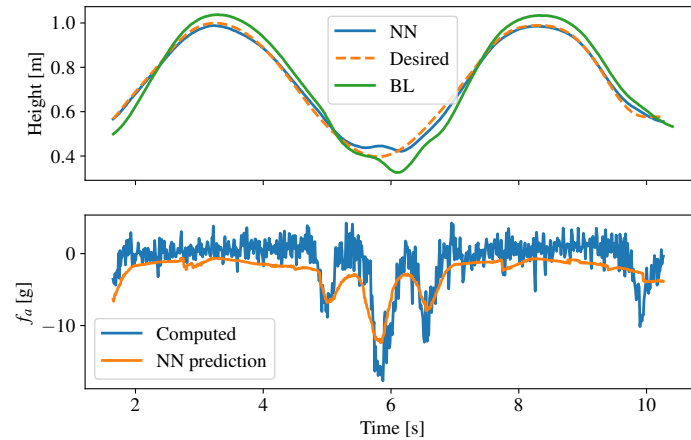


Figure 4.9: Generalization to a team of five multirotors. Three small multirotor move in a vertical ring and two large multirotor move in a horizontal ring. The maximum  $z$ -error of a small multirotor in the vertical ring with powerful motors is reduced from 10 cm to 5 cm and  $f_a$  is predicted accurately.

have significantly more neighbors (up to 15) compared to the training data (up to 2), making the prediction of  $f_{a,z}$  relatively less accurate. However, the maximum  $z$ -error of a small multirotor in one of the vertical rings with powerful motors is still reduced from 15 cm to 10 cm.

We note that a conceptually-simpler method is to estimate and compensate for  $f_{a,z}$  online without learning. However, online estimation will not only introduce significant delays, but also be very noisy especially in close-proximity flight. Our learning-based method has no delay (because it directly *predicts*  $f_{a,z}$  at the current time step), and considerably mitigates the noise due to the use of spectral normalization and delay-free filtering in the training process. In experiments, we observe that the online estimation and compensate method would quickly crash the drone.

## References

- Bentley, Jon Louis (1975). *Multidimensional binary search trees used for associative searching*. In: *Communications of the ACM* 18.9, pp. 509–517. DOI: [10.1145/361002.361007](https://doi.org/10.1145/361002.361007). URL: <http://doi.acm.org/10.1145/361002.361007>.
- Berg, Jur van den, Stephen J. Guy, Ming C. Lin, and Dinesh Manocha (2009). *Reciprocal n-body collision avoidance*. In: *International Symposium on Robotics Research*. Vol. 70, pp. 3–19. DOI: [10.1007/978-3-642-19457-3\\_1](https://doi.org/10.1007/978-3-642-19457-3_1). URL: [https://doi.org/10.1007/978-3-642-19457-3\\_1](https://doi.org/10.1007/978-3-642-19457-3_1).
- Bitcraze (2015). *Crazyfly 2.0 thrust investigation*. URL: <https://wiki.bitcraze.io/misc:investigations:thrust>.

- Čáp, Michal, Peter Novák, Alexander Kleiner, and Martin Selecký (2015). *Prioritized planning algorithms for trajectory coordination of multiple mobile robots*. In: *IEEE Transactions on Automation Science and Engineering* 12.3, pp. 835–849. DOI: 10.1109/TASE.2015.2445780. URL: <https://doi.org/10.1109/TASE.2015.2445780>.
- Cheng, Richard, Abhinav Verma, Gábor Orosz, Swarat Chaudhuri, Yisong Yue, and Joel Burdick (2019). *Control regularization for reduced variance reinforcement learning*. In: *International Conference on Machine Learning*, pp. 1141–1150. URL: <http://proceedings.mlr.press/v97/cheng19a.html>.
- Chung, Soon-Jo, Aditya A. Paranjape, Philip M. Dames, Shaojie Shen, and Vijay Kumar (2018). *A survey on aerial swarm robotics*. In: *IEEE Transactions on Robotics* 34.4, pp. 837–855. DOI: 10.1109/TRO.2018.2857475. URL: <https://doi.org/10.1109/TRO.2018.2857475>.
- Csáji, Balázs Csanád et al. (2001). *Approximation with artificial neural networks*. In: *Faculty of Sciences, Eötvös Loránd University, Hungary* 24.48, p. 7.
- Debord, Mark, Wolfgang Hönig, and Nora Ayanian (2018). *Trajectory planning for heterogeneous robot teams*. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 7924–7931. DOI: 10.1109/IROS.2018.8593876. URL: <https://doi.org/10.1109/IROS.2018.8593876>.
- Diamond, Steven and Stephen Boyd (2016). *CVXPY: A Python-embedded modeling language for convex optimization*. In: *Journal of Machine Learning Research* 17.83, pp. 1–5. URL: <http://jmlr.org/papers/v17/15-408.html>.
- Dinh, Quoc Tran and Moritz Diehl (2010). *Local convergence of sequential convex programming for nonconvex optimization*. In: *Recent Advances in Optimization and Its Applications in Engineering*. Springer, pp. 93–102. DOI: 10.1007/978-3-642-12598-0\_9. URL: [http://link.springer.com/10.1007/978-3-642-12598-0\\_9](http://link.springer.com/10.1007/978-3-642-12598-0_9).
- Du, Xintong, Carlos E. Luis, Marijan Vukosavljev, and Angela P. Schoellig (2019). *Fast and in sync: Periodic swarm patterns for quadrotors*. In: *IEEE International Conference on Robotics and Automation*, pp. 9143–9149. DOI: 10.1109/ICRA.2019.8794017. URL: <https://doi.org/10.1109/ICRA.2019.8794017>.
- Faessler, Matthias, Davide Falanga, and Davide Scaramuzza (2017). *Thrust mixing, saturation, and body-rate control for accurate aggressive quadrotor flight*. In: *IEEE Robotics and Automation Letters* 2.2, pp. 476–482. DOI: 10.1109/LRA.2016.2640362. URL: <https://doi.org/10.1109/LRA.2016.2640362>.
- Förster, Julian (2015). *System identification of the Crazyflie 2.0 nano quadcopter*. MA thesis. Zurich: ETH Zurich. DOI: 10.3929/ethz-b-000214143.
- Foust, Rebecca, Soon-Jo Chung, and Fred Y. Hadaegh (2020). *Optimal guidance and control with nonlinear dynamics using sequential convex programming*. In: *Journal of Guidance, Control, and Dynamics* 43.4, pp. 633–644. DOI: 10.2514/1.G004590. URL: <https://doi.org/10.2514/1.G004590>.



- Gurobi Optimization, LLC (2020). *Gurobi optimizer reference manual*. URL: <http://www.gurobi.com>.
- Hauser, Kris and Yilun Zhou (2016). *Asymptotically optimal planning by feasible kinodynamic planning in a state-cost space*. In: *IEEE Transactions on Robotics* 32.6, pp. 1431–1443. DOI: 10.1109/TR0.2016.2602363. URL: <https://doi.org/10.1109/TR0.2016.2602363>.
- Hönig, Wolfgang, James A. Preiss, T. K. Satish Kumar, Gaurav S. Sukhatme, and Nora Ayanian (2018). *Trajectory planning for quadrotor swarms*. In: *IEEE Transactions on Robotics* 34.4, pp. 856–869. DOI: 10.1109/TR0.2018.2853613. URL: <https://doi.org/10.1109/TR0.2018.2853613>.
- Jain, Karan P., Trey Fortmuller, Jaeseung Byun, Simo A. Mäkiharju, and Mark W. Mueller (2019). *Modeling of aerodynamic disturbances for proximity flight of multirotors*. In: *International Conference on Unmanned Aircraft Systems*, pp. 1261–1269. DOI: 10.1109/ICUAS.2019.8798116. URL: <https://doi.org/10.1109/ICUAS.2019.8798116>.
- Johannink, Tobias et al. (2019). *Residual reinforcement learning for robot control*. In: *IEEE International Conference on Robotics and Automation*, pp. 6023–6029. DOI: 10.1109/ICRA.2019.8794127. URL: <https://doi.org/10.1109/ICRA.2019.8794127>.
- Karaman, Sertac and Emilio Frazzoli (2011). *Sampling-based algorithms for optimal motion planning*. In: *International Journal of Robotics Research* 30.7, pp. 846–894. DOI: 10.1177/0278364911406761. URL: <https://doi.org/10.1177/0278364911406761>.
- Khatib, Oussama (1985). *Real-time obstacle avoidance for manipulators and mobile robots*. In: *IEEE International Conference on Robotics and Automation*, pp. 500–505. DOI: 10.1109/ROBOT.1985.1087247. URL: <https://doi.org/10.1109/ROBOT.1985.1087247>.
- Kunz, Tobias and Mike Stilman (2015). *Kinodynamic RRTs with fixed time step and best-input extension are not probabilistically complete*. In: *International Workshop on the Algorithmic Foundations of Robotics*, pp. 233–244. DOI: 10.1007/978-3-319-16595-0\_14. URL: [https://doi.org/10.1007/978-3-319-16595-0\\_14](https://doi.org/10.1007/978-3-319-16595-0_14).
- Le, Hoang Minh, Andrew Kang, Yisong Yue, and Peter Carr (2016). *Smooth imitation learning for online sequence prediction*. In: *International Conference on Machine Learning*. Vol. 48, pp. 680–688. URL: <http://proceedings.mlr.press/v48/le16.html>.
- Luis, Carlos E. and Angela P. Schoellig (2019). *Trajectory generation for multiagent point-to-point transitions via distributed model predictive control*. In: *IEEE Robotics and Automation Letters* 4.2, pp. 375–382. DOI: 10.1109/LRA.2018.2890572. URL: <https://doi.org/10.1109/LRA.2018.2890572>.

- McKinnon, Christopher D. and Angela P. Schoellig (2019). *Learn fast, forget slow: safe predictive learning control for systems with unknown and changing dynamics performing repetitive tasks*. In: *IEEE Robotics and Automation Letters* 4.2, pp. 2180–2187. DOI: 10.1109/LRA.2019.2901638. URL: <https://doi.org/10.1109/LRA.2019.2901638>.
- Mellinger, Daniel, Aleksandr Kushleyev, and Vijay Kumar (2012). *Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams*. In: *IEEE International Conference on Robotics and Automation*, pp. 477–483. DOI: 10.1109/ICRA.2012.6225009. URL: <https://doi.org/10.1109/ICRA.2012.6225009>.
- Morgan, Daniel, Soon-Jo Chung, and Fred Y. Hadaegh (2014). *Model predictive control of swarms of spacecraft using sequential convex programming*. In: *Journal of Guidance, Control, and Dynamics* 37.6, pp. 1725–1740. URL: <https://doi.org/10.2514/1.G000218>.
- Morgan, Daniel, Giri P. Subramanian, Soon-Jo Chung, and Fred Y. Hadaegh (2016). *Swarm assignment and trajectory optimization using variable-swarm, distributed auction assignment and sequential convex programming*. In: *The International Journal of Robotics Research* 35.10, pp. 1261–1285.
- Nakka, Yashwanth Kumar, Anqi Liu, Guanya Shi, Anima Anandkumar, Yisong Yue, and Soon-Jo Chung (2021). *Chance-constrained trajectory optimization for safe exploration and learning of nonlinear Systems*. In: *IEEE Robotics and Automation Letters* 6.2, pp. 389–396. DOI: 10.1109/LRA.2020.3044033.
- Paszke, Adam et al. (2019). *PyTorch: An imperative style, high-performance deep learning library*. In: *Advances in Neural Information Processing Systems*, pp. 8024–8035. URL: <http://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library>.
- Preiss, James A., Wolfgang Hönig, Gaurav S. Sukhatme, and Nora Ayanian (2017). *Crazyswarm: A large nano-quadcopter swarm*. In: *IEEE International Conference on Robotics and Automation*, pp. 3299–3304. DOI: 10.1109/ICRA.2017.7989376. URL: <https://doi.org/10.1109/ICRA.2017.7989376>.
- Rivière, Benjamin, Wolfgang Hönig, Yisong Yue, and Soon-Jo Chung (2020). *GLAS: Global-to-local safe autonomy synthesis for multi-robot motion planning with end-to-end learning*. In: *IEEE Robotics and Automation Letters* 5.3, pp. 4249–4256. DOI: 10.1109/LRA.2020.2994035. URL: <https://doi.org/10.1109/LRA.2020.2994035>.
- Saveriano, Matteo, Yuchao Yin, Pietro Falco, and Dongheui Lee (2017). *Data-efficient control policy search using residual dynamics learning*. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4709–4715. DOI: 10.1109/IROS.2017.8206343. URL: <https://doi.org/10.1109/IROS.2017.8206343>.

- Scarselli, Franco, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini (2008). *The graph neural network model*. In: *IEEE Transactions on Neural Networks* 20.1, pp. 61–80.
- Shi, Guanya, Wolfgang Hönig, Xichen Shi, Yisong Yue, and Soon-Jo Chung (2022). *Neural-Swarm2: Planning and control of heterogeneous multirotor swarms using learned interactions*. In: *IEEE Transactions on Robotics* 38.2, pp. 1063–1079. DOI: [10.1109/TRO.2021.3098436](https://doi.org/10.1109/TRO.2021.3098436).
- Shi, Guanya, Xichen Shi, Michael O’Connell, Rose Yu, Kamyar Azizzadenesheli, Animashree Anandkumar, Yisong Yue, and Soon-Jo Chung (2019). *Neural Lander: Stable drone landing control using learned dynamics*. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 9784–9790. DOI: [10.1109/ICRA.2019.8794351](https://doi.org/10.1109/ICRA.2019.8794351).
- Shi, Xichen, Michael O’Connell, and Soon-Jo Chung (2020). *Numerical predictive control for delay compensation*. In: *arXiv preprint arXiv:2009.14450*.
- Shukla, Dhwanil and Narayanan Komerath (2018). *Multirotor drone aerodynamic interaction investigation*. In: *Drones* 2.4. ISSN: 2504-446X. DOI: [10.3390/drones2040043](https://doi.org/10.3390/drones2040043). URL: <https://www.mdpi.com/2504-446X/2/4/43>.
- Taylor, Andrew J., Victor D. Dorobantu, Hoang M. Le, Yisong Yue, and Aaron D. Ames (2019). *Episodic learning with control Lyapunov functions for uncertain robotic systems*. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 6878–6884. DOI: [10.1109/IRoS40897.2019.8967820](https://doi.org/10.1109/IRoS40897.2019.8967820). URL: <https://doi.org/10.1109/IRoS40897.2019.8967820>.
- Tolstaya, Ekaterina, Fernando Gama, James Paulos, George Pappas, Vijay Kumar, and Alejandro Ribeiro (2020). *Learning decentralized controllers for robot swarms with graph neural networks*. In: *The Conference on Robot Learning (CoRL)*, pp. 671–682.
- Zaheer, Manzil, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R. Salakhutdinov, and Alexander J. Smola (2017). *Deep sets*. In: *Advances in Neural Information Processing Systems*, pp. 3391–3401.
- Zhang, Chuxu, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V. Chawla (2019). *Heterogeneous graph neural network*. In: *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 793–803.

## Chapter 5

## NEURAL-FLY

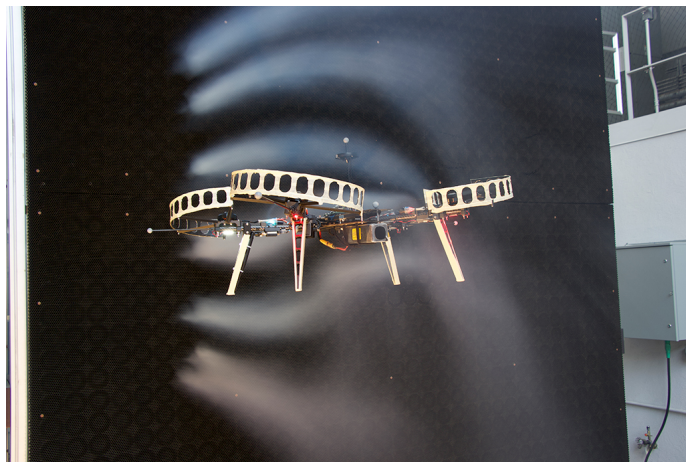


Figure 5.1: Visualization of wind effect for a drone flying in Caltech Real Weather Wind Tunnel.

In Chapter 2, we introduced a general mixed robot dynamics model  $M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = Bu + f$  where  $f$  is the unknown dynamics. We also introduced the corresponding drone dynamics model in Eq. (2.4). In Chapter 3 and Chapter 4, we discussed the case  $f = f(q, \dot{q}, u)$  and  $f = f(q, \dot{q}, \text{neighbors' states})$ , respectively.

In this chapter, we will move to a more challenging time-variant setting, where

$$f = f(q, \dot{q}, w(t))$$

and  $w(t)$  is an unknown environmental condition. For example,  $w$  in Fig. 5.1 is the unknown wind condition. In this case, the main challenge is to fast adapt the controller in changing environments in an efficient and reliable manner. This chapter is mainly based on the following paper<sup>1 2</sup>:

O'Connell, Michael, Guanya Shi, Xichen Shi, Kamyar Azizzadenesheli, Anima Anandkumar, Yisong Yue, and Soon-Jo Chung (2022). *Neural-Fly enables rapid learning for agile flight in strong winds*. In: *Science Robotics* 7.66, eabm6597. DOI: 10.1126/scirobotics.abm6597.

<sup>1</sup>Summary video: <https://youtu.be/TuF9teCZX0U>

<sup>2</sup>Data and code: <https://github.com/aerorobotics/neural-fly>

---

**Abstract.** Executing safe and precise flight maneuvers in dynamic high-speed winds is important for the ongoing commoditization of uninhabited aerial vehicles (UAVs). However, since the relationship between various wind conditions and its effect on aircraft maneuverability is not well understood, it is challenging to design effective robot controllers using traditional control design methods. We present Neural-Fly, a learning-based approach that allows rapid online adaptation by incorporating pre-trained representations through deep learning. Neural-Fly builds on two key observations that aerodynamics in different wind conditions share a common representation and that the wind-specific part lies in a low-dimensional space. To that end, Neural-Fly uses a proposed learning algorithm, Domain Adversarially Invariant Meta-Learning (DAIML), to learn the shared representation, only using 12 minutes of flight data. With the learned representation as a basis, Neural-Fly then uses a composite adaptation law to update a set of linear coefficients for mixing the basis elements. When evaluated under challenging wind conditions generated with the Caltech Real Weather Wind Tunnel with wind speeds up to 43.6 km/h (12.1 m/s), Neural-Fly achieves precise flight control with substantially smaller tracking error than state-of-the-art nonlinear and adaptive controllers. In addition to strong empirical performance, the exponential stability of Neural-Fly results in robustness guarantees. Finally, our control design extrapolates to unseen wind conditions, is shown to be effective for outdoor flights with only on-board sensors, and can transfer across drones with minimal performance degradation.

## 5.1 Introduction

The commoditization of uninhabited aerial vehicles (UAVs) requires that the control of these vehicles become more precise and agile. For example, drone delivery requires transporting goods to a narrow target area in various weather conditions.

Unmodeled and often complex aerodynamics (e.g.,  $f_a$  and  $\tau_a$  discussed in Chapter 2) are among the most notable challenges to precise flight control. Flying in windy environments (as shown in Fig. 5.1) introduces even more complexity because of the unsteady aerodynamic interactions between the drone, the induced airflow, and the wind. These unsteady and nonlinear aerodynamic effects substantially degrade the performance of conventional UAV control methods that neglect to account for them in the control design. Prior approaches partially capture these effects with simple linear or quadratic air drag models, which limit the tracking performance in

agile flight and cannot be extended to external wind conditions (Foehn et al., 2021; Faessler et al., 2018). Although more complex aerodynamic models can be derived from computational fluid dynamics (Ventura Diaz and Yoon, 2018), such modelling is often computationally expensive, and is limited to steady non-dynamic wind conditions. Adaptive control addresses this problem by estimating linear parametric uncertainty in the dynamical model in real time to improve tracking performance. Recent state-of-the-art in quadrotor flight control has used adaptive control methods that directly estimate the unknown aerodynamic force without assuming the structure of the underlying physics, but relying on high-frequency and low-latency control (Tal and Karaman, 2021; Mallikarjunan et al., 2012; Pravitra et al., 2020; Hanover et al., 2021). In parallel, there has been increased interest in data-driven modeling of aerodynamics (e.g., Torrente et al. (2021) and Chapters 3 and 4), however existing approaches cannot effectively adapt in changing or unknown environments such as time-varying wind conditions.

In this chapter, we present a data-driven approach called Neural-Fly, which is a deep-learning-based tracking controller that learns to quickly adapt to rapidly-changing wind conditions. Our method, depicted in Fig. 5.3, advances and offers insights into both adaptive flight control and deep-learning-based robot control. Our experimental demonstrates (e.g., Fig. 5.2) that Neural-Fly achieves centimeter-level position-error tracking of an agile and challenging trajectory in dynamic wind conditions on a standard UAV.

Our method has two main components: an offline learning phase and an online adaptive control phase used as real-time online learning. For the offline learning phase, we have developed Domain Adversarially Invariant Meta-Learning (DAIML) that learns a wind-condition-independent deep neural network (DNN) representation of the aerodynamics in a data-efficient manner. The output of the DNN is treated as a set of basis functions that represent the aerodynamic effects. This representation is adapted to different wind conditions by updating a set of linear coefficients that mix the output of the DNN. DAIML is data efficient and uses only 12 total minutes of flight data in just 6 different wind conditions to train the DNN. DAIML incorporates several key features which not only improve the data efficiency but also are informed by the downstream online adaptive control phase. In particular, DAIML uses spectral normalization (see Chapter 2) to control the Lipschitz property of the DNN to improve generalization to unseen data and provide closed-loop stability and robustness guarantees. DAIML also uses a discriminative network, which

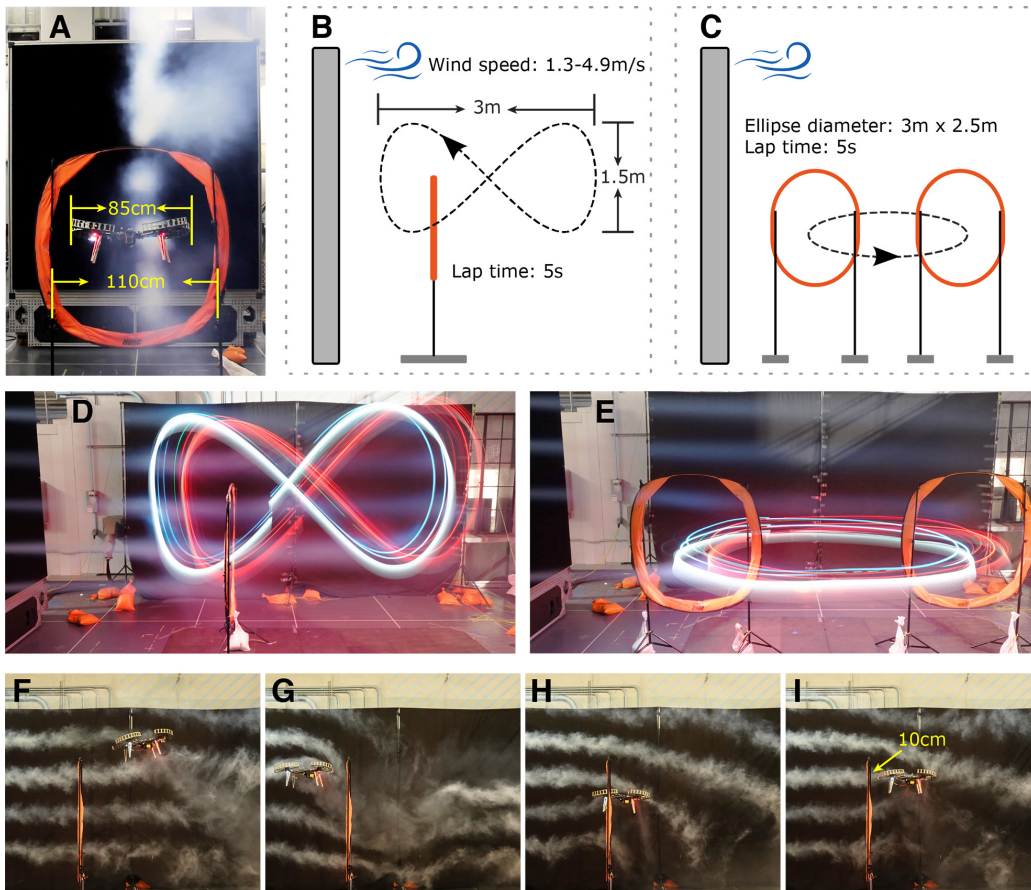


Figure 5.2: Agile flight through narrow gates. (A) Caltech Real Weather Wind Tunnel system, the quadrotor UAV, and the gate. In our flight tests, the UAV follows an agile trajectory through narrow gates, which are slightly wider than the UAV itself, under challenging wind conditions. (B-C) Trajectories used for the gate tests. In (B), the UAV follows a figure-8 through one gate, with wind speed 3.1 m/s or time-varying wind condition. In (C), the UAV follows an ellipse in the horizontal plane through two gates, with wind speed 3.1 m/s. (D-E) Long-exposure photos (with an exposure time of 5 s) showing one lap in two tasks. (F-I) High-speed photos (with a shutter speed of 1/200s) showing the moment the UAV passed through the gate and the interaction between the UAV and the wind.

ensures that the learned representation is wind-invariant and that the wind-dependent information is only contained in the linear coefficients that are adapted in the online control phase.

For the online adaptive control phase, we have developed a regularized composite adaptive control law, which we derived from a fundamental understanding of how the learned representation interacts with the closed-loop control system and which we support with rigorous theory. The adaptation law updates the wind-dependent linear coefficients using a composite of the position tracking error term and the

aerodynamic force prediction error term. Such a principled approach effectively guarantees stable and fast adaptation to any wind condition and robustness against imperfect learning. Although this adaptive control law could be used with a number of learned models, the speed of adaptation is further aided by the concise representation learned from DAIML.

Using Neural-Fly, we report an average improvement of 66 % over a nonlinear tracking controller, 42 % over an  $\mathcal{L}_1$  adaptive controller, and 35 % over an Incremental Nonlinear Dynamics Inversion (INDI) controller. These results are all accomplished using standard quadrotor UAV hardware, while running the PX4’s default regulation attitude control. Our tracking performance is competitive even compared to related work without external wind disturbances and with more complex hardware (for example, Tal and Karaman (2021) requires a 10-time higher control frequency and onboard optical sensors for direct motor speed feedback). We also compare Neural-Fly with two variants of our method: Neural-Fly-Transfer, which uses a learned representation trained on data from a different drone, and Neural-Fly-Constant, which only uses our adaptive control law with a trivial non-learning basis. Neural-Fly-Transfer demonstrates that our method is robust to changes in vehicle configuration and model mismatch. Neural-Fly-Constant,  $\mathcal{L}_1$ , and INDI all directly adapt to the unknown dynamics without assuming the structure of the underlying physics, and they have similar performance. Furthermore, we demonstrate that our method enables a new set of capabilities that allow the UAV to fly through low-clearance gates following agile trajectories in gusty wind conditions (Fig. 5.2).

## 5.2 Related Work

### Precise Quadrotor Control

Prior work on agile quadrotor control has achieved impressive results by considering aerodynamics (Tal and Karaman, 2021; Hanover et al., 2021; Torrente et al., 2021; Faessler et al., 2018). However, those approaches require specialized onboard hardware (Tal and Karaman, 2021), full custom flight control stacks (Tal and Karaman, 2021; Hanover et al., 2021), or cannot adapt to external wind disturbances (Torrente et al., 2021; Faessler et al., 2018). For example, state-of-the-art tracking performance has been demonstrated using incremental nonlinear dynamics inversion to estimate aerodynamic disturbance forces, with a root-mean-square tracking error of 6.6 cm and drone ground speeds up to 12.9 m/s (Tal and Karaman, 2021). However, Tal and Karaman (2021) relies on high-frequency control updates (500 Hz) and direct motor speed feedback using optical encoders to rapidly estimate external



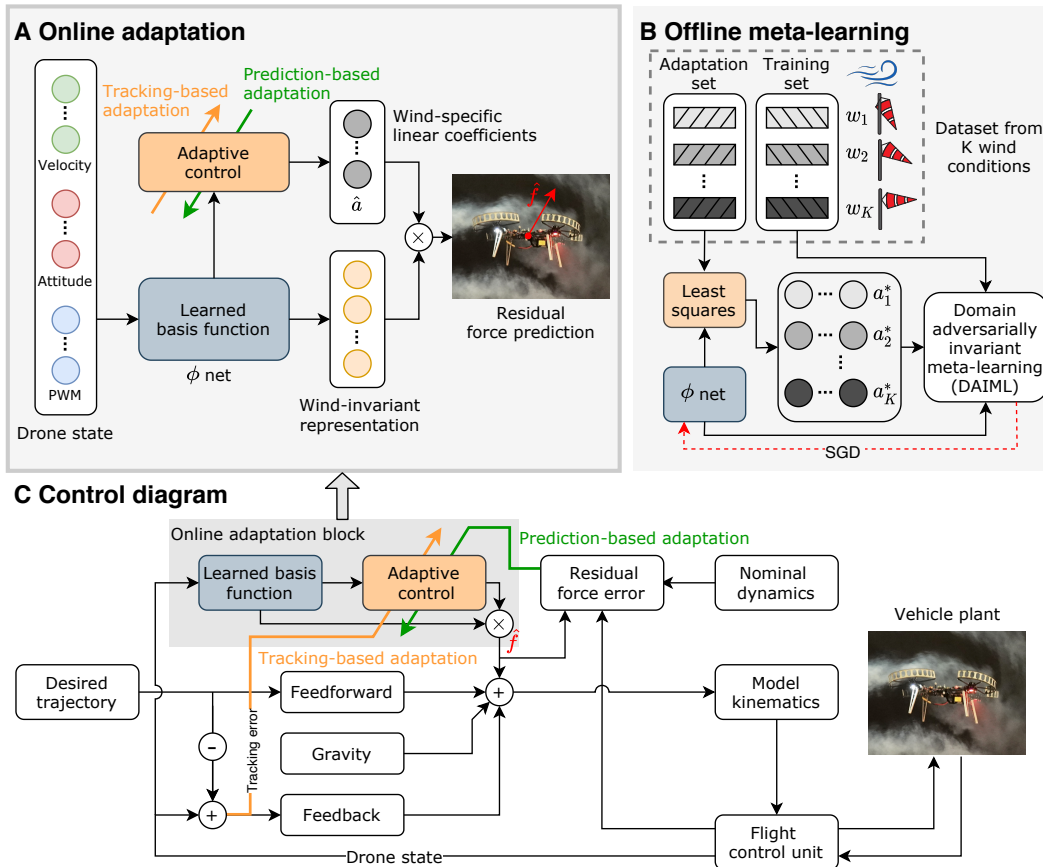


Figure 5.3: Offline meta-learning and online adaptive control design. (A) The online adaptation block in our adaptive controller. Our controller leverages the meta-trained basis function  $\phi$ , which is a wind-invariant representation of the aerodynamic effects, and uses composite adaptation (that is, including tracking-error-based and prediction-error-based adaptation) to update wind-specific linear weights  $\hat{a}$ . The output of this block is the wind-effect force estimate,  $\hat{f} = \phi \hat{a}$ . (B) The illustration of our meta-learning algorithm DAIML. We collected data from wind conditions  $\{w_1, \dots, w_K\}$  and applied Algorithm 5.1 to train the  $\phi$  net. (C) The diagram of our control method, where the grey part corresponds to (A). Interpreting the learned block as an aerodynamic force allows it to be incorporated into the feedback control easily.

disturbances. Both are challenging to deploy on standard systems. Hanover et al. (2021) simplifies the hardware setup and does not require optical motor speed sensors and has demonstrated state-of-the-art tracking performance. However, Hanover et al. (2021) relies on a high-rate  $\mathcal{L}_1$  adaptive controller inside a model predictive controller and uses a racing drone with a fully customized control stack. Torrente et al. (2021) leverages an aerodynamic model learned offline and represented as Gaussian Processes. However, Torrente et al. (2021) cannot adapt to unknown

or changing wind conditions and provides no theoretical guarantees. Another recent work focuses on deriving simplified rotor-drag models that are differentially flat (Faessler et al., 2018). However, Faessler et al. (2018) focuses on horizontal,  $xy$ -plane trajectories at ground speeds of 4 m/s without external wind, where the thrust is more constant than ours, achieves  $\sim 6$  cm tracking error, uses an attitude controller running at 4000 Hz, and is not extensible to faster flights as pointed out by Torrente et al. (2021).

### **Adaptive Control**

Adaptive control theory has been extensively studied for online control and identification problems with parametric uncertainty, for example, unknown linear coefficients for mixing known basis functions (Slotine and Li, 1991; Ioannou and Sun, 1996; Krstic et al., 1995; Narendra and Annaswamy, 2012; Farrell and Polycarpou, 2006; Wise et al., 2006). There are three common aspects of adaptive control which must be addressed carefully in any well-designed system and which we address in Neural-Fly: designing suitable basis functions for online adaptation, stability of the closed-loop system, and persistence of excitation, which is a property related to robustness against disturbances. These challenges arise due to the coupling between the unknown underlying dynamics and the online adaptation. This coupling precludes naive combinations of online learning and control. For example, gradient-based parameter adaptation has well-known stability and robustness issues as discussed in Slotine and Li (1991).

The basis functions play a crucial role in the performance of adaptive control, but designing or selecting proper basis functions might be challenging. A good set of basis functions should reflect important features of the underlying physics. In practice, basis functions are often designed using physics-informed modeling of the system, such as the nonlinear aerodynamic modeling in X. Shi, Spieler, et al. (2020). However, physics-informed modeling requires a tremendous amount of prior knowledge and human labor, and is often still inaccurate. Another approach is to use random features as the basis set, such as random Fourier features (Rahimi and Recht, 2007; Lale et al., 2021), which can model all possible underlying physics as long as the number of features is large enough. However, the high-dimensional feature space is not optimal for a specific system because many of the features might be redundant or irrelevant. Such suboptimality and redundancy not only increase the computational burden but also slow down the convergence speed of the adaptation process.

Given a set of basis functions, naive adaptive control designs may cause instability and fragility in the closed-loop system, due to the nontrivial coupling between the adapted model and the system dynamics. In particular, asymptotically stable adaptive control cannot guarantee robustness against disturbances and so exponential stability is desired. Even so, often, existing adaptive control methods only guarantee exponential stability when the desired trajectory is persistently exciting, by which information about all of the coefficients (including irrelevant ones) is constantly provided at the required spatial and time scales. In practice, persistent excitation requires either a succinct set of basis functions or perturbing the desired trajectory, which compromises tracking performance.

Recent multirotor flight control methods, including INDI (Tal and Karaman, 2021) and  $\mathcal{L}_1$  adaptive control, presented in (Mallikarjunan et al., 2012) and demonstrated inside a model predictive control loop in (Hanover et al., 2021), achieve good results by abandoning complex basis functions. Instead, these methods directly estimate the aerodynamic residual force vector. The residual force is observable, thus, these methods bypass the challenge of designing good basis functions and the associated stability and persistent excitation issues. However, these methods suffer from lag in estimating the residual force and encounter the the filter design performance trade of reduced lag versus amplified noise. Note that Neural-Fly-Constant only uses Neural-Fly’s composite adaptation law to estimate the residual force, and therefore, Neural-Fly-Constant also falls into this class of adaptive control structures. This chapter demonstrates that the inherent estimation lag in these existing methods limits performance on agile trajectories and in strong wind conditions.

Neural-Fly solves the aforementioned issues of basis function design and adaptive control stability, using newly developed methods for meta-learning and composite adaptation that can be seamlessly integrated together. Neural-Fly uses DAIML and flight data to learn an effective and compact set of basis functions, represented as a DNN. The regularized composite adaptation law uses the learned basis functions to quickly respond to wind conditions. Neural-Fly enjoys fast adaptation because of the conciseness of the feature space, and it guarantees closed-loop exponential stability and robustness without assuming persistent excitation.

Related to Neural-Fly, neural network based adaptive control has been researched extensively, but by and large was limited to shallow or single-layer neural networks without pretraining. Some early works focus on shallow or single-layer neural networks with unknown parameters which are adapted online (Farrell and Polycarpou,

2006; Nakanishi et al., 2002; Chen and Khalil, 1995; Johnson and Calise, 2003; Narendra and Mukhopadhyay, 1997). A recent work applies this idea to perform an impressive quadrotor flip (Bisheban and T. Lee, 2021). However, the existing neural network based adaptive control work does not employ multi-layer DNNs, and lacks a principled and efficient mechanism to pretrain the neural network before deployment. Instead of using shallow neural networks, recent trends in machine learning highly rely on DNNs due to their representation power (LeCun et al., 2015). In this chapter, we leverage modern deep learning advances to pretrain a DNN which represents the underlying physics compactly and effectively.

### **Multi-Environment Deep Learning for Robot Control**

Recently, researchers have been addressing the data and computation requirements for DNNs to help the field progress towards the fast online-learning paradigm. In turn, this progress has been enabling adaptable DNN-based control in dynamic environments. The most popular learning scheme in dynamic environments is meta-learning, or “learning-to-learn,” which aims to learn an efficient model from data across different tasks or environments (Finn et al., 2017; Hospedales et al., 2021). The learned model, typically represented as a DNN, ideally should be capable of rapid adaptation to a new task or an unseen environment given limited data. For robotic applications, meta-learning has shown great potential for enabling autonomy in highly-dynamic environments. For example, it has enabled quick adaptation against unseen terrain or slopes for legged robots (Nagabandi et al., 2018; Song et al., 2020), changing suspended payload for drones (Belkhale et al., 2021), and unknown operating conditions for wheeled robots (McKinnon and Schoellig, 2021).

In general, learning algorithms typically can be decomposed into two phases: offline learning and online adaptation. In the offline learning phase, the goal is to learn a model from data collected in different environments, such that the model contains shared knowledge or features across all environment, for example, learning aerodynamic features shared by all wind conditions. In the online adaptation phase, the goal is to adapt the offline-learned model, given limited online data from a new environment or a new task, for example, fine tuning the aerodynamic features in a specific wind condition.

There are two ways that the offline-learned model can be adapted. In the first class, the adaptation phase adapts the whole neural network model, typically using one or more gradient descent steps (Finn et al., 2017; Nagabandi et al., 2018; Belkhale

et al., 2021; Clavera et al., 2018). However, due to the notoriously data-hungry and high-dimensional nature of neural networks, for real-world robots it is still impossible to run such adaptation on-board as fast as the feedback control loop (e.g.,  $\sim 100\text{Hz}$  for quadrotor). Furthermore, adapting the whole neural network often lacks explainability and robustness and could generate unpredictable outputs that make the closed-loop unstable.

In the second class (including Neural-Fly), the online adaptation only adapts a relatively small part of the learned model, for example, the last layer of the neural network (O’Connell et al., 2021; McKinnon and Schoellig, 2021; Richards et al., 2021; Peng et al., 2021). The intuition is that, different environments share a common representation (e.g., the wind-invariant representation in Fig. 5.3(A)), and the environment-specific part is in a low-dimensional space (e.g., the wind-specific linear weight in Fig. 5.3(A)), which enables the real-time adaptation as fast as the control loop. In particular, the idea of integrating meta-learning with adaptive control is first presented in O’Connell et al. (2021), later followed by Richards et al. (2021). However, the representation learned in O’Connell et al. (2021) is ineffective and the tracking performance in O’Connell et al. (2021) is similar as the baselines; Richards et al. (2021) focuses on a planar and fully-actuated rotorcraft simulation without experiment validation and there is no stability or robustness analysis. Neural-Fly instead learns an effective representation using a new meta-learning algorithm called DAIML, demonstrates state-of-the-art tracking performance on real drones, and achieves non-trivial stability and robustness guarantees.

Another popular deep-learning approach for control in dynamic environments is robust policy learning via domain randomization (J. Lee et al., 2020; Tobin et al., 2017; Ramos et al., 2019). The key idea is to train the policy with random physical parameters such that the controller is robust to a range of conditions. For example, the quadrupedal locomotion controller in J. Lee et al. (2020) retains its robustness over challenging natural terrains. However, robust policy learning optimizes average performance under a broad range of conditions rather than achieving precise control by adapting to specific environments.

### 5.3 Problem Statement

In this chapter, we consider the general robot dynamics model given in Eq. (2.1) in Chapter 2:

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = u + \underbrace{f(q, \dot{q}, w)}_{\text{unknown}}. \quad (5.1)$$

Most importantly,  $f(q, \dot{q}, w)$  incorporates unmodeled dynamics, and  $w \in \mathbb{R}^m$  represents the underlying environmental conditions, which is potentially time-variant. Specifically, in this chapter,  $w$  represents the wind profile.

Neural-Fly can be broken into two main stages, the offline meta-learning stage and the online adaptive control stage. These two stages build a model of the unknown dynamics of the form

$$f(q, \dot{q}, w) \approx \phi(q, \dot{q})a(w), \quad (5.2)$$

where  $\phi$  is a basis or representation function shared by all wind conditions and captures the dependence of the unmodeled dynamics on the robot state, and  $a$  is a set of linear coefficients that is updated for each condition. In the Appendix (Section 5.7), we prove that the decomposition  $\phi(q, \dot{q})a(w)$  exists for any analytic function  $f(q, \dot{q}, w)$ . In the offline meta-learning stage, we learn  $\phi$  as a DNN using our meta-learning algorithm DAIML. This stage results in learning  $\phi$  as a wind-invariant representation of the unmodeled dynamics, which generalizes to new trajectories and new wind conditions. In the online adaptive control stage, we adapt the linear coefficients  $a$  using adaptive control. Our adaptive control algorithm is a type of composite adaptation and was carefully designed to allow for fast adaptation while maintaining the global exponential stability and robustness of the closed loop system. The offline learning and online control architectures are illustrated in Fig. 5.3(B) and Fig. 5.3(A,C), respectively.

#### 5.4 Offline Meta-Learning

In this section, we will present the methodology and details of learning the representation function  $\phi$ . In particular, we will first introduce the goal of meta-learning, motivate the proposed algorithm DAIML by the observed domain shift problem from the collected dataset, and finally discuss key algorithmic details.

For notional simplicity, we define  $x = [q; \dot{q}]$ . The offline meta-learning phase optimizes an efficient representation  $\phi$  given a multi-environment dataset  $\mathcal{D} = \{D_{w_1}, \dots, D_{w_K}\}$ , where

$$D_{w_k} = \left\{ x_k^{(i)}, y_k^{(i)} = f(x_k^{(i)}, w_k) + \epsilon_k^{(i)} \right\}_{i=1}^{N_k} \quad (5.3)$$

is the collection of  $N_k$  noisy input-output pairs with wind condition  $w_k$ .

### Meta-Learning Goal

Given the dataset, the goal of meta-learning is to learn a representation  $\phi(x)$ , such that for any wind condition  $w$ , there exists a latent variable  $a(w)$  which allows  $\phi(x)a(w)$  to approximate  $f(x, w)$  well. Formally, an optimal representation,  $\phi$ , solves the following optimization problem:

$$\min_{\phi, a_1, \dots, a_K} \sum_{k=1}^K \sum_{i=1}^{N_k} \left\| y_k^{(i)} - \phi(x_k^{(i)}) a_k \right\|^2, \quad (5.4)$$

where  $\phi(\cdot) : \mathbb{R}^{2n} \rightarrow \mathbb{R}^{n \times h}$  is the representation function and  $a_k \in \mathbb{R}^h$  is the latent linear coefficient. Note that the optimal weight  $a_k$  is specific to each wind condition, but the optimal representation  $\phi$  is shared by all wind conditions. We use a deep neural network (DNN) to represent  $\phi$ . In the Appendix (Section 5.7), we prove that for any analytic function  $f(x, w)$ , the structure  $\phi(x)a(w)$  can approximate  $f(x, w)$  with an arbitrary precision, as long as the DNN  $\phi$  has enough neurons. This result implies that the  $\phi$  solved from the optimization in Eq. (5.4) is a reasonable representation of the unknown dynamics  $f(x, w)$ .

### Domain Shift Problems

One challenge of the optimization in Eq. (5.4) is the *inherent domain shift* in  $x$  caused by the shift in  $w$ . Recall that during data collection we have a program flying the drone in different winds. The actual flight trajectories differ vastly from wind to wind because of the wind effect. Formally, the distribution of  $x_k^{(i)}$  varies between  $k$  because the underlying environment or context  $w$  has changed. For example, as depicted by Fig. 5.4(C), the drone pitches into the wind, and the average degree of pitch depends on the wind condition. Note that pitch is only one component of the state  $x$ . The domain shift in the whole state  $x$  is even more drastic.

Such inherent shifts in  $x$  bring challenges for deep learning. The DNN  $\phi$  may memorize the distributions of  $x$  in different wind conditions, such that the variation in the dynamics  $\{f(x, w_1), f(x, w_2), \dots, f(x, w_K)\}$  is reflected via the distribution of  $x$ , rather than the wind condition  $\{w_1, w_2, \dots, w_K\}$ . In other words, the optimization in Eq. (5.4) may lead to *over-fitting* and may not properly find a wind-invariant representation  $\phi$ .

To solve the domain shift problem, inspired by Ganin et al. (2016), we propose the following adversarial optimization framework:

$$\max_h \min_{\phi, a_1, \dots, a_K} \sum_{k=1}^K \sum_{i=1}^{N_k} \left( \left\| y_k^{(i)} - \phi(x_k^{(i)}) a_k \right\|^2 - \alpha \cdot \text{loss} \left( h(\phi(x_k^{(i)})), k \right) \right), \quad (5.5)$$

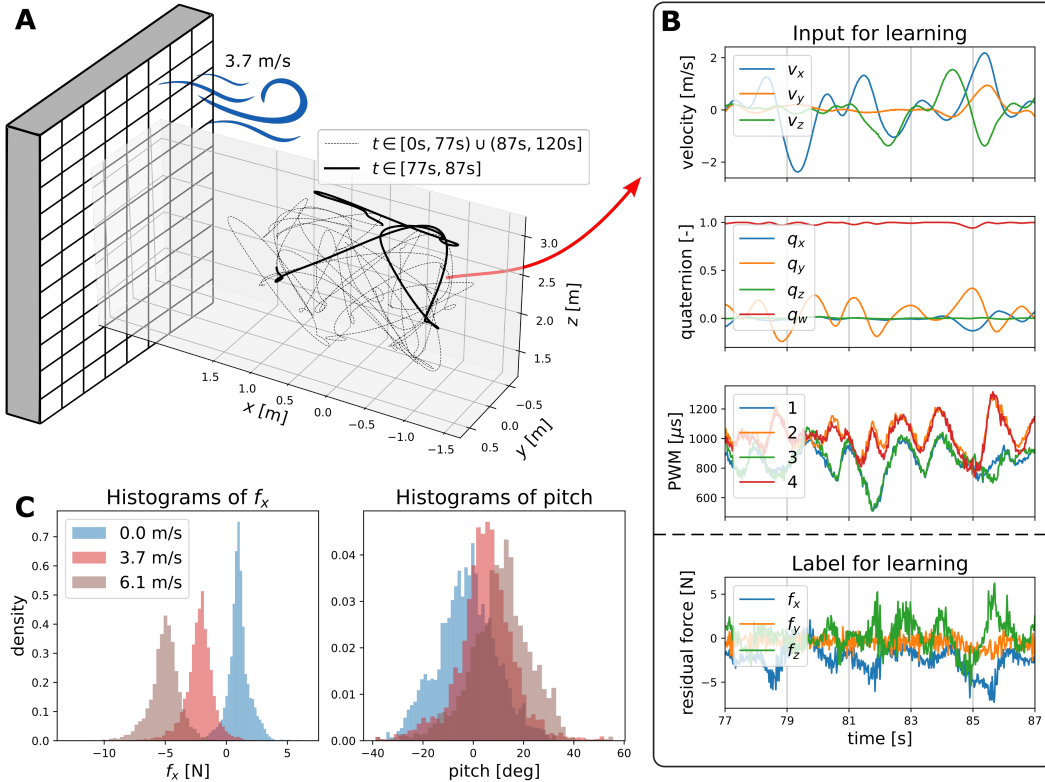


Figure 5.4: Training data collection. (A) The xyz position along a two-minute randomized trajectory for data collection with wind speed 8.3 km/h (3.7 m/s), in the Caltech Real Weather Wind Tunnel. (B) A typical 10-second trajectory of the inputs (velocity, attitude quaternion, and motor speed PWM command) and label (offline calculation of aerodynamic residual force) for our learning model, corresponding to the highlighted part in (A). (C) Histograms showing data distributions in different wind conditions. (C) Left: distributions of the  $x$ -component of the wind-effect force,  $f_x$ . This shows that the aerodynamic effect changes as the wind varies. (C) Right: distributions of the pitch, a component of the state used as an input to the learning model. This shows that the shift in wind conditions causes a distribution shift in the input.

where  $h$  is another DNN that works as a discriminator to predict the environment index out of  $K$  wind conditions,  $\text{loss}(\cdot)$  is a classification loss function (e.g., the cross entropy loss),  $\alpha \geq 0$  is a hyperparameter to control the degree of regularization,  $k$  is the wind condition index, and  $(i)$  is the input-output pair index. Intuitively,  $h$  and  $\phi$  play a zero-sum max-min game: the goal of  $h$  is to predict the index  $k$  directly from  $\phi(x)$  (achieved by the outer max); the goal of  $\phi$  is to approximate the label  $y_k^{(i)}$  while making the job of  $h$  harder (achieved by the inner min). In other words,  $h$  is a learned regularizer to remove the environment information contained in  $\phi$ . In our experiments, the output of  $h$  is a  $K$ -dimensional vector for the classification



probabilities of  $K$  conditions, and we use the cross entropy loss for  $\text{loss}(\cdot)$ , which is given as

$$\text{loss} \left( h(\phi(x_k^{(i)})), k \right) = - \sum_{j=1}^K \delta_{kj} \log \left( h(\phi(x_k^{(i)}))^\top e_j \right) \quad (5.6)$$

where  $\delta_{kj} = 1$  if  $k = j$  and  $\delta_{kj} = 0$  otherwise and  $e_j$  is the standard basis function.

### Design of the DAIML Algorithm

Finally, we solve the optimization problem in Eq. (5.5) by the proposed algorithm DAIML (described in Algorithm 5.1 and illustrated in Fig. 5.3(B)), which belongs to the category of gradient-based meta-learning (Hospedales et al., 2021), but with least squares as the adaptation step. DAIML contains three steps: (i) The adaptation step (Line 4-6) solves an least squares problem as a function of  $\phi$  on the adaptation set  $B^a$ . (ii) The training step (Line 7) updates the learned representation  $\phi$  on the training set  $B$ , based on the optimal linear coefficient  $a^*$  solved from the adaptation step. (iii) The regularization step (Line 8-9) updates the discriminator  $h$  on the training set.

We emphasize important features of DAIML: (i) After the adaptation step,  $a^*$  is a function of  $\phi$ . In other words, in the training step (Line 7), the gradient with respect to the parameters in the neural network  $\phi$  will backpropagate through  $a^*$ . Note that the least-square problem (Line 4) can be solved efficiently with a closed-form solution. Namely, DAIML is a Hessian-free meta-learning algorithm because of the closed-form inner loop. (ii) The normalization (Line 6) is to make sure  $\|a^*\| \leq \gamma$ , which improves the robustness of our adaptive control design. We also use spectral normalization in training  $\phi$ , to control the Lipschitz property of the neural network and improve generalizability (G. Shi, X. Shi, et al., 2019; G. Shi, Hönl, et al., 2022; Bartlett et al., 2017). (iii) We train  $h$  and  $\phi$  in an alternating manner. In each iteration, we first update  $\phi$  (Line 7) while fixing  $h$  and then update  $h$  (Line 9) while fixing  $\phi$ . However, the probability to update the discriminator  $h$  in each iteration is  $\eta \leq 1$  instead of 1, to improve the convergence of the algorithm (Goodfellow et al., 2014).

We further motivate the algorithm design using Fig. 5.4 and Fig. 5.5. Figure 5.4(A,B) shows the input and label from one wind condition, and Fig. 5.4(C) shows the distributions of the pitch component in input and the  $x$ -component in label, in different wind conditions. The distribution shift in label implies the importance of meta-learning and adaptive control, because the aerodynamic effect changes

**Algorithm 5.1:** Domain Adversarially Invariant Meta-Learning (DAIML)**Hyperparameter:**  $\alpha \geq 0, 0 < \eta \leq 1, \gamma > 0$ **Input:**  $\mathcal{D} = \{D_{w_1}, \dots, D_{w_K}\}$ **Initialize:** Neural networks  $\phi$  and  $h$ **Result:** Trained neural networks  $\phi$  and  $h$ 

```

1 repeat
2   Randomly sample  $D_{w_k}$  from  $\mathcal{D}$ 
3   Randomly sample two disjoint batches  $B^a$  (adaptation set) and  $B$  (training
      set) from  $D_{w_k}$ 
4   Solve the least squares problem  $a^*(\phi) = \arg \min_a \sum_{i \in B^a} \left\| y_k^{(i)} - \phi(x_k^{(i)})a \right\|^2$ 
5   if  $\|a^*\| > \gamma$  then
6      $a^* \leftarrow \gamma \cdot \frac{a^*}{\|a^*\|}$   $\triangleright$  normalization
7   Train DNN  $\phi$  using stochastic gradient descent (SGD) and spectral
      normalization with loss
          
$$\sum_{i \in B} \left( \left\| y_k^{(i)} - \phi(x_k^{(i)})a^* \right\|^2 - \alpha \cdot \text{loss} \left( h(\phi(x_k^{(i)})), k \right) \right)$$

8   if  $\text{rand}() \leq \eta$  then
9     Train DNN  $h$  using SGD with loss  $\sum_{i \in B} \text{loss} \left( h(\phi(x_k^{(i)})), k \right)$ 
10 until convergence

```

drastically as the wind condition switches. On the other hand, the distribution shift in input motivates the need of DAIML. Figure 5.5 depicts the evolution of the optimal linear coefficient ( $a^*$ ) solved from the adaptation step in DAIML, via the t-distributed stochastic neighbor embedding (t-SNE) dimension reduction, which projects the 12-dimensional vector  $a^*$  into 2-d. The distribution of  $a^*$  is more and more clustered as the number of training epochs increases. In addition, the clustering behavior in Fig. 5.5 has a concrete physical meaning: right top part of the t-SNE plot corresponds to a higher wind speed. These properties imply the learned representation  $\phi$  is indeed shared by all wind conditions, and the linear weight  $a$  contains the wind-specific information. Finally, note that  $\phi$  with 0 training epoch reflects random features, which cannot decouple different wind conditions as cleanly as the trained representation  $\phi$ . Similarly, as shown in Fig. 5.5, if we ignore the adversarial regularization term (by setting  $\alpha = 0$ ), different  $a^*$  vectors in different conditions are less disentangled, which indicates that the learned representation might be less robust and explainable.

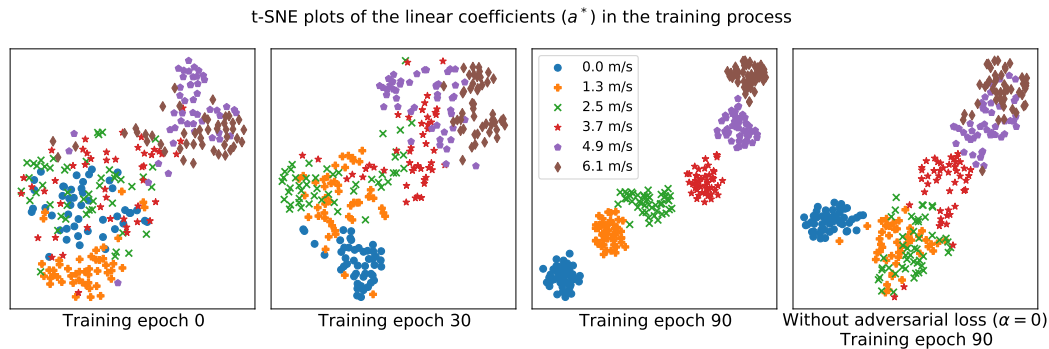


Figure 5.5: t-SNE plots showing the evolution of the linear weights ( $a^*$ ) during the training process. As the number of training epochs increases, the distribution of  $a^*$  becomes more clustered with similar wind speed clusters near each other. The clustering also has a physical meaning: after training convergence, the right top part corresponds to a higher wind speed. This suggests that DAIML successfully learned a basis function  $\phi$  shared by all wind conditions, and the wind-dependent information is contained in the linear weights. Compared to the case without the adversarial regularization term (using  $\alpha = 0$  in Algorithm 5.1), the learned result using our algorithm is also more explainable, in the sense that the linear coefficients in different conditions are more disentangled.

## 5.5 Online Adaptive Control and Stability Analysis

During the offline meta-training process, a least-squares fit is used to find a set of parameters  $a$  that minimizes the force prediction error for each data batch. The least-squares fit plays a crucial role in the offline training because it yields a Hessian-free and efficient meta-learning algorithm. However, during the online control phase, we are ultimately interested in minimizing the position tracking error. Thus, in this section, we propose a more sophisticated adaptation law for the linear coefficients based upon a Kalman-filter estimator. This formulation results in automatic gain tuning for the update law, which allows the controller to quickly estimate parameters with large uncertainty. We further boost this estimator into a composite adaptation law, that is the parameter update depends both on the prediction error in the dynamics model as well as on the tracking error, as illustrated in Fig. 5.3. This allows the system to quickly identify and adapt to new wind conditions without requiring persistent excitation. In turn, this enables online adaptation of the high dimensional learned models from DAIML.

## Robust Composite Adaptive Controller Design

Our online adaptive control algorithm can be summarized by the following control law, adaptation law, and covariance update equations, respectively.

$$u = \underbrace{M(q)\ddot{q}_r + C(q, \dot{q})\dot{q}_r + g(q)}_{\text{nominal nonlinear feedforward}} \underbrace{-Ks}_{\text{linear feedback}} \underbrace{-\phi(q, \dot{q})\hat{a}}_{\text{learning-based feedforward}} \quad (5.7)$$

$$\dot{\hat{a}} = \underbrace{-\lambda\hat{a}}_{\ell_2 \text{ regularization}} \underbrace{-P\phi^\top R^{-1}(\phi\hat{a} - y)}_{\text{prediction error term}} \underbrace{+P\phi^\top s}_{\text{tracking error term}} \quad (5.8)$$

$$\dot{P} = -2\lambda P + Q - P\phi^\top R^{-1}\phi P. \quad (5.9)$$

Note that the controller in Eq. (5.7) can be viewed as an augmentation of Eq. (2.3) in Chapter 2 with  $\hat{f} = \phi\hat{a}$ ,  $\hat{a}$  as the online linear-parameter update, and  $P$  as a covariance-like matrix used for automatic gain tuning. Recall that the composite error term  $s = \dot{\tilde{q}} + \Lambda\tilde{q}$  (see Eq. (2.2)), so the tracking error term in Eq. (5.8) updates  $\hat{a}$  based on the tracking error. The prediction error term in Eq. (5.8) instead updates  $\hat{a}$  based on the mismatch between model prediction  $\phi\hat{a}$  and  $y$ , which is the measured aerodynamic residual force. In this controller,  $K$ ,  $\Lambda$ ,  $R$ ,  $Q$ , and  $\lambda$  are all tunable positive definite gain matrices. The structure of this control law is illustrated in Fig. 5.3. Figure 5.3 also shows further quadrotor specific details for the implementation of our method. These blocks are discussed further in the ‘‘Implementation of Our Control Methods and Baselines’’ section.

Some readers may note that the regularization term, prediction error term, and covariance update, when taken alone, are in the form of a Kalman-Bucy filter. This Kalman-Bucy filter can be derived as the optimal estimator that minimizes the variance of the parameter error (Kalman and Bucy, 1961). The Kalman-Bucy filter perspective provides intuition for tuning the adaptive controller: the damping gain  $\lambda$  corresponds to how quickly the environment returns to the nominal conditions,  $Q$  corresponds to how quickly the environment changes, and  $R$  corresponds to the combined representation error  $d$  and measurement noise for  $y$ . However, naively combining this parameter estimator with the controller can lead to instabilities in the closed-loop system behavior unless extra care is taken in constraining the learned model and tuning the gains. Thus, we have designed our adaptation law to include a tracking error term, making Eq. (5.8) a composite adaptation law, guaranteeing stability of the closed-loop system (see Theorem 5.1), and in turn simplifying the gain tuning process. The regularization term allows the stability result to be independent of the persistent excitation of the learned model  $\phi$ , which is particularly

relevant when using high-dimensional learned representations. The adaptation gain and covariance matrix,  $P$ , acts as automatic gain tuning for the adaptive controller, which allows the controller to quickly adapt to when a new mode in the learned model is excited.

In terms of theoretical guarantees, the control law and adaptation law have been designed so that the closed-loop behavior of the system is robust to imperfect learning and time-varying wind conditions. Specifically, we formally define  $d(a, t)$  as the representation error and  $\epsilon(t)$  as the measurement noise as follows:

**Definition 5.1** (Representation error and measurement noise). *We define the representation error  $d(a, t)$  as the gap between the ground-truth unknown dynamics  $f$  and the learned representation  $\phi$  multiplying the “true” linear parameter  $a(t)$ , namely:*

$$f(q, \dot{q}, w(t)) = \phi(q, \dot{q})a(t) + d(a(t), t).$$

*Note that  $d(a, t)$  depends on  $a$ . For example, if  $\phi$  is **perfect**, there exists an oracle  $a^*$  such that  $f(q, \dot{q}, w(t)) = \phi(q, \dot{q})a^*(w(t))$  and  $d(a^*, t) = 0$ .*

*On the other hand, we define the measurement noise as*

$$\epsilon(t) = y(t) - f(q, \dot{q}, w(t))$$

*which is the gap between the ground-truth unknown dynamics  $f$  and its online noisy measurement  $y$ . Finally, we define the joint error  $\bar{\epsilon}$  as*

$$\bar{\epsilon} = \epsilon + d.$$

In the next subsection, our theory shows that the robot tracking error exponentially converges to an error ball whose size is upper bounded by  $\|d(a, t) + \epsilon(t)\| = \|\bar{\epsilon}\|$  and  $\|\dot{a}\|$  (i.e., how fast the wind condition changes) for arbitrary  $a(t)$ .

### Stability Analysis

Before discussing the main theorem and proof, let us consider the stability and robustness properties of the feedback controller without considering any specific adaptation law. Define the parameter estimation error  $\tilde{a} = \hat{a} - a$ . Taking the dynamics Eq. (5.1) and the control law Eq. (5.7), we find

$$M\dot{s} + (C + K)s = f - \phi\hat{a} = -\phi\tilde{a} + d(a, t). \quad (5.10)$$

As the stability analysis in Chapter 3, we can use the Lyapunov function  $V = s^\top M s$  under the assumption of bounded  $\tilde{a}$  to show that

$$\lim_{t \rightarrow \infty} \|s\| \leq \frac{\sup_t \|d(a, t) - \phi \tilde{a}\| \lambda_{\max}(M)}{\lambda_{\min}(K) \lambda_{\min}(M)} \quad \text{for arbitrary } a(t). \quad (5.11)$$

Taking this results alone, one might expect that any online estimator or learning algorithm will lead to good performance. However, the boundedness of  $\tilde{a}$  is not guaranteed (Slotine and Li (1991) discuss this topic thoroughly). Even though  $\|\tilde{a}\|$  is bounded, it may not converge and could be arbitrarily large. Therefore, in the theorem and proof below, we formally show that  $[s; \tilde{a}]$  will jointly converge.

**Theorem 5.1** (Main stability and robustness guarantee). *The adaptive controller given in Eqs. (5.7) to (5.9) guarantees that the tracking error  $\tilde{q}$  exponentially converges to the following error ball:*

$$\lim_{t \rightarrow \infty} \|\tilde{q}\| \leq \sup_t [C_1 \|d(a(t), t)\| + C_2 \|\epsilon(t)\| + C_3 (\lambda \|a(t)\| + \|\dot{a}(t)\|)] \quad (5.12)$$

for arbitrary  $a(t)$ , where  $C_1$ ,  $C_2$ , and  $C_3$  are three bounded constants depending on  $\phi$ ,  $R$ ,  $Q$ ,  $K$ ,  $\Lambda$ ,  $M$  and  $\lambda$ .

*Proof.* Combining the closed-loop dynamics Eq. (5.10) and the adaptation law Eq. (5.8), we have the following joint closed-loop dynamics for the tracking error  $s$  and the parameter error  $\tilde{a}$ :

$$\begin{bmatrix} M & 0 \\ 0 & P^{-1} \end{bmatrix} \begin{bmatrix} \dot{s} \\ \dot{\tilde{a}} \end{bmatrix} + \begin{bmatrix} C + K & \phi \\ -\phi^T & \phi^T R^{-1} \phi + \lambda P^{-1} \end{bmatrix} \begin{bmatrix} s \\ \tilde{a} \end{bmatrix} = \begin{bmatrix} d \\ \phi^T R^{-1} \bar{\epsilon} - \lambda P^{-1} a - P^{-1} \dot{a} \end{bmatrix}. \quad (5.13)$$

We also have the following  $P^{-1}$  dynamics from Eq. (5.9):

$$\frac{d}{dt} (P^{-1}) = -P^{-1} \dot{P} P^{-1} = 2\lambda P^{-1} - P^{-1} Q P^{-1} + \phi^T R^{-1} \phi. \quad (5.14)$$

In this proof, we rely on the fact that  $P^{-1}$  is both uniformly positive definite and uniformly bounded, that is, there exists some positive definite constant matrices  $A$  and  $B$  such that  $A \geq P^{-1} \geq B$  (see a proof when  $\phi$  is bounded and  $Q > 0$  in Dieci and Eirola (1994)).

Now consider the Lyapunov function  $V$  given by

$$V = \begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \underbrace{\begin{bmatrix} M & 0 \\ 0 & P^{-1} \end{bmatrix}}_M \begin{bmatrix} s \\ \tilde{a} \end{bmatrix}. \quad (5.15)$$

Plugging in Eqs. (5.13) and (5.14), this Lyapunov function has the following time derivative:

$$\begin{aligned}
\dot{V} &= 2 \begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \begin{bmatrix} M & 0 \\ 0 & P^{-1} \end{bmatrix} \begin{bmatrix} \dot{s} \\ \dot{\tilde{a}} \end{bmatrix} + \begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \begin{bmatrix} \dot{M} & 0 \\ 0 & \frac{d}{dt}(P^{-1}) \end{bmatrix} \begin{bmatrix} s \\ \tilde{a} \end{bmatrix} \\
&= -2 \begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \begin{bmatrix} C + K & \phi \\ -\phi^\top & \phi^\top R^{-1} \phi + \lambda P^{-1} \end{bmatrix} \begin{bmatrix} s \\ \tilde{a} \end{bmatrix} + 2 \begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \begin{bmatrix} d \\ \phi^\top R^{-1} \bar{\epsilon} - \lambda P^{-1} a - P^{-1} \dot{a} \end{bmatrix} + \\
&\quad \begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \begin{bmatrix} \dot{M} & 0 \\ 0 & \frac{d}{dt}(P^{-1}) \end{bmatrix} \begin{bmatrix} s \\ \tilde{a} \end{bmatrix} \\
&= - \underbrace{\begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \begin{bmatrix} 2K & 0 \\ 0 & \phi^\top R^{-1} \phi + P^{-1} Q P^{-1} \end{bmatrix} \begin{bmatrix} s \\ \tilde{a} \end{bmatrix}}_{\mathcal{K}} + 2 \underbrace{\begin{bmatrix} s \\ \tilde{a} \end{bmatrix}^\top \begin{bmatrix} d \\ \phi^\top R^{-1} \bar{\epsilon} - \lambda P^{-1} a - P^{-1} \dot{a} \end{bmatrix}}_D
\end{aligned} \tag{5.16}$$

where we used the fact  $\dot{M} - 2C$  is skew-symmetric again. Since  $M$ ,  $P^{-1}$ ,  $K$  and  $Q$  are both uniformly bounded and positive definite, and  $\phi^\top R^{-1} \phi$  is positive semi-definite, there exists some  $\alpha > 0$  such that

$$\mathcal{K} \geq 2\alpha \mathcal{M}, \quad \forall t. \tag{5.17}$$

Therefore we have

$$\dot{V} \leq -2\alpha V + 2\|D\| \sqrt{\frac{V}{\lambda_{\min}(\mathcal{M})}}, \quad \forall t. \tag{5.18}$$

Focusing on  $\sqrt{V}$  and following the analysis in Chapter 3, we have

$$\sqrt{V} \leq \sqrt{V(0)} e^{-\alpha t} + \frac{\|D\|}{\alpha \sqrt{\lambda_{\min}(\mathcal{M})}}. \tag{5.19}$$

Therefore the joint state  $[s; \tilde{a}]$  exponentially converges to the following error ball

$$\lim_{t \rightarrow \infty} \left\| \begin{bmatrix} s \\ \tilde{a} \end{bmatrix} \right\| \leq \sup_t \frac{\|D\|}{\alpha \cdot \lambda_{\min}(\mathcal{M})}. \tag{5.20}$$

Finally, the relation between  $s$  and  $\tilde{q}$  (Lemma 2.3) yields that  $\|\tilde{q}\|$  exponentially converges to

$$\lim_{t \rightarrow \infty} \|\tilde{q}\| \leq \sup_t \frac{\|D\|}{\alpha \cdot \lambda_{\min}(\mathcal{M}) \lambda_{\min}(\Lambda)} \tag{5.21}$$

which completes the proof.  $\square$

## 5.6 Experiments

In this section, we first discuss the experimental platform for data collection and experiments. Second, we discuss the implementations of our method and baselines. Third, we discuss several experiments to quantitatively compare the closed-loop trajectory-tracking performance of our methods to a nonlinear baseline method and two state-of-the-art adaptive flight control methods, and we observe our methods reduce the average tracking error substantially. In order to demonstrate the new capabilities brought by our methods, we present agile flight results in gusty winds, where the UAV must quickly fly through narrow gates that are only slightly wider than the vehicle. Finally, we show our methods are also applicable in outdoor agile tracking tasks without external motion capture systems.

### Experimental Platform

All of our experiments are conducted at Caltech’s Center for Autonomous Systems and Technologies (CAST). The experimental setup consists of an OptiTrack motion capture system with 12 infrared cameras for localization streaming position measurements at 50 Hz, a WiFi router for communication, the Caltech Real Weather Wind Tunnel for generating dynamic wind conditions, and a custom-built quadrotor UAV. For outdoor flight, the drone is also equipped with a Global Positioning System (GPS) module and an external antenna. We now discuss the design of the UAV and the wind condition in detail.

**UAV design.** We built a quadrotor UAV for our primary data collection and all experiments, shown in Fig. 5.2(A). The quadrotor weighs 2.6 kg with a thrust to weight ratio of 2.2. The UAV is equipped with a Pixhawk flight controller running PX4, an open-source commonly used drone autopilot platform (Meier et al., 2012). The UAV incorporates a Raspberry Pi 4 onboard computer running a Linux operation system, which performs real-time computation and adaptive control and interfaces with the flight controller through MAVROS, an open-source set of communication drivers for UAVs. State estimation is performed using the built-in PX4 Extended Kalman Filter (EKF), which fuses inertial measurement unit (IMU) data with global position estimates from OptiTrack motion capture system (or the GPS module for outdoor flight tasks). The UAV platform features a wide-X configuration, measuring 85 cm in width, 75 cm in length, and 93 cm diagonally, and tilted motors for improved yaw authority. This general hardware setup is standard and similar to many quadrotors.

To study the generalizability and robustness of our approach, we also use an Intel



Aero Ready to Fly drone for data collection. This dataset is used to train a representation of the wind effects on the Intel Aero drone, which we test on our custom UAV. The Intel Aero drone (weighing 1.4 kg) has a symmetric X configuration, 52 cm in width and 52 cm in length, without tilted motors.

**Wind condition design.** To generate dynamic and diverse wind conditions for the data collection and experiments, we leverage the state-of-the-art Caltech Real Weather Wind Tunnel system (Fig. 5.2). The wind tunnel is a 3 m by 3 m array of 1296 independently controllable fans capable of generating wind conditions up to 43.6 km/h. The distributed fans are controlled in real-time by a Python-based Application Programming Interface (API). For data collection and flight experiments, we designed two types of wind conditions. For the first type, each fan has uniform and constant wind speed between 0 km/h and 43.6 km/h (12.1 m/s). The second type of wind follows a sinusoidal function in time, e.g.,  $30.6 + 8.6 \sin(t)$  km/h. Note that the training data only covers constant wind speeds up to 6.1 m/s. To visualize the wind, we use 5 smoke generators to indicate the direction and intensity of the wind condition (see examples in Fig. 5.2 and the video).

### Data Collection and Offline Meta-Training

To learn an effective representation of the aerodynamic effects, we have a custom-built drone follow a randomized trajectory for 2 minutes each in six different static wind conditions, with speeds ranging from 0 km/h to 22.0 km/h. However, in experiments we used wind speeds up to 43.6 km/h (12.1 m/s) to study how our methods extrapolate to unseen wind conditions (e.g., Fig. 5.7). The randomized trajectory follows a polynomial spline between 3 waypoints: the current position and two randomly generated target positions. This process allows us to generate a large amount of data using a trajectory very different from the trajectories used to test our method, such as the figure-8 in Fig. 5.2. By training and testing on different trajectories, we demonstrate that the learned model generalizes well to new trajectories.

Along each trajectory, we collect time-stamped data  $[q, \dot{q}, u]$ . Next, we compute the acceleration  $\ddot{q}$  by fifth-order numerical differentiation and compute the noisy label of  $f$  using Eq. (5.1). The data is collected at 50 Hz with a total of 36,000 data points. Figure 5.4(A) shows the data collection process, and Fig. 5.4(B) shows the inputs and labels of the training data, under one wind condition of 13.3 km/h (3.7 m/s). Figure 5.4(C) shows the distributions of input data (pitch) and

label data ( $x$ -component of the aerodynamic force) in different wind conditions. Clearly, a shift in wind conditions causes distribution shifts in both input domain and label domain, which motivates the algorithm design of DAIML. The same data collection process is repeated on the Intel Aero drone, to study whether the learned representation can generalize to a different drone.

On the collected datasets for both our custom drone and the Intel Aero drone, we apply the DAIML algorithm (Algorithm 5.1) to learn two representations  $\phi$  of the wind effects. The learning process is done offline on a normal desktop computer, and depicted in Fig. 5.3(B). Figure 5.5 shows the evolution of the linear coefficients ( $a^*$ ) during the learning process, where DAIML learns a representation of the aerodynamic effects shared by all wind conditions, and the linear coefficient contains the wind-specific information. Moreover, the learned representation is explainable in the sense that the linear coefficients in different wind conditions are well disentangled (see Fig. 5.5).

### Implementation of Our Control Methods and Baselines

We following the cascading control structure in Chapter 2 to apply our controller and baselines as position controllers for a drone. In particular, the position dynamics of a drone in Eq. (2.4a) can be reduced to Eq. (5.1) by taking  $M = mI$ ,  $C = 0$ ,  $q = p$ ,  $u = Rf_u \in \mathbb{R}^3$  and  $f = f_a \in \mathbb{R}^3$ .

Therefore, we implemented all control methods in the position control loop in Python, and run it on the onboard Linux computer at 50 Hz. The PX4 was set to the offboard flight mode and received thrust and attitude commands from the position control loop. The built-in PX4 multicopter attitude controller was then executed at the default rate. The online inference of the learned representation is also in Python via PyTorch (Paszke et al., 2019).

We implemented three baselines:

- Globally exponentially-stabilizing nonlinear tracking controller for quadrotor control (Morgan et al., 2016; G. Shi, X. Shi, et al., 2019; X. Shi, Kim, et al., 2018). This controller uses an integral term  $\int \tilde{q}$  to compensate  $f$ .
- Incremental nonlinear dynamics inversion (INDI) linear acceleration control (Tal and Karaman, 2021). In particular, we implemented the position and acceleration controller from Sections III.A and III.B in Tal and Karaman (2021).

- $\mathcal{L}_1$  adaptive control (Mallikarjunan et al., 2012; Hanover et al., 2021). We followed the adaptation law first presented in Pravitra et al. (2020), and augment the nonlinear baseline control with  $\hat{f} = -u_{\mathcal{L}_1}$ .

The primary difference between these baseline methods and Neural-Fly is how the controller compensates for the unmodelled residual force (that is, each baseline method has the same control structure, in Fig. 5.3(C), except for the estimation of the  $\hat{f}$ ). The integral gain in the first baseline is limited by the stability of the interaction with the position and velocity error feedback, leading to slow model correction. In contrast, both INDI and  $\mathcal{L}_1$  decouple the adaptation rate from the PD gains, which allow for fast adaptation. However, both INDI and  $\mathcal{L}_1$  directly estimate the unknown vector  $f$  which might change very fast, thus limited by more fundamental design factors, such as system delay, measurement noise, and controller rate.

We also implemented three variants of our methods, depending on the learned representation  $\phi$ :

- Neural-Fly is our primary method using a representation learned from the dataset collected by the custom-built drone, which is the same drone used in experiments.
- Neural-Fly-Transfer uses the Neural-Fly algorithm where the representation is trained using the dataset collected by the aforementioned Intel Aero drone. This variant is included to show the generalizability and robustness of our approach with drone transfer, i.e., using a different drone in experiments than data collection.
- Neural-Fly-Constant uses the online adaptation algorithm from Neural-Fly, but the representation is an artificially designed constant mapping ( $\phi = I$ ). Neural-Fly-Constant demonstrates the benefit of using a better representation learned from the proposed meta-learning method DAIML. Note that Neural-Fly-Constant is a composite adaptation form of a Kalman-filter disturbance observer, that is a Kalman-filter augmented with a tracking error update term.

**Neural network architectures and training details.** In practice, we found that in addition to the drone velocity, the aerodynamic effects also depend on the drone attitude and the rotor rotation speed. To that end, the input state  $x$  to the deep neural network  $\phi$  is a 11-d vector, consisting of the drone velocity (3-d), the drone

attitude represented as a quaternion (4-d), and the rotor speed commands as a pulse width modulation (PWM) signal (4-d) (see Figs. 5.3 and 5.4). The DNN  $\phi$  has four fully-connected hidden layers, with an architecture  $11 \rightarrow 50 \rightarrow 60 \rightarrow 50 \rightarrow 4$  and Rectified Linear Units (ReLU) activation. We found that the three components of the wind-effect force,  $f_x, f_y, f_z$ , are highly correlated and sharing common features, so we use  $\phi$  as the basis function for all the component. Therefore, the wind-effect force  $f$  is approximated by

$$f \approx \begin{bmatrix} \phi(x) & 0 & 0 \\ 0 & \phi(x) & 0 \\ 0 & 0 & \phi(x) \end{bmatrix} \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}, \quad (5.22)$$

where  $a_x, a_y, a_z \in \mathbb{R}^4$  are the linear coefficients for each component of the wind-effect force.

Note that we explicitly include the PWM as an input to the  $\phi$  network. The PWM information is a function of  $u$ , which makes the controller law Eq. (5.7) non-affine in  $u$ . We solve this issue by using the PWM from the last time step as an input to  $\phi$ , to compute the desired force  $u$  at the current time step. Because we train  $\phi$  using spectral normalization (see Algorithm 5.1), this method is stable and guaranteed to converge to a fixed point, as discussed in Chapter 3.

### Trajectory Tracking Performance

We quantitatively compare the performance of the aforementioned control methods when the UAV follows a 2.5 m wide, 1.5 m tall figure-8 trajectory with a lap time of 6.28 s under constant, uniform wind speeds of 0 km/h, 15.1 km/h (4.2 m/s), 30.6 km/h (8.5 m/s), and 43.6 km/h (12.1 m/s) and under time-varying wind speeds of  $30.6 + 8.6 \sin(t)$  km/h ( $8.5 + 2.4 \sin(t)$  m/s).

The flight trajectory for each of the experiments is shown in Fig. 5.6, which includes a warm up lap and six 6.28 s laps. The nonlinear baseline integral term compensates for the mean model error within the first lap. As the wind speed increases, the aerodynamic force variation becomes larger and we notice a substantial performance degradation. INDI and  $\mathcal{L}_1$  both improve over the nonlinear baseline, but INDI is more robust than  $\mathcal{L}_1$  at high wind speeds. Neural-Fly-Constant outperforms INDI except during the two most challenging tasks: 43.6 km/h and sinusoidal wind speeds. The learning based methods, Neural-Fly and Neural-Fly-Transfer, outperform all other methods in all tests. Neural-Fly outperforms Neural-Fly-

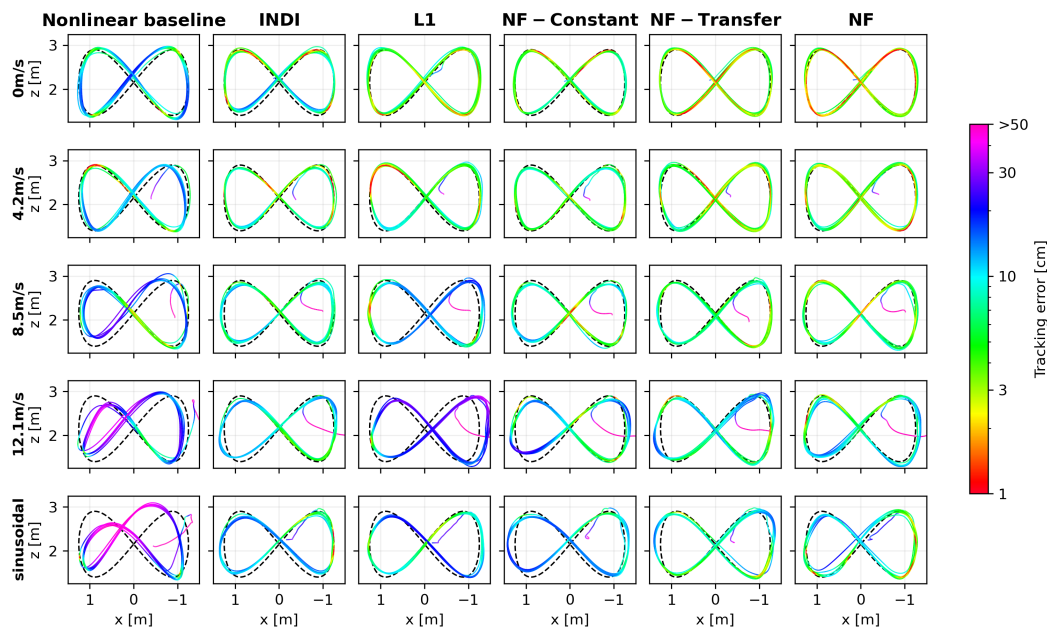


Figure 5.6: Depiction of the trajectory tracking performance of each controller in several wind conditions. The baseline nonlinear controller can track the trajectory well, however, the performance substantially degrades at higher wind speeds. INDI,  $\mathcal{L}_1$ , and Neural-Fly-Constant have similar performance and improve over the nonlinear baseline by estimating the aerodynamic disturbance force quickly. Neural-Fly and Neural-Fly-Transfer use a learned model of the aerodynamic effects and adapt the model in real time to achieve lower tracking error than the other methods.

Table 5.1: Tracking error statistics in cm for different wind conditions. Two metrics are considered: root-mean-square (RMS) and mean.

Method	Wind [m/s]		0		4.2		8.5		12.1		8.5 + 2.4 sin( $t$ )	
	RMS	Mean	RMS	Mean	RMS	Mean	RMS	Mean	RMS	Mean	RMS	Mean
<b>Nonlinear</b>	11.9	10.8	10.7	9.9	16.3	14.7	23.9	21.6	31.2	28.2		
<b>INDI</b>	7.3	6.3	6.4	5.9	8.5	8.2	10.7	10.1	11.1	10.3		
<b>L1</b>	4.6	4.2	5.8	5.2	12.1	11.1	22.7	21.3	13.0	11.6		
<b>NF-Constant</b>	5.4	5.0	6.1	5.7	7.5	6.9	12.7	11.2	12.7	12.1		
<b>NF-Transfer</b>	3.7	3.4	4.8	4.4	6.2	5.9	10.2	9.4	8.8	8.0		
<b>NF</b>	<b>3.2</b>	<b>2.9</b>	<b>4.0</b>	<b>3.7</b>	<b>5.8</b>	<b>5.3</b>	<b>9.4</b>	<b>8.7</b>	<b>7.6</b>	<b>6.9</b>		

Transfer slightly, which is because the learned model was trained on data from the same drone and thus better matches the dynamics of the vehicle.

In Table 5.1, we tabulate the root-mean-square position error and mean position error values over the six laps for each experiment. Figure 5.7 shows how the mean tracking error changes for each controller as the wind speed increases, and includes the standard deviation for the mean lap position error. In all cases, Neural-Fly and Neural-Fly-Transfer outperform the state-of-the-art baseline methods, including the

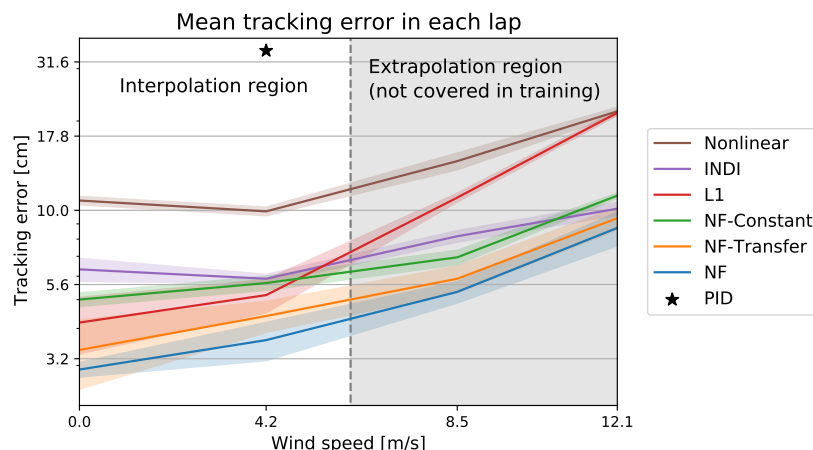


Figure 5.7: Mean tracking errors of each lap in different wind conditions. This figure shows position tracking errors of different methods as wind speed increases. Solid lines show the mean error over 6 laps and the shade areas show standard deviation of the mean error on each lap. The grey area indicates the extrapolation region, where the wind speeds are not covered in training. Our primary method (Neural-Fly) achieves state-of-the-art performance even with a strong wind disturbance.

30.6 km/h, 43.6 km/h, and sinusoidal wind speeds all of which exceed the wind speed in the training data. All of these results presents a clear trend: adaptive control substantially outperforms the nonlinear baseline which relies on integral-control, and learning markedly improves adaptive control.

### Agile Flight Through Narrow Gates

Precise flight control in dynamic and strong wind conditions has many applications, such as rescue and search, delivery, and transportation. In this section, we present a challenging drone flight task in strong winds, where the drone must follow agile trajectories through narrow gates, which are only slightly wider than the drone. The overall result is depicted in Fig. 5.2 and the video. As shown in Fig. 5.2(A), the gates used in our experiments are 110 cm in width, which is only slightly wider than the drone (85 cm wide, 75 cm long). To visualize the trajectory using long-exposure photography, our drone is deployed with four main light emitting diodes (LEDs) on its legs, where the two rear LEDs are red and the front two are white. There are also several small LEDs on the flight controller, the computer, and the motor controllers, which can be seen in the long-exposure shots.

**Task design.** We tested our method on three different tasks. In the first task (see Fig. 5.2(B,D,F-I) and the video), the desired trajectory is a 3 m by 1.5 m figure-8 in the  $x - z$  plane with a lap time of 5 s. A gate is placed at the left bottom part

of the trajectory. The minimum clearance is about 10 cm (see Fig. 5.2(I), which requires that the controller precisely tracks the trajectory. The maximum speed and acceleration of the desired trajectory are 2.7 m/s and  $5.0 \text{ m/s}^2$ , respectively. The wind speed is 3.1 m/s. The second task (see Video 1) is the same as the first one, except that it uses a more challenging, time-varying wind condition,  $3.1 + 1.8 \sin(\frac{2\pi}{5}t) \text{ m/s}$ . In the third task (see Fig. 5.2(C,E) and the video), the desired trajectory is a 3 m by 2.5 m ellipse in the  $x - y$  plane with a lap time of 5 s. We placed two gates on the left and right sides of the ellipse. As with the first task, the wind speed is 3.1 m/s.

**Performance.** For all three tasks, we used our primary method, Neural-Fly, where the representation is learned using the dataset collected by the custom-built drone. Figure 5.2(D,E) are two long-exposure photos with an exposure time of 5 s, which is the same as the lap time of the desired trajectory. We see that our method precisely tracked the desired trajectories and flew safely through the gates (see the video). These long-exposure photos also captured the smoke visualization of the wind condition. We would like to emphasize that the drone is wider than the LED light region, since the LEDs are located on the legs (see Fig. 5.2(A)). Figure 5.2(F-I) are four high-speed photos with a shutter speed of  $1/200\text{s}$ . These four photos captured the moment the drone passed through the gate in the first task, as well as the complex interaction between the drone and the wind. We see that the aerodynamic effects are complex and non-stationary and depend on the UAV attitude, the relative velocity, and aerodynamic interactions between the propellers and the wind.

## Outdoor Experiments

We tested our algorithm outdoors in gentle breeze conditions (wind speeds measured up to 17 km/h). An onboard GPS receiver provided position information to the EKF, giving lower precision state estimation, and therefore less precise aerodynamic residual force estimation. Following the same aforementioned figure-8 trajectory, the controller reached 7.5 cm mean tracking error, shown in Fig. 5.8.

## Discussions

**State-of-the-art tracking performance.** When measuring position tracking errors, we observe that our Neural-Fly method outperforms state-of-the-art flight controllers in all wind conditions. Furthermore, we observe a mean tracking error of 2.9 cm in 0 km/h wind, which is comparable with state-of-the-art tracking performance demonstrated on more aggressive racing drones (Tal and Karaman, 2021; Hanover

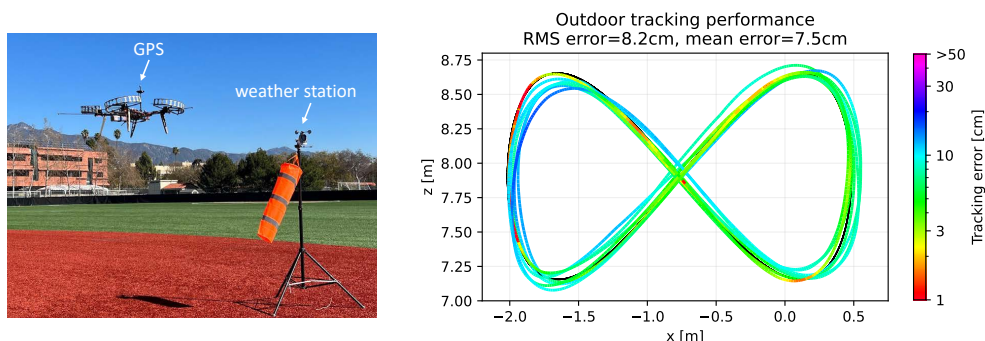


Figure 5.8: Outdoor flight setup and performance. Left: In outdoor experiments, a GPS module is deployed for state estimation, and a weather station records wind profiles. The maximum wind speed during the test was around 17 km/h (4.9 m/s). Right: Trajectory tracking performance of Neural-Fly.

et al., 2021) despite several architectural limitations such as limited control rate in offboard mode, a larger, less maneuverable vehicle, and without direct motor speed measurements. All our experiments were conducted using the standard PX4 attitude controller, with Neural-Fly implemented in an onboard, low cost, and “credit-card sized” Raspberry Pi 4 computer. Furthermore, Neural-Fly is robust to changes in vehicle configuration, as demonstrated by the similar performance of Neural-Fly-Transfer.

To understand the fundamental tracking-error limit, we estimate that the localization precision from the OptiTrack system is about 1 cm, which is a practical lower bound for the average tracking error in our system. This is based on the fact that the difference between the OptiTrack position measurement and the onboard EKF position estimate is around 1 cm.

To achieve a tracking error of 1 cm, remaining improvements should focus on reducing code execution time, communication delays, and attitude tracking delay. We measured the combined code execution time and communication delay to be at least 15 ms and often as much as 30 ms. A faster implementation (such as using C++ instead of Python) and streamlined communication layer (such as using ROS2’s real-time features) could allow us to achieve tracking errors on the order of the localization accuracy. Attitude tracking delay can be substantially reduced through the use of a nonlinear attitude controller (e.g., X. Shi, Kim, et al. (2018)).



Our method is also directly extensible to attitude control because attitude dynamics match the Euler-Lagrange dynamics used in our derivations. However, further work is needed to understand the interaction of the learned dynamics with the cascaded control design when implementing a tracking attitude controller.

**Challenges caused by unknown and time-varying wind conditions.** In the real world, the wind conditions are not only unknown but also constantly changing, and the vehicle must continuously adapt. We designed the sinusoidal wind test to emulate unsteady or gusty wind conditions. Although our learned model is trained on static and approximately uniform wind condition data, Neural-Fly can quickly identify changing wind speed and maintains precise tracking even on the sinusoidal wind experiment. Moreover, in each of our experiments, the wind conditions were unknown to the UAV before starting the test yet were quickly identified by the adaptation algorithm.

Our work demonstrated that it is possible to repeatably and quantitatively test quadrotor flight in time-varying wind. Our method separately learns the wind effect's dependence on the vehicle state (i.e., the wind-invariant representation in Fig. 5.3(A)) and the wind condition (i.e., the wind-specific linear weight in Fig. 5.3(A)). This separation allows Neural-Fly to quickly adapt to the time-varying wind even as the UAV follows a dynamic trajectory, with an average tracking error below 8.7 cm in Table 5.1.

**Computational efficiency of our method.** In the offline meta-learning phase, the proposed DAIML algorithm is able to learn an effective representation of the aerodynamic effect in a data efficient manner. This requires only 12 minutes of flight data at 50 Hz, for a total of 36,000 data points. The training procedure only takes 5 minutes on a normal desktop computer. In the online adaptation phase, our adaptive control method only takes 10 ms to compute on a compact onboard Linux computer (Raspberry Pi 4). In particular, the feedforward inference time via the learned basis function is about 3.5 ms and the adaptation update is about 3.0 ms, which implies the compactness of the learned representation.

**Generalization to new trajectories and new aircrafts.** It is worth noting that our control method is orthogonal to the design of the desired trajectory  $q_d$ . In this chapter, we focus on the figure-8 trajectory which is a commonly used control benchmark. We also demonstrate our method flying a horizontal ellipse during the narrow gate demonstration Fig. 5.2. Note that our method supports any trajectory planners such as Foehn et al. (2021) or learning-based planners (Nakka et al., 2021;

Loquercio et al., 2021). In particular, for those planners which require a precise and agile downstream controller (e.g., for close-proximity flight or drone racing (Foehn et al., 2021; G. Shi, Hönig, et al., 2022)), our method immediately provides a solution and further pushes the boundary of these planners, because our state-of-the-art tracking capabilities enable tighter configurations and smaller clearances. However, further research is required to understand the coupling between planning and learning-based control near actuation limits. Future work will consider using Neural-Fly in a combined planning and control structure such as MPC, which will be able to handle actuation limits.

The comparison between Neural-Fly and Neural-Fly-Transfer show that our approach is robust to changing vehicle design and the learned representation does not depend on the vehicle. This demonstrates the generalizability of the proposed method running on different quadrotors. Moreover, our control algorithm is formulated generally for all robotic systems described by the Euler-Langrange equation Eq. (5.1), including many types of aircraft such as X. Shi, Spieler, et al. (2020) and Kim et al. (2021).

## 5.7 Appendix

### The Expressiveness of the Learning Architecture

In this section, we theoretically justify the decomposition  $f(x, w) \approx \phi(x)a(w)$ . In particular, we prove that any analytic function  $\bar{f}(x, w) : [-1, 1]^n \times [-1, 1]^m \rightarrow \mathbb{R}$  can be split into a  $w$ -invariant part  $\bar{\phi}(x)$  and a  $w$ -dependant part  $\bar{a}(w)$  in the structure  $\bar{\phi}(x)\bar{a}(w)$  with arbitrary precision  $\epsilon$ , where  $\bar{\phi}(x)$  and  $\bar{a}(w)$  are two polynomials. Further, the dimension of  $\bar{a}(w)$  only scales polylogarithmically with  $1/\epsilon$ .

We first introduce the following multivariate polynomial approximation lemma in the hypercube proved in Trefethen (2017).

**Lemma 5.1.** *(Multivariate polynomial approximation in the hypercube) Let  $\bar{f}(x, w) : [-1, 1]^n \times [-1, 1]^m \rightarrow \mathbb{R}$  be a smooth function of  $[x, w] \in [-1, 1]^{n+m}$  for  $n, m \geq 1$ . Assume  $\bar{f}(x, w)$  is analytic for all  $[x, w] \in \mathbb{C}^{n+m}$  with  $\Re(x_1^2 + \dots + x_n^2 + w_1^2 + \dots + w_m^2) \geq -t^2$  for some  $t > 0$ , where  $\Re(\cdot)$  denotes the real part of a complex number. Then  $\bar{f}$  has a uniformly and absolutely convergent multivariate Chebyshev series*

$$\sum_{k_1=0}^{\infty} \dots \sum_{k_n=0}^{\infty} \sum_{l_1=0}^{\infty} \dots \sum_{l_m=0}^{\infty} b_{k_1, \dots, k_n, l_1, \dots, l_m} T_{k_1}(x_1) \dots T_{k_n}(x_n) T_{l_1}(w_1) \dots T_{l_m}(w_m).$$

Define  $s = [k_1, \dots, k_n, l_1, \dots, l_m]$ . The multivariate Chebyshev coefficients satisfy

the following exponential decay property:

$$b_s = O\left((1+t)^{-\|s\|_2}\right).$$

Note that this lemma shows that the truncated Chebyshev expansions

$$C_p = \sum_{k_1=0}^p \cdots \sum_{k_n=0}^p \sum_{l_1=0}^p \cdots \sum_{l_m=0}^p b_{k_1, \dots, k_n, l_1, \dots, l_m} T_{k_1}(x_1) \cdots T_{k_n}(x_n) T_{l_1}(w_1) \cdots T_{l_m}(w_m)$$

will converge to  $\bar{f}$  with the rate  $O((1+t)^{-p\sqrt{n+m}})$  for some  $t > 0$ , i.e.,

$$\sup_{[x,w] \in [-1,1]^{n+m}} \|\bar{f}(x,w) - C_p(x,w)\| \leq O((1+t)^{-p\sqrt{n+m}}).$$

Finally we are ready to present the following representation theorem.

**Theorem 5.2.**  $\bar{f}(x,w)$  is a function satisfying the assumptions in Lemma 5.1. For any  $\epsilon > 0$ , there exist  $h \in \mathbb{Z}^+$ , and two Chebyshev polynomials  $\bar{\phi}(x) : [-1, 1]^n \rightarrow \mathbb{R}^{1 \times h}$  and  $\bar{a}(w) : [-1, 1]^m \rightarrow \mathbb{R}^{h \times 1}$  such that

$$\sup_{[x,w] \in [-1,1]^{n+m}} \|\bar{f}(x,w) - \bar{\phi}(x)\bar{a}(w)\| \leq \epsilon$$

and  $h = O((\log(1/\epsilon))^m)$ .

*Proof.* First note that there exists  $p = O\left(\frac{\log(1/\epsilon)}{\sqrt{n+m}}\right)$  such that

$$\sup_{[x,w] \in [-1,1]^{n+m}} \|\bar{f}(x,w) - C_p(x,w)\| \leq \epsilon.$$

To simplify the notation, define

$$\begin{aligned} g(x, k, l) &= g(x_1, \dots, x_n, k_1, \dots, k_n, l_1, \dots, l_m) = b_{k_1, \dots, k_n, l_1, \dots, l_m} T_{k_1}(x_1) \cdots T_{k_n}(x_n) \\ &g(w, l) = g(w_1, \dots, w_m, l_1, \dots, l_m) = T_{l_1}(w_1) \cdots T_{l_m}(w_m). \end{aligned}$$

Then we have

$$C_p(x,w) = \sum_{k_1, \dots, k_n=0}^p \sum_{l_1, \dots, l_m=0}^p g(x, k_1, \dots, k_n, l_1, \dots, l_m) g(w, l_1, \dots, l_m).$$

Then we rewrite  $C_p$  as  $C_p(x, w) = \bar{\phi}(x)\bar{a}(w)$ :

$$\bar{\phi}(x)^\top = \begin{bmatrix} \sum_{k_1, \dots, k_n=0}^p g(x, k_1, \dots, k_n, l = [0, 0, \dots, 0]) \\ \sum_{k_1, \dots, k_n=0}^p g(x, k_1, \dots, k_n, l = [1, 0, \dots, 0]) \\ \sum_{k_1, \dots, k_n=0}^p g(x, k_1, \dots, k_n, l = [2, 0, \dots, 0]) \\ \vdots \\ \sum_{k_1, \dots, k_n=0}^p g(x, k_1, \dots, k_n, l = [p, p, \dots, p]) \end{bmatrix},$$

$$\bar{a}(w) = \begin{bmatrix} g(w, l = [0, 0, \dots, 0]) \\ g(w, l = [1, 0, \dots, 0]) \\ g(w, l = [2, 0, \dots, 0]) \\ \vdots \\ g(w, l = [p, p, \dots, p]) \end{bmatrix}.$$

Note that the dimension of  $\bar{\phi}(x)$  and  $\bar{a}(w)$  is

$$h = (p + 1)^m = O\left(\left(1 + \frac{\log(1/\epsilon)}{\sqrt{n+m}}\right)^m\right) = O((\log(1/\epsilon))^m).$$

□

Note that Theorem 5.2 can be generalized to vector-valued functions with bounded input space straightforwardly. Finally, since deep neural networks are universal approximators for polynomials (Yarotsky, 2017), Theorem 5.2 immediately guarantees the expressiveness of our learning structure, i.e.,  $\phi(x)a(w)$  can approximate  $f(x, w)$  with arbitrary precision, where  $\phi(x)$  is a deep neural network and  $\hat{a}$  includes the linear coefficients for all the elements of  $f$ . In experiments, we show that a four-layer neural network can efficiently learn an effective representation for the underlying unknown dynamics  $f(x, w)$ .

## References

- Bartlett, Peter L., Dylan J. Foster, and Matus J. Telgarsky (2017). *Spectrally-normalized margin bounds for neural networks*. In: *Advances in Neural Information Processing Systems*, pp. 6240–6249.
- Belkhale, Suneel, Rachel Li, Gregory Kahn, Rowan McAllister, Roberto Calandra, and Sergey Levine (2021). *Model-based meta-reinforcement learning for flight with suspended payloads*. In: *IEEE Robotics and Automation Letters* 6.2, pp. 1471–1478.
- Bisheban, Mahdis and Taeyoung Lee (July 2021). *Geometric adaptive control with neural networks for a quadrotor in wind fields*. In: *IEEE Transactions on Control Systems Technology* 29.4, pp. 1533–1548. ISSN: 1558-0865. DOI: 10.1109/TCST.2020.3006184.

- Chen, Fu-Chuang and Hassan K. Khalil (1995). *Adaptive control of a class of nonlinear discrete-time systems using neural networks*. In: *IEEE Transactions on Automatic Control* 40.5, pp. 791–801.
- Clavera, Ignasi, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel (2018). *Model-based reinforcement learning via meta-policy optimization*. In: *Conference on Robot Learning*. Proceedings of Machine Learning Research, pp. 617–629.
- Dieci, Luca and Timo Eirola (Apr. 1994). *Positive definiteness in the numerical solution of Riccati differential equations*. In: *Numerische Mathematik* 67.3, pp. 303–313. ISSN: 0945-3245. DOI: 10.1007/s002110050030. URL: <https://doi.org/10.1007/s002110050030>.
- Faessler, Matthias, Antonio Franchi, and Davide Scaramuzza (Apr. 2018). *Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories*. In: *IEEE Robotics and Automation Letters* 3.2, pp. 620–626. ISSN: 2377-3766. DOI: 10.1109/LRA.2017.2776353.
- Farrell, Jay A. and Marios M. Polycarpou (2006). *Adaptive approximation based control*. John Wiley & Sons, Ltd. ISBN: 978-0-471-78181-3. DOI: 10.1002/0471781819.fmatter. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/0471781819.fmatter>.
- Finn, Chelsea, Pieter Abbeel, and Sergey Levine (2017). *Model-agnostic meta-learning for fast adaptation of deep networks*. In: *International Conference on Machine Learning*. Proceedings of Machine Learning Research, pp. 1126–1135.
- Foehn, Philipp, Angel Romero, and Davide Scaramuzza (2021). *Time-optimal planning for quadrotor waypoint flight*. In: *Science Robotics* 6.56.
- Ganin, Yaroslav, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky (2016). *Domain-adversarial training of neural networks*. In: *The Journal of Machine Learning Research* 17.1, pp. 2096–2030.
- Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio (2014). *Generative adversarial nets*. In: *Advances in Neural Information Processing Systems* 27.
- Hanover, Drew, Philipp Foehn, Sihao Sun, Elia Kaufmann, and Davide Scaramuzza (2021). *Performance, precision, and payloads: Adaptive nonlinear MPC for quadrotors*. In: *IEEE Robotics and Automation Letters* 7.2, pp. 690–697.
- Hospedales, Timothy M., Antreas Antoniou, Paul Micaelli, and Amos J. Storkey (2021). *Meta-learning in neural networks: A survey*. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1. DOI: 10.1109/TPAMI.2021.3079209.
- Ioannou, Petros A. and Jing Sun (1996). *Robust adaptive control*. Vol. 1. Prentice-Hall Upper Saddle River, NJ.

- Johnson, Eric N. and Anthony J. Calise (2003). *Limited authority adaptive flight control for reusable launch vehicles*. In: *Journal of Guidance, Control, and Dynamics* 26.6, pp. 906–913.
- Kalman, R. E. and R. S. Bucy (Mar. 1961). *New results in linear filtering and prediction theory*. In: *Journal of Basic Engineering* 83.1, pp. 95–108. ISSN: 0021-9223. DOI: 10.1115/1.3658902. URL: <https://asmedigitalcollection.asme.org/fluidsengineering/article/83/1/95/426820/New-Results-in-Linear-Filtering-and-Prediction>.
- Kim, Kyunam, Patrick Spieler, Elena-Sorina Lupu, Alireza Ramezani, and Soon-Jo Chung (2021). *A bipedal walking robot that can fly, slackline, and skateboard*. In: *Science Robotics* 6.59, eabf8136. DOI: 10.1126/scirobotics.abf8136.
- Krstic, Miroslav, Petar V. Kokotovic, and Ioannis Kanellakopoulos (1995). *Nonlinear and adaptive control design*. John Wiley & Sons, Inc.
- Lale, Sahin, Kamyar Azizzadenesheli, Babak Hassibi, and Anima Anandkumar (Dec. 2021). *Model learning predictive control in nonlinear dynamical systems*. In: *2021 60th IEEE Conference on Decision and Control (CDC)*. ISSN: 2576-2370, pp. 757–762. DOI: 10.1109/CDC45484.2021.9683670.
- LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). *Deep learning*. In: *Nature* 521.7553, pp. 436–444.
- Lee, Joonho, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter (2020). *Learning quadrupedal locomotion over challenging terrain*. In: *Science Robotics* 5.47.
- Loquercio, Antonio, Elia Kaufmann, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza (2021). *Learning high-speed flight in the wild*. In: *Science Robotics* 6.59, eabg5810. DOI: 10.1126/scirobotics.abg5810. URL: <https://www.science.org/doi/abs/10.1126/scirobotics.abg5810>.
- Mallikarjunan, Srinath, Bill Nesbitt, Evgeny Kharisov, Enric Xargay, Naira Hovakimyan, and Chengyu Cao (Aug. 2012). *LI adaptive controller for attitude control of multirotors*. In: *AIAA Guidance, Navigation, and Control Conference*. Minneapolis, Minnesota: American Institute of Aeronautics and Astronautics. ISBN: 978-1-60086-938-9. DOI: 10.2514/6.2012-4831. URL: <https://arc.aiaa.org/doi/10.2514/6.2012-4831>.
- McKinnon, Christopher D. and Angela P. Schoellig (2021). *Meta learning with paired forward and inverse models for efficient receding horizon control*. In: *IEEE Robotics and Automation Letters* 6.2, pp. 3240–3247.
- Meier, Lorenz, Petri Tanskanen, Lionel Heng, Gim Hee Lee, Friedrich Fraundorfer, and Marc Pollefeys (Aug. 2012). *PIXHAWK -A micro aerial vehicle design for autonomous flight using onboard computer vision*. In: *Autonomous Robots* 33.1-2, pp. 21–39. ISSN: 0929-5593. DOI: 10.1007/s10514-012-9281-4. URL: <https://graz.pure.elsevier.com/en/publications/pixhawk-a-micro-aerial-vehicle-design-for-autonomous-flight-using>.

- Morgan, Daniel, Giri P. Subramanian, Soon-Jo Chung, and Fred Y. Hadaegh (2016). *Swarm assignment and trajectory optimization using variable-swarm, distributed auction assignment and sequential convex programming*. In: *The International Journal of Robotics Research* 35.10, pp. 1261–1285.
- Nagabandi, Anusha, Ignasi Clavera, Simin Liu, Ronald S. Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn (2018). *Learning to adapt in dynamic, real-world environments through meta-reinforcement learning*. In: *arXiv preprint arXiv:1803.11347*.
- Nakanishi, J., J.A. Farrell, and S. Schaal (Sept. 2002). *A locally weighted learning composite adaptive controller with structure adaptation*. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. Vol. 1, 882–889 vol.1. DOI: 10.1109/IRDS.2002.1041502.
- Nakka, Yashwanth Kumar, Anqi Liu, Guanya Shi, Anima Anandkumar, Yisong Yue, and Soon-Jo Chung (2021). *Chance-constrained trajectory optimization for safe exploration and learning of nonlinear Systems*. In: *IEEE Robotics and Automation Letters* 6.2, pp. 389–396. DOI: 10.1109/LRA.2020.3044033.
- Narendra, Kumpati S. and Anuradha M. Annaswamy (2012). *Stable adaptive systems*. Courier Corporation.
- Narendra, Kumpati S. and Snehasis Mukhopadhyay (1997). *Adaptive control using neural networks and approximate models*. In: *IEEE Transactions on Neural Networks* 8.3, pp. 475–485.
- O’Connell, Michael, Guanya Shi, Xichen Shi, and Soon-Jo Chung (2021). *Meta-learning-based robust adaptive flight control under uncertain wind conditions*. In: *arXiv preprint arXiv:2103.01932*. URL: <https://arxiv.org/abs/2103.01932>.
- Paszke, Adam et al. (2019). *PyTorch: An imperative style, high-performance deep learning library*. In: *Advances in Neural Information Processing Systems*, pp. 8024–8035. URL: <http://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library>.
- Peng, Matt, Banghua Zhu, and Jiantao Jiao (2021). *Linear representation meta-reinforcement learning for instant adaptation*. In: *arXiv preprint arXiv:2101.04750*.
- Pravitra, Jintasit, Kasey A. Ackerman, Chengyu Cao, Naira Hovakimyan, and Evangelos A. Theodorou (Oct. 2020). *L1-adaptive MPPI architecture for robust and agile control of multirotors*. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. ISSN: 2153-0866, pp. 7661–7666. DOI: 10.1109/IROS45743.2020.9341154.
- Rahimi, Ali and Benjamin Recht (2007). *Random features for large-scale kernel machines*. In: *Proceedings of the 20th International Conference on Neural Information Processing Systems*, pp. 1177–1184.

- Ramos, Fabio, Rafael Carvalhaes Possas, and Dieter Fox (2019). *Bayessim: Adaptive domain randomization via probabilistic inference for robotics simulators*. In: *arXiv preprint arXiv:1906.01728*.
- Richards, Spencer M., Navid Azizan, Jean-Jacques E. Slotine, and Marco Pavone (2021). *Adaptive-control-oriented meta-learning for nonlinear systems*. In: *arXiv preprint arXiv:2103.04490*.
- Shi, Guanya, Wolfgang Hönig, Xichen Shi, Yisong Yue, and Soon-Jo Chung (2022). *Neural-Swarm2: Planning and control of heterogeneous multirotor swarms using learned interactions*. In: *IEEE Transactions on Robotics* 38.2, pp. 1063–1079. DOI: 10.1109/TRO.2021.3098436.
- Shi, Guanya, Xichen Shi, Michael O’Connell, Rose Yu, Kamyar Azizzadenesheli, Animashree Anandkumar, Yisong Yue, and Soon-Jo Chung (2019). *Neural Lander: Stable drone landing control using learned dynamics*. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 9784–9790. DOI: 10.1109/ICRA.2019.8794351.
- Shi, Xichen, Kyunam Kim, Salar Rahili, and Soon-Jo Chung (2018). *Nonlinear control of autonomous flying cars with wings and distributed electric propulsion*. In: *2018 IEEE Conference on Decision and Control (CDC)*. IEEE, pp. 5326–5333.
- Shi, Xichen, Patrick Spieler, Ellande Tang, Elena-Sorina Lupu, Phillip Tokumar, and Soon-Jo Chung (May 2020). *Adaptive nonlinear control of fixed-wing VTOL with airflow vector sensing*. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. Paris, France: IEEE, pp. 5321–5327. ISBN: 978-1-72817-395-5. DOI: 10.1109/ICRA40945.2020.9197344. URL: <https://ieeexplore.ieee.org/document/9197344/>.
- Slotine, Jean-Jacques E. and Weiping Li (1991). *Applied nonlinear control*. Vol. 199. 1. Prentice Hall, Englewood Cliffs, NJ.
- Song, Xingyou, Yuxiang Yang, Krzysztof Choromanski, Ken Caluwaerts, Wenbo Gao, Chelsea Finn, and Jie Tan (2020). *Rapidly adaptable legged robots via evolutionary meta-learning*. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 3769–3776.
- Tal, Ezra and Sertac Karaman (May 2021). *Accurate tracking of aggressive quadrotor trajectories using incremental nonlinear dynamic inversion and differential flatness*. In: *IEEE Transactions on Control Systems Technology* 29.3, pp. 1203–1218. ISSN: 1558-0865. DOI: 10.1109/TCST.2020.3001117.
- Tobin, Josh, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel (2017). *Domain randomization for transferring deep neural networks from simulation to the real world*. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 23–30.



- Torrente, Guillem, Elia Kaufmann, Philipp Föhn, and Davide Scaramuzza (2021). *Data-driven MPC for quadrotors*. In: *IEEE Robotics and Automation Letters* 6.2, pp. 3769–3776.
- Trefethen, Lloyd (2017). *Multivariate polynomial approximation in the hypercube*. In: *Proceedings of the American Mathematical Society* 145.11, pp. 4837–4844.
- Ventura Diaz, Patricia and Steven Yoon (2018). *High-fidelity computational aerodynamics of multi-rotor unmanned aerial vehicles*. In: *2018 AIAA Aerospace Sciences Meeting*, p. 1266.
- Wise, Kevin A., Eugene Lavretsky, and Naira Hovakimyan (2006). *Adaptive control of flight: Theory, applications, and open problems*. In: *2006 American Control Conference*.
- Yarotsky, Dmitry (2017). *Error bounds for approximations with deep ReLU networks*. In: *Neural Networks* 94, pp. 103–114.

## Chapter 6

## SAFE EXPLORATION

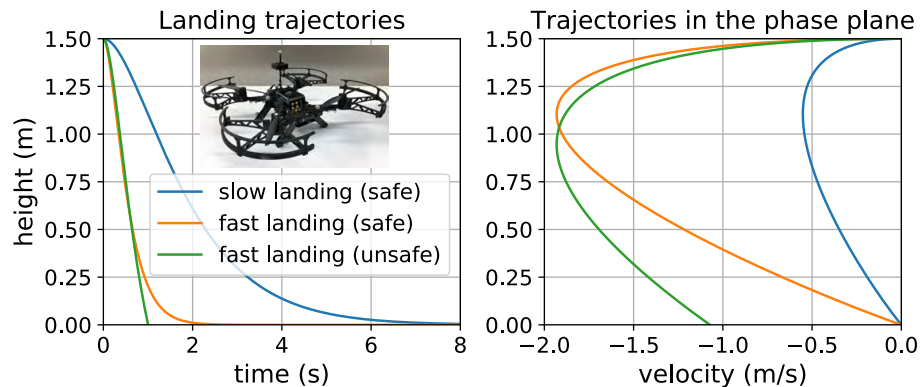


Figure 6.1: Motivating example: drone landing trajectories with different speeds.

In Chapter 2, we introduced a general mixed robot dynamics model  $M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = Bu + f$  where  $f$  is the unknown dynamics. In Chapters 3 to 5, we discussed different settings for  $f$ :  $f$  depends on  $u$ ,  $f$  considers multi-agent interactions, and  $f$  depends on time-variant environmental conditions. However, all these settings rely on *pre-collected data* to train or pre-train deep learning models. In general, data collection for learning robotic agility needs either human supervision or fine-tuned programs in well-controlled environments. For example, in *Neural-Lander* (Chapter 3) an expert human pilot flew the drone for 5 min to collect data where the drone was flying aggressively and very close to the ground.

In this chapter, we will discuss several key principles to safe learn an agile controller under uncertainty *from scratch*, i.e., without any pre-collected data. Figure 6.1 depicts an example, where the drone must start from extremely slow but safe landing, explore and learn new skills itself, and eventually achieve fast and safe landing. In this process, we must avoid catastrophic failure such as the fast and unsafe landing curve in Fig. 6.1. This chapter is mainly based on the following papers:

Liu, Anqi, Guanya Shi, Soon-Jo Chung, Anima Anandkumar, and Yisong Yue (2020). *Robust regression for safe exploration in control*. In: *Learning for Dynamics and Control*. PMLR, pp. 608–619. URL: <https://proceedings.mlr.press/v120/liu20a.html>.

Nakka, Yashwanth Kumar, Anqi Liu, Guanya Shi, Anima Anandkumar, Yisong Yue, and Soon-Jo Chung (2021). *Chance-constrained trajectory optimization for safe exploration and learning of nonlinear Systems*. In: *IEEE Robotics and Automation Letters* 6.2, pp. 389–396. doi: 10.1109/LRA.2020.3044033.

---

**Abstract.** We study the problem of safe learning and exploration in sequential control problems. The goal is to safely collect data samples from operating in an environment, in order to learn to achieve a challenging control goal (e.g., an agile maneuver close to a boundary). A central challenge in this setting is how to quantify uncertainty in order to choose provably-safe actions that allow us to collect informative data and reduce uncertainty, thereby achieving both improved controller safety and optimality. To address this challenge, we present a deep robust regression model that is trained to directly predict the uncertainty bounds for safe exploration. We derive generalization bounds for learning, and connect them with safety and stability bounds in control. We demonstrate empirically that our robust regression approach can outperform conventional Gaussian process (GP) based safe exploration in settings where it is difficult to specify a good GP prior.

## 6.1 Introduction and Related Work

A key challenge in data-driven design for robotic controllers (e.g., Chapters 3 to 5) is automatically and safely collecting training data. Consider safely landing a drone at fast landing speeds (beyond a human expert’s piloting abilities). The dynamics are both highly non-linear and poorly modeled as the drone approaches the ground, but such dynamics can be learnable given the appropriate training data (see Chapter 3). To collect such data autonomously, one must guarantee safety while operating in the environment, which is the problem of *safe exploration*. In the drone landing example, collecting informative training data requires the drone to land increasingly faster while not crashing. Figure 6.1 depicts an example, where the goal is to learn the most aggressive yet safe trajectory (orange), while not being overconfident and execute trajectories that crash (green); the initial nominal controller may only be able to execute very conservative trajectories (blue).

In order to safely collect such informative training data, we need to overcome two difficulties. First, we must quantify the learning errors in out-of-sample data. Every step of data collection creates a shift in the training data distribution. More specifically, our setting is an instance of covariate shift, where the underlying true

physics stay constant, but the sampling of the state space is biased by the data collection (Chen et al., 2016). In order to leverage modern learning approaches, such as deep learning, we must reason about the impact of covariate shift when predicting on states not well represented by the training set. Second, we must reason about how to guarantee safety and stability when controlling using the current learned model. Our ultimate goal is to control the dynamical system with desired properties but staying safe and stable while data collection. The imperfect dynamical model’s error translate to possible control error, which must be quantified and controlled.

In this chapter, we propose a deep robust regression approach for safe exploration in model-based control. We view exploration as a data shift problem, i.e., the “test” data in the proposed exploratory trajectory comes from a shifted distribution compared to the training set. Our approach explicitly learns to quantify uncertainty under such covariate shift, which we use to learn robust dynamics models to quantify uncertainty of entire trajectories for safe exploration.

We analyze learning performance from both generalization and data perturbation perspectives. We use our robust regression analysis to derive stability bounds for control performance when learning robust dynamics models, which is used for safe exploration. We empirically show that our approach outperforms conventional safe exploration approaches with much less tuning effort in two scenarios: (a) inverted pendulum trajectory tracking under wind disturbance; and (b) fast drone landing using an aerodynamics simulation based on real-world flight data from Chapter 3.

### **Related Work in Safe Exploration**

Most approaches for safe exploration use Gaussian processes (GPs) to quantify uncertainty (Sui, Gotovos, et al., 2015; Sui, Zhuang, et al., 2018; Kirschner et al., 2019; Akametalu et al., 2014; Berkenkamp, Schoellig, et al., 2016; Turchetta et al., 2016; Wachi et al., 2018; Berkenkamp, Turchetta, et al., 2017; Fisac et al., 2018). These methods are related to bandit algorithms (Bubeck, Cesa-Bianchi, et al., 2012) and typically employ upper confidence bounds (Auer, 2002) to balance exploration versus exploitation (Srinivas et al., 2010). However, GPs are sensitive to model (i.e., the kernel) selection, and thus are often not suitable for tasks that aim to gradually reach boundaries of safety sets in a highly non-linear environment. In the high-dimensional case and under finite information, GPs suffer from bad priors even more severely (Owhadi et al., 2015).

One could blend GP-based modeling with general function approximations (such as

deep learning) (Berkenkamp, Turchetta, et al., 2017; Cheng, Orosz, et al., 2019), but the resulting optimization-based control problem can be challenging to solve. Other approaches either require having a safety model pre-specified upfront (Alshiekh et al., 2018), are restricted to relatively simple models (Moldovan and Abbeel, 2012), have no convergence guarantees during learning (Taylor et al., 2019), or have no safety guarantees (Garcia and Fernández, 2012).

### **Related Work in Distribution Shift**

The study of learning under distribution shift has seen increasing interest, owing to the widespread practical issue of distribution mismatch. Our work frames uncertainty quantification through the lens of covariate shift (Liu and Ziebart, 2014) with rigorous guarantees. Dealing with domain shift is a fundamental challenge in deep learning, as highlighted by their vulnerability to adversarial inputs (Goodfellow et al., 2014), and the implied lack of robustness. Beyond robust estimation, the typical approaches are to either regularize (Srivastava et al., 2014; Wager et al., 2013; Le et al., 2016; Bartlett et al., 2017; Miyato et al., 2018; Benjamin et al., 2019; Cheng, Verma, et al., 2019) or synthesize an augmented dataset that anticipates the domain shift (Prest et al., 2012; Zheng et al., 2016; Stewart and Ermon, 2017). We also utilize spectral normalization (see Chapter 2) in conjunction with robust estimation.

### **Related Work in Robust and Adaptive Control**

Robust control (Zhou and Doyle, 1998) and adaptive control (see the related work section in Chapter 5) are two classical frameworks to handle uncertainties in the dynamics. GPs have been combined with nonlinear MPC for online adaptation and uncertainty estimation (Ostafew et al., 2016). However, robust control suffers from large uncertainty set, and it is hard to jointly analyse learning and control convergence under the framework of model reference adaptive control. For example, in Chapter 5 we need to assume the representation error from the learned model is bounded by  $d$ , to be able to show exponential convergence to a bounded error ball. However, how to quantify  $d$  remains unclear. Ours is the first to explicitly consider covariate shift in learning dynamics. We pick the region to estimate uncertainty carefully and adapt the controller to track safe proposed trajectory in data collection.

## **6.2 Problem Statement**

To simplify the notation, we define the robot state as  $x = [q; \dot{q}]$  and rewrite the robotic dynamics model in Chapter 2 as

$$\dot{x} = \underbrace{f_0(x, u)}_{\text{nominal dynamics}} + \underbrace{f(x, u)}_{\text{unknown dynamics}} + \underbrace{\epsilon(t)}_{\text{bounded noise}} \quad (6.1)$$

where  $\epsilon$  is a noise term and we assume  $\|\epsilon\| \leq \bar{\epsilon}$ . In this chapter, on top of the dynamics model introduced in Chapter 2, we also consider a cost function  $c(x, u)$ , a terminal cost function  $c_f(x, u)$  and a safety set  $\mathcal{S}$ . Our goal is to optimally move a robot from its start state  $x_0$  to its goal state  $x_f$  at the terminal time  $t_f$ , which can be framed as the following time-optimal control problem:

$$\begin{aligned} \min_{u(t), x(t), t_f} J &= \int_0^{t_f} c(x(t), u(t)) dt + c_f(x(t_f), u(t_f)) \\ \text{s.t.} \quad &\left\{ \begin{array}{l} \text{dynamics in (6.1)} \\ x(t) \in \mathcal{S}, \quad u(t) \in \mathcal{U}, \quad \forall t \\ x(0) = x_0, \quad \|x(t_f) - x_f\| \leq \delta_f \end{array} \right. \end{aligned} \quad (6.2)$$

where  $\mathcal{U}$  is the control feasible set and  $\delta_f$  is a threshold for the goal reaching.

As discussed in Chapter 4, solving Eq. (6.2) is very challenging because: (1)  $f$  in the dynamics is unknown and highly nonlinear; (2) the whole program is typically highly non-convex and  $t_f$  is also a decision variable; and (3) the safety constraint  $x \in \mathcal{S}$  further complicates the problem. In this chapter, we address these challenges by considering an *episodic learning and control* setting, which can be summarized in Fig. 6.2. We discuss the three important modules in Fig. 6.2 (i.e., model learning and uncertainty quantification, safe planning, and safe control) as follows.

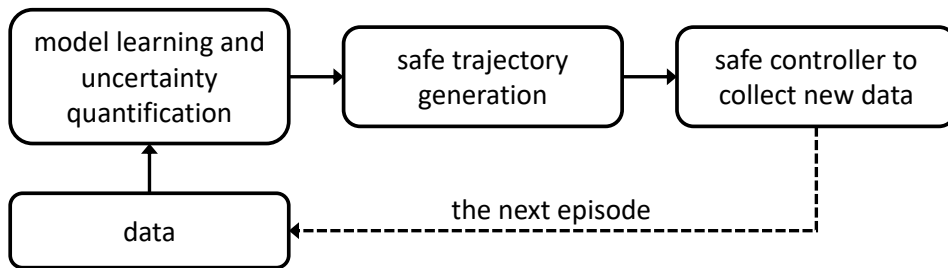


Figure 6.2: Episodic learning and control diagram.

**Model learning and uncertainty quantification under domain shift.** The goal for the learning module in Fig. 6.2 is to estimate the residual unknown dynamics  $f(x, u)$  in a way that admits rigorous uncertainty estimates for safety certification.

The key challenge is that the training data (from previous episodes or trajectories) and test data (the next episode) are not sampled from the same distribution, which can be framed as covariate shift (Shimodaira, 2000). Generally speaking, covariate shift refers to distribution shift caused by the input variables  $P(x, u)$ , while keeping  $P(y = f(x, u)|x, u)$  fixed. In our motivating safe landing example, there is a universal “true” aerodynamics model, but we typically only observe training data from a limited source data distribution  $P_{\text{src}}(x, u)$ . Certifying safety of a proposed trajectory will inevitably cover  $(x, u)$  that are not well-represented by the training, i.e., data from a target data distribution  $P_{\text{trg}}(x, u)$ . In other words, the distribution of  $(x, u)$  in a proposed trajectory (for the next episode) does not match the distribution of  $(x, u)$  in the training data.

The output of the learning module includes an estimate of the unknown dynamics,  $\hat{f}(x, u)$ , and the uncertainty quantification of  $\hat{f}(x, u)$ , i.e., bounding the learning error  $d(x, u) = \hat{f}(x, u) - f(x, u)$ . Note that such an uncertainty quantification could be either deterministic in the worst case or stochastic in expectation. Moreover, most importantly, the uncertainty quantification should be with respect to the *target* data distribution  $P_{\text{trg}}(x, u)$  rather than the source.

**Safe trajectory generation.** Given the learned model  $\hat{f}(x, u)$  and the quantified uncertainty from the learning module, the next step is to optimize (or generate) a safe and feasible reference trajectory  $\{x_d(t), u_d(t)\}_{t=0}^{t_f}$  from solving Eq. (6.2).

However, as discussed before, solving Eq. (6.2) is typically intractable, so prior works typically solve surrogates of Eq. (6.2), e.g., using sequential convex programming (SCP, see Chapter 4 for an example). Another type of surrogate approximately solves Eq. (6.2) by searching or sampling (e.g., see the AO-RRT method discussed in Chapter 4).

Note that the true dynamics  $f$  is unknown when solving Eq. (6.2). Therefore, we must use  $\hat{f}$  and  $d$  to solve a *robust* optimal control problem considering all possible  $f$  given  $\hat{f}$  and the learning error  $d$ .

A typical technical assumption one needs to make to start the episodic process is that, the safe trajectory generation problem is *feasible* in the beginning. Namely, we need to assume that the aforementioned robust optimal control problem is feasible when  $\hat{f} = \hat{f}^{(0)}$  and  $d = d^{(0)}$  where  $\hat{f}^{(0)}$  is the initial guess of the unknown dynamics (typically set to be 0) and  $d^{(0)} = \hat{f}^{(0)} - f$  is the initial uncertainty bound (typically depending on the boundedness of  $f$  and its Lipschitz properties). We make the same

assumption in this chapter. It is not hard to show that the safe exploration problem is ill-posed without such an assumption.

**Safe controller design to collect new data.** The safe trajectory generator outputs a safe reference trajectory  $\{x_d(t), u_d(t)\}_{t=0}^{t_f}$ . Taking  $u_d(t)$  alone, one might expect that it is enough to directly execute the open-loop control signal  $u_d(t)$  without any feedback. However, the open-loop trajectory  $u_d(t)$  might be arbitrarily fragile and not guaranteed to be safe because of higher order model uncertainty not covered in Eq. (6.2) (this is thoroughly discussed in Åström and Murray (2021)).

Therefore, we still need to design a closed-loop tracking controller to guarantee the closed-loop safety. Note that such a controller must also be *robust* to the model uncertainty  $d$ . For example, the nonlinear tracking controllers developed in Chapter 3 and the adaptive controllers developed in Chapter 5 are both robust and safe in the sense that the tracking error  $\|x - x_d(t)\|$  will be proportionally bounded by  $\|d\|$ . Another example of safe controller is based on robust Control Barrier Functions (CBFs) (Ames et al., 2016; Cheng, Orosz, et al., 2019), which can be viewed as a “safety filter” on top of  $u_d(t)$ .

**Trade-offs between exploration and exploitation.** The exploration-exploitation trade-off is one of the most important algorithmic considerations in bandits (Bubeck, Cesa-Bianchi, et al., 2012) and active learning (Srinivas et al., 2010). Similarly, such a trade-off is equally important in the episodic learning and control setting. Namely,  $\{x_d(t), u_d(t)\}_{t=0}^{t_f}$  is not always trying to minimize the cost function in Eq. (6.2) (i.e., exploitation). Often times, we need to design  $\{x_d(t), u_d(t)\}_{t=0}^{t_f}$  such that they can gather new information to reduce the uncertainty  $d$  (i.e., exploration), rather than just minimizing the cost given the current information. For more in-depth discussions please see Nakka et al. (2021).

**End-to-end theoretical guarantees: consistency and convergence.** In the episodic learning and control setting, we pursue end-to-end guarantees in the sense of consistency and convergence. Note that we assume the initial episode is feasible so we have an initial bounded cost  $J^{(0)} < \infty$ . We pursue the following two guarantees: (1) all episodes are safe ( $x_d \in \mathcal{S}$  and  $x \in \mathcal{S}$ ); and (2)  $J^{(i+1)} < J^{(i)}$ , which is weaker than  $\lim_{i \rightarrow \infty} J^{(i)} = J^*$  where  $J^*$  is the optimal value of Eq. (6.2) with the perfect knowledge of  $f$ . To have as strong guarantees as  $\lim_{i \rightarrow \infty} J^{(i)} = J^*$ , we need much more assumptions regarding the problem structure (e.g., system is linear and cost functions are quadratic).



**Stochastic variant of the optimal control problem.** Equation (6.2) considers a *deterministic* optimal control problem. In Nakka et al. (2021), we also consider a stochastic variant, where  $x$  is a stochastic random variable governed by a stochastic differential equation (SDE). The safety requirement is changed to a *chance constraint*  $\Pr(x \in \mathcal{S}) \geq 1 - \delta_s$ . In this case, the goal is to generate a deterministic control signal  $u_d$  such that it minimizes the cost  $J$  in expectation and also satisfies the chance constraint for safety. We will focus on the deterministic optimal control problem in this chapter for simplicity and refer to Nakka et al. (2021) for more details in the stochastic case.

### 6.3 Uncertainty Quantification under Domain Shift

Recall that the key challenge of the learning module in Fig. 6.2 is that the training data (from previous episodes or trajectories) and test data (the next episode) are not sampled from the same distribution. More specifically, we observe training data from a source data distribution  $P_{\text{src}}(x, u)$ , and we need to quantify the model uncertainty in a target data distribution  $P_{\text{trg}}(x, u)$ . In this section, we will present a robust regression framework for uncertainty quantification under domain shift.

**General intuition.** We use robust regression (Chen et al., 2016) to estimate the residual dynamics under covariate shift. Robust regression is derived from a minimax estimation framework, where the estimator  $P(y|x, u)$  tries to minimize a loss function  $\mathcal{L}$  on target data distribution, and the adversary  $Q(y|x, u)$  tries to maximize the loss under source data constraints  $\Gamma$ :

$$\min_{P(y|x,u)} \max_{Q(y|x,u) \in \Gamma} \mathcal{L}. \quad (6.3)$$

Using the minimax framework, we achieve robustness to the worst-case possible conditional distribution that is “compatible” with finite training data if the estimator reaches the Nash equilibrium by minimizing a loss function defined on target data distribution.

**Technical design choices.** Our derivation hinges on a choice of loss function  $\mathcal{L}$  and constraint set for the adversary  $\Gamma$ , from which one can derive a formal objective, a learning algorithm, and an uncertainty bound. We use a relative loss function defined as the difference in conditional log-loss between an estimator  $P(y|x, u)$  and a baseline conditional distribution  $P_0(y|x, u)$  on the target data distribution  $P_{\text{trg}}(x, u)P(y|x, u)$ :

$$\text{relative loss } \mathcal{L} := \mathbb{E}_{P_{\text{trg}}(x,u)Q(y|x,u)} \left[ -\log \frac{P(y|x,u)}{P_0(y|x,u)} \right]. \quad (6.4)$$

To construct the constraint set  $\Gamma$ , we utilize statistical properties of the source data distribution  $P_{\text{src}}(x, u)$ :

$$\Gamma := \left\{ Q(y|x, u) \mid \left| \mathbb{E}_{P_{\text{src}}(x,u)Q(y|x,u)} [\Phi(x, u, y)] - \mathbf{c} \right| \leq \lambda \right\} \quad (6.5)$$

where  $\Phi(x, u, y)$  correspond to the sufficient statistics of the estimation task, and  $\mathbf{c} = \frac{1}{n} \sum_i^n \Phi(x_i, u_i, y_i)$  is a vector of sample mean of statistics in the source data. This constraint means the adversary cannot choose a distribution whose sufficient statistics deviate too far from the collected training data.

The consequence of the above choices is that the solution has a parametric form:

$$P(y|x, u) \propto P_0(y|x, u) e^{\frac{P_{\text{src}}(x,u)}{P_{\text{trg}}(x,u)} \theta^T \Phi(x,u,y)}. \quad (6.6)$$

This form has two useful properties. First, it is straightforward to compute gradients on  $\theta$  using only the training data. One can also train deep neural networks by treating  $\Phi(x, u, y)$  as the last hidden layer, i.e, we learn a representation of the sufficient statistics. Second, this form yields a concrete uncertainty bound that can be used to certify safety. For specific choices of  $P_0$  and  $\Phi$ , the uncertainty is Gaussian distributed, which can be useful for many stochastic control approaches that assume Gaussian uncertainty (see Nakka et al. (2021)). Moreover, our method generalizes naturally to multidimensional output setting, where we predict a multidimensional Gaussian.

The learning performance (i.e., the learning error  $d(x, u)$ ) of robust regression approach can be analyzed from two perspectives: generalization error under covariant shift and perturbation error based on Lipschitz continuity. The generalization error reflects the expected error on a target distribution given certain function class, bounded distribution discrepancy, and base distribution. The perturbation error reflects the maximum error if target data deviates from training but stays in a Lipschitz ball. These error bounds are compatible with deep neural networks whose Rademacher complexity and Lipschitz constant can be controlled and measured (e.g., spectrally normalized neural networks).

**Generalization error bound (informal).** In this case, we bound the generalization error *on the target data*:

$$\mathbb{E}_{P_{\text{trg}}(x,u,y)} [\|f(x,u) - \hat{f}(x,u)\|^2] \leq \left( \sup_{x,u \sim P_{\text{src}}(x)} \frac{P_{\text{trg}}(x,u)}{P_{\text{src}}(x,u)} \right) \cdot C_1 \quad (6.7)$$

with probability  $1 - \delta_g$ .  $C_1$  is from standard statistical learning theory and depends on the Rademacher complexity, base distribution  $P_0$ , the number of data points, and  $\log(1/\delta_g)$ . Most importantly,  $C_1$  is multiplied by the *density ratio*  $\frac{P_{\text{trg}}(x,u)}{P_{\text{src}}(x,u)}$ , which highlights the extra challenge from domain shifts. We refer to Liu, Shi, et al. (2020) for the formal statement and proof.

**Perturbation error bound (informal).** Alternatively, if we assume that target data samples in a ball  $\mathbb{B}(\delta_b)$  with diameter  $\delta_b$  from the source data, we can also bound  $\sup_{x,u \in \mathbb{B}(\delta_b)} [\|f(x,u) - \hat{f}(x,u)\|^2]$  based on the Lipschitz properties of the true dynamics  $f$  and the learner  $\hat{f}$ . We refer to Liu, Shi, et al. (2020) for the formal statement and proof.

## 6.4 Safe Planning and Control

In this chapter, we solve the planning problem with a finite pool of reference trajectories:  $\{x_{d,1}(t), \dots, x_{d,N}(t)\}$ . These trajectories are safe (i.e.,  $x_d \in \mathcal{S}$ ) but with decreasing margins. For example,  $x_{d,1}$  is very conservative (far away from the boundary of  $\mathcal{S}$ , e.g., the slow trajectory in Fig. 6.1) and  $x_{d,N}$  is very aggressive (close to the boundary of  $\mathcal{S}$ , e.g., the fast and safe trajectory in Fig. 6.1).

With  $\hat{f}(x,u)$  and the learning error bound (i.e., the bound of  $\|d(x,u)\| = \|\hat{f}(x,u) - f(x,u)\|$ ), we can apply the learning-based controller developed in Chapter 3. For a reference trajectory candidate  $x_d$ , we consider its neighborhood  $P_{\text{trg}}$ . Define  $\bar{d} = \sup_{x \in P_{\text{trg}}} \|d\|$  as the worst-case learning error in the target domain. Then the tracking controller will guarantee  $\|x - x_d\| \leq C_2 \cdot (\bar{d} + \bar{\epsilon})$ .

Therefore, for trajectory generation, we select the most aggressive trajectory in the pool such that the worst case closed-loop behavior is still safe. Namely, we select  $x_{d,i}$  such that  $x_{d,i}$  plus a  $C_2 \cdot (\bar{d} + \bar{\epsilon})$  error tube remains safe.

This trajectory-pool-based method is simple, efficient, and commonly used in practice. However, as expected, it needs heuristics to design and typically yields a sub-optimal solution. Therefore, we solve the full optimal control problem in Nakka et al. (2021).

## 6.5 Simulations

We conduct simulation experiments on the inverted pendulum and drone landing. We use kernel density estimation to estimate the density ratios. We demonstrate that our approach can reliably and safely converge to optimal behavior. We also compare with a Gaussian process (GP) version of the proposed algorithm. In general, we find it is difficult to tune the GP kernel parameters, especially in the multidimensional output cases.

Both inverted pendulum and drone can be described by Eq. (2.1):  $M(q)\ddot{q}+C(q, \dot{q})\dot{q}+g(q) = Bu + f(q, \dot{q})$ . For pendulum,  $f$  is the unknown drag torque from an analytic quadratic air drag model; for the drone landing problem,  $f$  is represented as a DNN learned in Chapter 3. For pendulum the safety set  $\mathcal{S}$  is defined such that the angle is within  $[-1.5, 1.5]$  radius; for drone it is defined such that the landing velocity is small than 1 m/s.

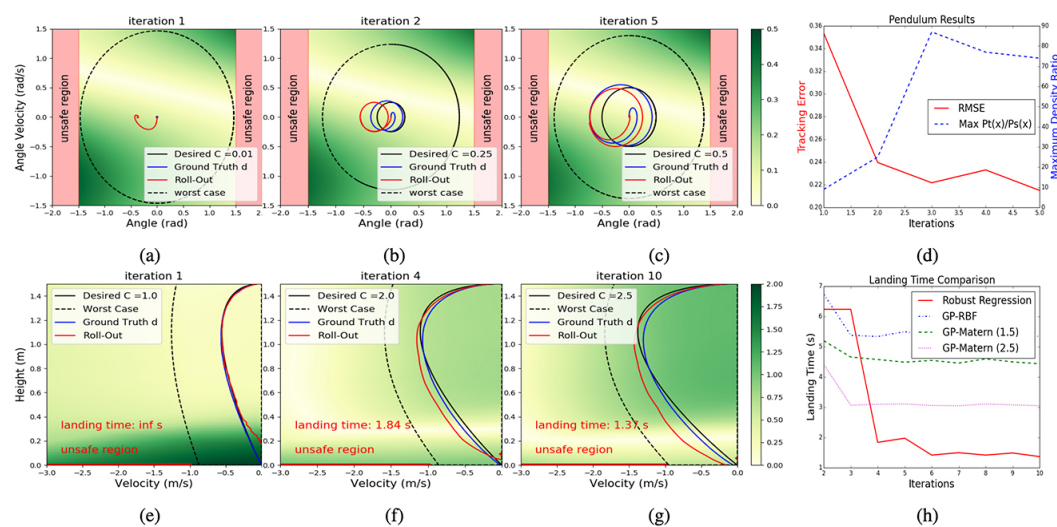


Figure 6.3: Experimental results. Top: The pendulum task: (a)-(c) are the phase portraits of angle and angular velocity; Blue curve is tracking the desired trajectory with perfect knowledge of  $f$ ; the worst-case possible trajectory is calculated according to the uncertainty bound; heatmap is the difference between predicted dynamics (the wind) and the ground truth; and (d) is the tracking error and the maximum density ratio. Bottom: The drone landing task: (e)-(g) are the phase portraits with height and velocity; heatmap is difference between the predicted ground effect) and the ground truth; (h) is the comparison with GPs in landing time.

Main experimental results are shown in Fig. 6.3. Figure 6.3(a) to (c) and (e) to (g) demonstrate the exploration process with selected desired trajectories, worst-case tracking trajectory under current dynamics model, tracking trajectories with the ground truth unknown dynamics model, and actual tracking trajectories. In

both tasks, the algorithm selects conservative trajectories to guarantee safety at the beginning, and gradually is able to select more aggressive trajectories while staying the worst-case safety. We also demonstrate the decaying tracking error in each iteration for the pendulum task in Fig. 6.3(d). We validate that our density ratio is always bounded along the exploration. We examine the drone landing time in Fig. 6.3(h) and compare against multitask GP models (Bonilla et al., 2008) with both RBF kernel and Matern kernel. Our approach outperforms all GP models. Modeling the ground effect is notoriously challenging (see Chapter 3), and the GP suffers from model misspecification, especially in the multidimensional setting (Owhadi et al., 2015). Besides, GP models are also more computationally expensive than our method in making predictions. In contrast, our approach can fit general non-linear function estimators such as deep neural networks adaptively to the available data efficiently, which leads to more flexible inductive bias and better fitting of the data and uncertainty quantification.

## References

- Akametalu, Anayo K., Jaime F. Fisac, Jeremy H. Gillula, Shahab Kaynama, Melanie N. Zeilinger, and Claire J. Tomlin (2014). *Reachability-based safe learning with Gaussian processes*. In: *53rd IEEE Conference on Decision and Control*. IEEE, pp. 1424–1431.
- Alshiekh, Mohammed, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu (2018). *Safe reinforcement learning via shielding*. In: *Thirty-Second AAAI Conference on Artificial Intelligence*.
- Ames, Aaron D., Xiangru Xu, Jessy W. Grizzle, and Paulo Tabuada (2016). *Control barrier function based quadratic programs for safety critical systems*. In: *IEEE Transactions on Automatic Control* 62.8, pp. 3861–3876.
- Åström, Karl Johan and Richard M. Murray (2021). *Feedback systems: An introduction for scientists and engineers*. Princeton University Press.
- Auer, Peter (2002). *Using confidence bounds for exploitation-exploration trade-offs*. In: *Journal of Machine Learning Research* 3.Nov, pp. 397–422.
- Bartlett, Peter L., Dylan J. Foster, and Matus J. Telgarsky (2017). *Spectrally-normalized margin bounds for neural networks*. In: *Advances in Neural Information Processing Systems*, pp. 6240–6249.
- Benjamin, Ari S., David Rolnick, and Konrad Kording (2019). *Measuring and regularizing networks in function space*. In: *International Conference on Learning Representations (ICLR)*.
- Berkenkamp, Felix, Angela P. Schoellig, and Andreas Krause (2016). *Safe controller optimization for quadrotors with Gaussian Processes*. In: *IEEE International*

- Conference on Robotics and Automation (ICRA)*, pp. 493–496. URL: <https://arxiv.org/abs/1509.01066>.
- Berkenkamp, Felix, Matteo Turchetta, Angela P. Schoellig, and Andreas Krause (2017). *Safe model-based reinforcement learning with stability guarantees*. In: *Neural Information Processing Systems (NIPS)*. URL: <https://arxiv.org/abs/1705.08551>.
- Bonilla, Edwin V., Kian M. Chai, and Christopher Williams (2008). *Multi-task Gaussian process prediction*. In: *Advances in Neural Information Processing Systems*, pp. 153–160.
- Bubeck, Sébastien, Nicolo Cesa-Bianchi, et al. (2012). *Regret analysis of stochastic and nonstochastic multi-armed bandit problems*. In: *Foundations and Trends® in Machine Learning* 5.1, pp. 1–122.
- Chen, Xiangli, Mathew Monfort, Anqi Liu, and Brian D. Ziebart (2016). *Robust covariate shift regression*. In: *Artificial Intelligence and Statistics*, pp. 1270–1279.
- Cheng, Richard, Gábor Orosz, Richard M. Murray, and Joel W. Burdick (2019). *End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks*. In: *Conference on Artificial Intelligence (AAAI)*.
- Cheng, Richard, Abhinav Verma, Gábor Orosz, Swarat Chaudhuri, Yisong Yue, and Joel Burdick (2019). *Control regularization for reduced variance reinforcement learning*. In: *International Conference on Machine Learning*, pp. 1141–1150. URL: <http://proceedings.mlr.press/v97/cheng19a.html>.
- Fisac, Jaime F., Anayo K. Akametalu, Melanie N. Zeilinger, Shahab Kaynama, Jeremy Gillula, and Claire J. Tomlin (2018). *A general safety framework for learning-based control in uncertain robotic systems*. In: *IEEE Transactions on Automatic Control*.
- Garcia, Javier and Fernando Fernández (2012). *Safe exploration of state and action spaces in reinforcement learning*. In: *Journal of Artificial Intelligence Research* 45, pp. 515–564.
- Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy (2014). *Explaining and harnessing adversarial examples*. In: *arXiv preprint arXiv:1412.6572*.
- Kirschner, Johannes, Mojmír Mutny, Nicole Hiller, Rasmus Ischebeck, and Andreas Krause (2019). *Adaptive and safe Bayesian optimization in high dimensions via one-dimensional subspaces*. In: *International Conference on Machine Learning (ICML)*.
- Le, Hoang M., Andrew Kang, Yisong Yue, and Peter Carr (2016). *Smooth imitation learning for online sequence prediction*. In: *International Conference on Machine Learning (ICML)*.

- Liu, Anqi, Guanya Shi, Soon-Jo Chung, Anima Anandkumar, and Yisong Yue (2020). *Robust regression for safe exploration in control*. In: *Learning for Dynamics and Control*. PMLR, pp. 608–619. URL: <https://proceedings.mlr.press/v120/liu20a.html>.
- Liu, Anqi and Brian Ziebart (2014). *Robust classification under sample selection bias*. In: *Advances in Neural Information Processing Systems*, pp. 37–45.
- Miyato, Takeru, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida (2018). *Spectral normalization for generative adversarial networks*. In: *arXiv preprint arXiv:1802.05957*.
- Moldovan, Teodor Mihai and Pieter Abbeel (2012). *Safe exploration in Markov decision processes*. In: *International Conference on Machine Learning (ICML)*.
- Nakka, Yashwanth Kumar, Anqi Liu, Guanya Shi, Anima Anandkumar, Yisong Yue, and Soon-Jo Chung (2021). *Chance-constrained trajectory optimization for safe exploration and learning of nonlinear Systems*. In: *IEEE Robotics and Automation Letters* 6.2, pp. 389–396. DOI: 10.1109/LRA.2020.3044033.
- Ostafew, Chris J., Angela P. Schoellig, and Timothy D. Barfoot (2016). *Robust constrained learning-based NMPC enabling reliable mobile robot path tracking*. In: *The International Journal of Robotics Research* 35.13, pp. 1547–1563.
- Owhadi, Houman, Clint Scovel, and Tim Sullivan (2015). *On the brittleness of Bayesian inference*. In: *SIAM Review* 57.4, pp. 566–582.
- Prest, Alessandro, Christian Leistner, Javier Civera, Cordelia Schmid, and Vittorio Ferrari (2012). *Learning object class detectors from weakly annotated video*. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, pp. 3282–3289.
- Shimodaira, Hidetoshi (2000). *Improving predictive inference under covariate shift by weighting the log-likelihood function*. In: *Journal of Statistical Planning and Inference* 90.2, pp. 227–244.
- Srinivas, Niranjan, Andreas Krause, Sham M. Kakade, and Matthias Seeger (2010). *Gaussian process optimization in the bandit setting: No regret and experimental design*. In: *International Conference on Machine Learning (ICML)*.
- Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2014). *Dropout: A simple way to prevent neural networks from overfitting*. In: *The Journal of Machine Learning Research* 15.1, pp. 1929–1958.
- Stewart, Russell and Stefano Ermon (2017). *Label-free supervision of neural networks with physics and domain knowledge*. In: *Thirty-First AAAI Conference on Artificial Intelligence*.
- Sui, Yanan, Alkis Gotovos, Joel Burdick, and Andreas Krause (2015). *Safe exploration for optimization with Gaussian processes*. In: *International Conference on Machine Learning*, pp. 997–1005.

- Sui, Yanan, Vincent Zhuang, Joel W. Burdick, and Yisong Yue (2018). *Stagewise safe Bayesian optimization with Gaussian processes*. In: *International Conference on Machine Learning (ICML)*.
- Taylor, Andrew J., Victor D. Dorobantu, Hoang M. Le, Yisong Yue, and Aaron D. Ames (2019). *Episodic learning with control Lyapunov functions for uncertain robotic systems*. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 6878–6884. DOI: [10.1109/IRoS40897.2019.8967820](https://doi.org/10.1109/IRoS40897.2019.8967820). URL: <https://doi.org/10.1109/IRoS40897.2019.8967820>.
- Turchetta, Matteo, Felix Berkenkamp, and Andreas Krause (2016). *Safe exploration in finite Markov decision processes with Gaussian processes*. In: *Advances in Neural Information Processing Systems*, pp. 4312–4320.
- Wachi, Akifumi, Yanan Sui, Yisong Yue, and Masahiro Ono (2018). *Safe exploration and optimization of constrained mdps using Gaussian processes*. In: *Thirty-Second AAAI Conference on Artificial Intelligence*.
- Wager, Stefan, Sida Wang, and Percy S. Liang (2013). *Dropout training as adaptive regularization*. In: *Advances in Neural Information Processing Systems*, pp. 351–359.
- Zheng, Stephan, Yang Song, Thomas Leung, and Ian Goodfellow (2016). *Improving the robustness of deep neural networks via stability training*. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4480–4488.
- Zhou, Kemin and John Comstock Doyle (1998). *Essentials of robust control*. Vol. 104. Prentice Hall Upper Saddle River, NJ.



## Chapter 7

### DISCUSSION AND FUTURE WORK

In Part I, through Chapters 2 to 6, we have built a holistic view of the *Neural-Control Family*, which is a family of deep-learning-based nonlinear controllers with theoretical guarantees and learned new capabilities in robotics. See Table 2.1 for a summary of four main members in the family: Neural-Lander, Neural-Swarm, Neural-Fly, and Safe Exploration.

In this chapter, we will conclude Part I by discussing several important future research directions.

#### 7.1 Neural-Control for Other Robotic Systems

In Neural-Lander (Chapter 3), Neural-Swarm (Chapter 4), and Neural-Fly (Chapter 5), we all considered a mixed robot dynamics model in Eq. (2.1):

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = Bu + \underbrace{f(q, \dot{q}, u, t)}_{\text{unknown}}$$

which is fully observable (we can directly observe  $q$  and  $\dot{q}$ ) and fully actuated ( $\text{rank}(B) = \text{dim}(q) = n$ ). In this section, we will discuss how to generalize the *Neural-Control* methodology to other systems.

#### Partially Observable Systems

By and large, learning and control with end-to-end theoretical guarantees in partially observable systems is very challenging due to the following fundamental challenges.

**Coupling between estimation and control.** In linear time-invariant (LTI) systems, the well-known *separation principle* (Åström and Murray, 2021) suggests that we can safely decouple the design of the state estimator and the design of the feedback controller. Moreover, one can show that in a LQG problem (a LTI system with Gaussian noise and quadratic costs), an optimal Kalman filter plus an optimal LQR controller is jointly optimal (Åström and Murray, 2021).

The separation principle is also widely applied in practice, and often yields satisfactory performance. For example, in the outdoor flight experiment in Chapter 5 (see Fig. 5.8), we first used a Kalman filter to estimate the drone state, and then use our Neural-Fly method for control by treating the estimated state as the true state.

However, generally speaking, for either time-variant or nonlinear systems, the separation principle is no longer true. That being said, a good estimator plus a good controller does not necessarily imply a good policy. In the worst case, the overall policy could even be unstable! Therefore, one must jointly analyze the estimation and control in a loop. The analysis becomes further involved if either of them is data-driven.

**Fundamental limits of robustness.** To apply data-driven methods in partially observable systems, we must consider how learning error propagates through the system and how to guarantee robustness against such an error. For fully observable systems, we achieve this by designing exponentially stable feedback controllers, which naturally guarantees robustness against imperfect learning (e.g., Theorem 3.1 and Theorem 5.1).

However, unfortunately, robustness in partially observable system raises a lot more fundamental limits. For example, John Doyle constructed a simple counter example in his well-known paper in 1978 (Doyle, 1978), to show that an optimal LQG controller could be *arbitrarily fragile* in the sense of having an arbitrarily small gain margin. Those fundamental limits suggest that, when designing learning and control method for partially observable systems, we might have to trade-off between optimality and robustness.

**Uncertainty of vision-based deep learning models.** One interesting future direction is to apply *Neural-Control* in vision-based control settings, i.e., only with raw vision data.

However, deep-learning-based computer vision methods are typically overconfident, especially under domain shift. For example, we show that state-of-the-art deep-learning-based object pose detectors are very sensitive to the training data and could be overconfident (Shi et al., 2021). In Fig. 7.1, we trained three different deep pose detectors with different simulation data under different randomization, and tested them in two real-world images for the milk object detection task. In the left image, these three models highly disagree with each other but in the right image (the only change is the light condition) they almost give identical detection results. Figure 7.1 suggests that when designing learning and control methods with vision input, we must carefully consider domain shift (especially sim-to-real shift) issues and quantify the uncertainty of the deep learning model in principled ways.



Figure 7.1: Detection results of the milk object of three deep learning models in two images. In the left image, these three models disagree much more than the right image.

### Under-Actuated Systems

The fully-actuated structure in Eq. (2.1) has two major benefits in learning and control: (1) We can freely command  $Bu$  in  $\mathbb{R}^n$ , which allows us to design a globally exponentially stable controller for the nominal system. (2) The unknown term  $f$  becomes a *matched uncertainty* term. Namely, we can compensate (or cancel)  $f$  via the certainty-equivalence principle.

For under-actuated systems, in general we need more structures for both (1) and (2). For example, if we know a Control Lyapunov Function (CLF) a priori, we can leverage the CLF for stabilization (e.g., Taylor et al. (2019)) or adaptive control (e.g., Lopez and Slotine (2021)). Similarly, Westenbroek et al. (2019) designed a particular reinforcement learning method assuming the system is feedback linearizable with a known relative degree. However, those methods typically need strong prior knowledge (e.g., a CLF).

### Robotic Systems with Constraints

Another interesting future direction is to consider constraints. Here we briefly discuss two types of constraints.

**Control input constraint** considers  $u \in \mathcal{U}$  where  $\mathcal{U}$  is the control input feasible set. Generally speaking, dealing with such a constraint needs multi-step planning (e.g., via MPC) or reachability analysis.

**Holonomic constraint** augments Eq. (2.1) to

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = Bu + \underbrace{f(q, \dot{q}, u, t)}_{\text{unknown}} + J^T(q)\lambda$$

$$J(q)\ddot{q} + \dot{J}(q)\dot{q} = 0$$

where  $J$  is the Jacobian of the holonomic constraint and  $\lambda$  is the Lagrange multi-

plier (Murray et al., 2017). Such a system can be used to model legged robots and manipulators.

## 7.2 End-to-End Learning and Control Guarantees

Our tracking error bounds in Theorem 3.1 and Theorem 5.1 assume known learning error bounds. More specifically, in Theorem 3.1 we need to know  $\|\hat{f} - f\|_\infty$  in a compact set, and in Theorem 5.1 we need to know that the learned representation  $\phi$  can adapt sufficiently well in any new wind condition.

One interesting and important future research direction is to establish an *end-to-end guarantee*, by quantitatively and directly bounding the tracking error as a function of the number of data points. However, this could be very challenging because: (1) Classic learning theory needs to assume i.i.d. data distribution in training and testing data, but the nature of dynamical systems yields neither identical nor independent data distribution. (2) Classic learning theory needs a non-vacuous notion of learning model capacity (e.g., coverage number or Rademacher complexity, see details in Vapnik (1999)), but modern neural networks are in general *overparameterized*, which makes classic generalization error bound more or less vacuous.

By and large, end-to-end guarantees with “classic” learning method (e.g., least squares) in “simple” systems (e.g., LQR) are tractable, and have been attracting a lot of research attention recently (e.g., regret bounds for learning in LQR problems (Dean et al., 2020)). However, for nonlinear systems with “modern” learning methods, the analysis is much more complicated and in general more structured assumptions are required (e.g., we limited the spectral complexity of the DNN for end-to-end non-vacuous guarantees in the episodic setting in Chapter 6).

## 7.3 Model-Free and Model-Based Learning

In Part I, we have been focusing on the *model-based learning* scheme, where we first learn the model  $f$ , and then design a policy based on the learned model.

The model-based methodology enjoys several properties: (1) It yields a clean interface between learning theory and control theory, which allows us to build a unified framework between control and learning. (2) Compared to the model-free approach, it is much easier to have rigorous safety and robustness guarantees. (3) Often times, model-based methods are more data-efficient and interpretable (Kaiser et al., 2019).

However, compared to the model-based approach, model-free policy learning is potentially more flexible and more globally optimal. One future direction is to combine

model-free policy learning and model-based methods. In particular, one promising direction is encoding control-theoretic knowledge to model-free reinforcement learning. Most popular RL algorithms (e.g., TRPO, SAC) are *universal* for all tasks. In contrast, drastically different nonlinear control methods are developed for different systems/tasks, and their successes highly rely on structures inside these systems/tasks. It is important and interesting to encode these structures into RL algorithms in a principled manner.

#### 7.4 Neurosymbolic Learning for Data-Efficient Decision Making

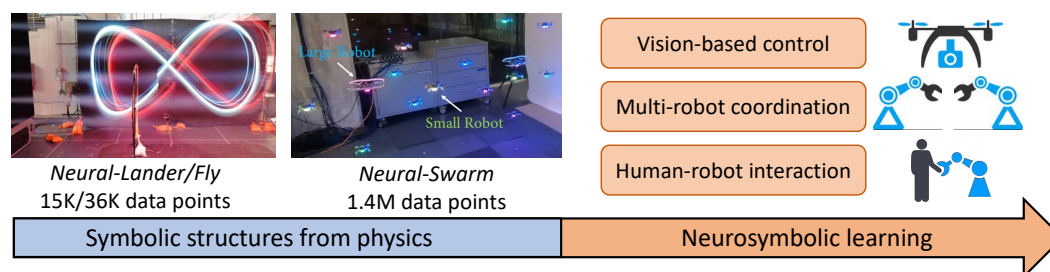


Figure 7.2: From Part I to future research (more data required from left to right).

In Part I, we found that more data is needed as tasks get more involved (Fig. 7.2). *Neural-Lander* needs 15K data points (5-min flight) (Chapter 3); Moving to the multi-environment case, *Neural-Fly* needs 36K points (12-min) (Chapter 5); For the heterogeneous swarm case, *Neural-Swarm* needs 1.4M points (Chapter 4).

One key observation is that *having symbolic structures in black box learning is critical for learning in autonomous systems*. For example, if we learn the full dynamics instead of the residual for drone landing, we need 1hour of data instead of 5min. Encoding permutation invariance and domain invariance also greatly improves sample efficiency for *Neural-Swarm* and *Neural-Fly*, respectively.

However, these structures are from physics and relatively straightforward to discover. Neurosymbolic learning (Chaudhuri et al., 2021) is required when moving to more complex scenarios, including vision-based control, multi-robot coordination, and human-robot interaction (depicted in Fig. 7.2). For these settings, data is often very high-dimensional, and there is no clear symbolic prior such as the nominal dynamics in *Neural-Lander*. Therefore we need to develop a principled neurosymbolic learning framework that discovers symbolic priors from data and integrates these priors with control theory. For instance, in vision-based control, we can discover *causal* structures by learning a low-dimensional representation that causally relates to the control task, then integrating these structures with robust and optimal control

theory. Another example is multi-robot coordination with uncertainty: consider three drones carrying a payload in strong winds. Drones need to share common information (e.g., wind effect) and negotiate potential conflicts (e.g., which drone moves first). We need to disentangle the shared and conflict parts and incorporate them using game theory and hierarchical planning.

## 7.5 Other Residual Learning Methods

In Part I, we have been focusing on residual learning in the *dynamics level*. Namely, we learn  $f_{\text{unknown}}$  in  $\dot{x} = f_{\text{known}} + f_{\text{unknown}}$ . Note that there are at least two other residual learning methods:

**Action-level residual learning** learns a residual action  $\Delta u$  in  $u = u_{\text{nominal}} + \Delta u$ . For example, Johannink et al. (2019) learns  $\Delta u$  using deep RL methods on top of a manually designed  $u_{\text{nominal}}$ .

**Program-level residual learning** aims to learn the residual information in a “program.” For example, Taylor et al. (2019) learns the projection of the residual dynamics on a Lyapunov function, rather than the residual dynamics itself; Amos et al. (2018) learns system parameters or cost functions via a differentiable MPC program; Zhou et al. (2017) learns reference signals as an add-on block for the nominal feedback controller.

In the future, it will be very interesting to compare these different levels of residual learning and understand their trade-offs.

## References

- Amos, Brandon, Ivan Jimenez, Jacob Sacks, Byron Boots, and Zico Kolter (2018). *Differentiable MPC for end-to-end planning and control*. In: *Advances in Neural Information Processing Systems* 31.
- Åström, Karl Johan and Richard M. Murray (2021). *Feedback systems: An introduction for scientists and engineers*. Princeton University Press.
- Chaudhuri, Swarat, Kevin Ellis, Oleksandr Polozov, Rishabh Singh, Armando Solar-Lezama, Yisong Yue, et al. (2021). *Neurosymbolic Programming*. In: *Foundations and Trends® in Programming Languages* 7.3, pp. 158–243.
- Dean, Sarah, Horia Mania, Nikolai Matni, Benjamin Recht, and Stephen Tu (2020). *On the sample complexity of the linear quadratic regulator*. In: *Foundations of Computational Mathematics* 20.4, pp. 633–679.
- Doyle, John C. (1978). *Guaranteed margins for LQG regulators*. In: *IEEE Transactions on Automatic Control* 23.4, pp. 756–757.

- Johannink, Tobias et al. (2019). *Residual reinforcement learning for robot control*. In: *IEEE International Conference on Robotics and Automation*, pp. 6023–6029. DOI: 10.1109/ICRA.2019.8794127. URL: <https://doi.org/10.1109/ICRA.2019.8794127>.
- Kaiser, Lukasz et al. (2019). *Model-based reinforcement learning for Atari*. In: *arXiv preprint arXiv:1903.00374*.
- Lopez, Brett T. and Jean-Jacques E. Slotine (2021). *Universal adaptive control of nonlinear systems*. In: *IEEE Control Systems Letters* 6, pp. 1826–1830.
- Murray, Richard M., Zexiang Li, and S. Shankar Sastry (Dec. 2017). *A mathematical introduction to robotic manipulation*. 1st ed. CRC Press. ISBN: 978-1-315-13637-0. DOI: 10.1201/9781315136370. URL: <https://www.taylorfrancis.com/books/9781351469791>.
- Shi, Guanya, Yifeng Zhu, Jonathan Tremblay, Stan Birchfield, Fabio Ramos, Animashree Anandkumar, and Yuke Zhu (2021). *Fast uncertainty quantification for deep object pose estimation*. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5200–5207. DOI: 10.1109/ICRA48506.2021.9561483.
- Taylor, Andrew J., Victor D. Dorobantu, Hoang M. Le, Yisong Yue, and Aaron D. Ames (2019). *Episodic learning with control Lyapunov functions for uncertain robotic systems*. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 6878–6884. DOI: 10.1109/IRoS40897.2019.8967820. URL: <https://doi.org/10.1109/IRoS40897.2019.8967820>.
- Vapnik, Vladimir N. (1999). *An overview of statistical learning theory*. In: *IEEE Transactions on Neural Networks* 10.5, pp. 988–999.
- Westenbroek, Tyler, David Fridovich-Keil, Eric Mazumdar, Shreyas Arora, Valmik Prabhu, S. Shankar Sastry, and Claire J. Tomlin (2019). *Feedback linearization for unknown systems via reinforcement learning*. In: *arXiv preprint arXiv:1910.13272*.
- Zhou, Siqi, Mohamed K. Helwa, and Angela P. Schoellig (2017). *Design of deep neural networks as add-on blocks for improving impromptu trajectory tracking*. In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, pp. 5201–5207.

# **Part II**

**Unifying Interfaces Between Learning and Control Theory**



In this part of the thesis, we will discuss three unifying interfaces between learning and control theory. Such interfaces deepen the fundamental connections between the two fields, enable much more efficient translation, and bring new perspectives and algorithmic principles.

In particular, Part II will focus on joint end-to-end learning and control theoretical guarantees in online learning and control problems. In Chapter 8, we will give an overview. In Chapters 9 to 11, we introduce the three interfaces, respectively. This part is mainly based on the following papers:

Li, Tongxin, Ruixiao Yang, Guannan Qu, Guanya Shi, Chenkai Yu, Adam Wierman, and Steven Low (2022). *Robustness and consistency in linear quadratic control with untrusted predictions*. In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 6.1, pp. 1–35. DOI: [10.1145/3508038](https://doi.org/10.1145/3508038).

Lin, Yiheng, Yang Hu, Guanya Shi, Haoyuan Sun, Guannan Qu, and Adam Wierman (2021). *Perturbation-based regret analysis of predictive control in linear time varying systems*. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 34. Curran Associates, Inc., pp. 5174–5185. URL: <https://arxiv.org/abs/2106.10497>.

Pan, Weici, Guanya Shi, Yiheng Lin, and Adam Wierman (2022). *Online optimization with feedback delay and nonlinear switching cost*. In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 6.1, pp. 1–34. DOI: [10.1145/3508037](https://doi.org/10.1145/3508037).

Shi, Guanya, Kamyar Azizzadenesheli, Michael O’Connell, Soon-Jo Chung, and Yisong Yue (2021). *Meta-adaptive nonlinear control: Theory and algorithms*. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 34. Curran Associates, Inc., pp. 10013–10025. URL: <https://proceedings.neurips.cc/paper/2021/file/52fc2aee802efbad698503d28ebd3a1f-Paper.pdf>.

Shi, Guanya, Yiheng Lin, Soon-Jo Chung, Yisong Yue, and Adam Wierman (2020). *Online optimization with memory and competitive control*. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 33. Curran Associates, Inc. URL: <https://arxiv.org/abs/2002.05318>.

Yu, Chenkai, Guanya Shi, Soon-Jo Chung, Yisong Yue, and Adam Wierman (2020). *The power of predictions in online control*. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 33. Curran Associates, Inc., pp. 1994–2004. URL: <https://proceedings.neurips.cc/paper/2020/file/155fa09596c7e18e50b58eb7e0c6ccb4-Paper.pdf>.

– (2022). *Competitive control with delayed imperfect information*. In: *American Control Conference (ACC)*. URL: <https://arxiv.org/abs/2010.11637>.

## Chapter 8

## OVERVIEW

**Notations.** In Part II,  $\|\cdot\|$  indicates the 2-norm of a vector or the induced 2-norm of a matrix.  $\|\cdot\|_F$  denotes the Frobenius norm of a matrix and  $\lambda_{\min}(\cdot)$  ( $\lambda_{\max}(\cdot)$ ) denotes the minimum (maximum) eigenvalue of a real symmetric matrix. We use  $\rho(\cdot)$  to denote the spectral radius of a matrix.  $\text{vec}(\cdot) \in \mathbb{R}^{mn}$  denotes the vectorization of a  $m \times n$  matrix, and  $\otimes$  denotes the Kronecker product. Finally, we use  $x_{1:t}$  to denote a sequence  $\{x_1, x_2, \dots, x_t\}$ .

## 8.1 The Importance of Building Interfaces Between Learning and Control

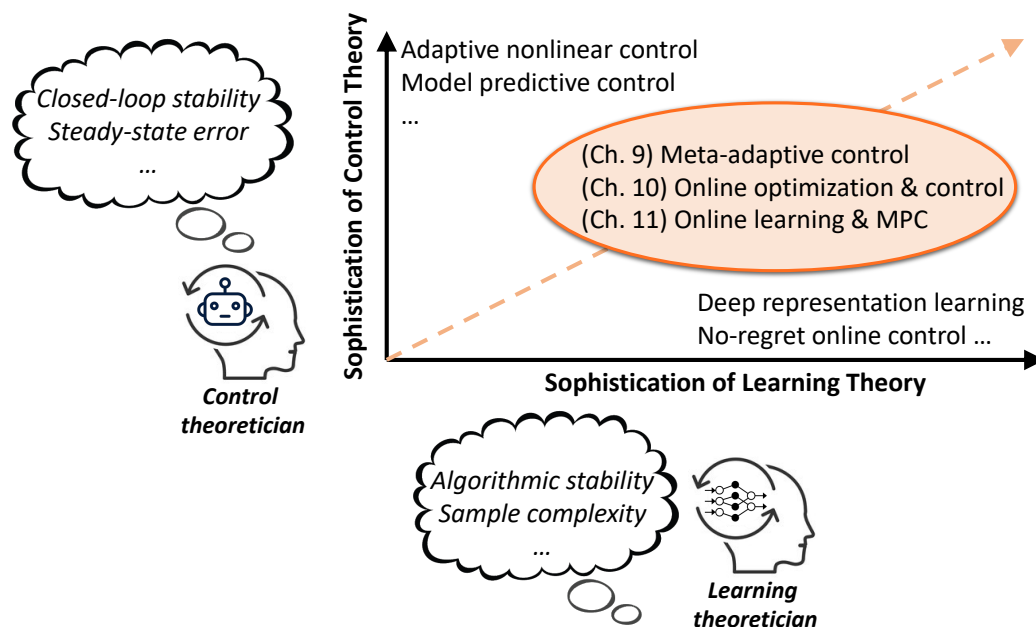


Figure 8.1: Motivations of building the interface between learning and control, and the organization of Part II.

The motivation of Part II is depicted in Fig. 8.1. In y-axis, control theory becomes more and more sophisticated. For example, adaptive nonlinear control (Slotine and W. Li, 1991; Krstic et al., 1995) and model predictive control (Rawlings, 2000) have been heavily researched since the 1980s, because they can deal with highly nonlinear and time-variant systems. Typically, control theoreticians focus on concepts such as closed-loop stability, steady-state error, or robustness.

On the other hand, in x-axis, learning theory (in particular, for decision making) also gets more and more complicated. For example, online learning and control with *no-regret* guarantees (a learning theory concept) has attracted a lot of attention since 2011 (Abbasi-Yadkori and Szepesvári, 2011). Generally speaking, learning theoreticians are interested in pursuing learning-theoretic guarantees such as algorithmic stability or sample complexity.

However, it is worth noting that the top left of Fig. 8.1 has limited learning theory. For instance, most adaptive nonlinear control methods focus on *linear parametric uncertainty* with known basis functions (see the introduction of Chapter 5). The non-parametric-uncertainty case is still very challenging under the framework of model reference adaptive control (MRAC, Åström and Wittenmark (2013)). Combining meta-learning or representation learning and adaptive control for the non-parametric case is a promising direction with recently developed exciting algorithms and experimental results (e.g., Chapter 5 and Richards et al. (2021)), but there is no joint or end-to-end learning and control theoretical analysis.

Similarly, the bottom right of Fig. 8.1 has limited control theory. For example, most no-regret online control research focuses on linear time-invariant systems such as LQR with classic *static* regret guarantees. However, most real robotic systems are highly nonlinear or time-variant (e.g., considering the changing wind condition in Chapter 5).

Since both learning and control perspectives have been developed to a mature level, now is the time to focus on the *interface* (i.e., the orange dashed line in Fig. 8.1) and deeply integrate learning and control theories. The benefit includes:

**Building interfaces deepens the fundamental connections and enables more efficient translation between the two fields.** One of focuses in Part II is to study how to reconcile fairly fragmented analysis between the two fields. For example, for online control problems, what does exponentially stable control imply about online regret (a notion of algorithm stability)? In Chapter 11, we will show that the closed-loop exponential stability is the key reason behind MPC's competitiveness (a even stronger guarantee in online learning than regret). In Chapter 9, we will show that exponential input-to-state stability (e-ISS) is the key to translate learning-theoretic bounds to control-theoretic bounds. These examples demonstrate that building interfaces can deepen understanding and translate rich results between the two fields.

**Building interfaces brings new perspectives and new algorithms.** Often times, interfaces between learning and control yield “learning-inspired control algorithms” or “control-inspired learning algorithms.” In Chapter 11, we analyze a well-known and very successful control algorithm, Model Predictive Control (MPC), from learning theory perspectives. Chapter 11 not only provides new learning-theoretic perspectives of MPC, but yields new algorithms, e.g., a new variant of MPC from a *robustness-consistency* trade-off perspective in online learning.

**Building interfaces also has educational values.** As mentioned before, Model Predictive Control (MPC) has been one of the most successful methods in industrial control since the 1980s. However, many learning theorists are studying RL algorithms, but few are analyzing MPC and why it is so powerful. In particular, in his blog<sup>1</sup>, Prof. Ben Recht said:

*“So many theorists are spending a lot of time studying RL algorithms, but few in the ML community are analyzing MPC and why it’s so successful. We should rebalance our allocation of mental resources! . . . I’d urge the MPC crowd to connect more with the learning theory crowd to see if a common ground can be found to better understand how MPC works and how we might push its performance even farther.”*

Thus, building interfaces can find common ground for learning and control theory, and also provide unique educational values in the sense of diversifying researchers’ research visions and methodology.

## 8.2 Organization of Part II: Three Interfaces

As shown in Fig. 8.1, in this thesis we will focus on the following three interfaces between online learning and control theory:

**Online meta-adaptive control (Chapter 9).** In order to have rapidly adaptable autonomous systems operating in changing environments (e.g., varying wind conditions for drones), it is crucial to extract common representations from all environments. However, existing theoretical results focus on either representation/meta-learning with i.i.d. data (i.e., no dynamics) or adaptive control in a single environment. Therefore, Chapter 9 proposes a novel *meta-adaptive control* framework, where meta-learning optimizes a representation shared by all environments. Then

<sup>1</sup>Link: <http://www.argmin.net/2020/06/29/tour-revisited/>

the representation is fine-tuned by adaptive nonlinear control in a low-dimensional space. Chapter 9 jointly analyzes outer-loop learning and inner-loop adaptive control under a unified framework, and provides the first end-to-end non-asymptotic guarantee for multi-task nonlinear control.

**Competitive online optimization and control (Chapter 10).** The recent progress in learning-theoretic analyses for online control begs the question: What learning-theoretic guarantees do we need for real-world systems? Existing results focus on linear systems with classic no-regret guarantees. However, no-regret policies compare with the optimal static controller in a specific class (typically linear policy class), which could be arbitrarily suboptimal in real-world nonlinear or time-varying systems (see details in Chapter 10). Therefore, Chapter 10 builds interfaces between *competitive online optimization* and control, which uses stronger metrics beyond regret, i.e., competitive ratio and dynamic regret (competitive difference). Those metrics directly compare with the global optimum, thus naturally suitable for real-world systems. We have designed competitive policies in time-varying and nonlinear systems, via novel reductions from online optimization to control. Moreover, we show new fundamental limits via novel lower bounds, e.g., the impact of delay.

**Online learning perspectives on model predictive control (Chapter 11).** Another critical question is begged in online learning and control: Do established control methods such as MPC have strong learning guarantees? To close this gap, Chapter 11 proves the first non-asymptotic guarantee for MPC, showing that MPC is *near-optimal* in the sense of dynamic regret in online LQR control with predictable disturbance. Chapter 11 also extends to settings with delayed inexact predictions and LTV systems. These results found common ground for learning and control theory and imply fundamental algorithmic principles.

### 8.3 Preliminaries on Online Optimization and Learning

In this section, we introduce some preliminaries on online optimization and learning, which will be heavily used in Part II. In online optimization and learning, an online player (or learner) iteratively makes decisions. Most importantly, at the time of each decision, the outcomes associated with the choices are *unknown* to the player. Formally, at time step  $t \in [1, T]$ , the online player picks an action  $x_t$  in a convex action set  $\mathcal{K} \subset \mathbb{R}^d$  and then a loss function  $f_t : \mathcal{K} \rightarrow \mathbb{R}$  is revealed. The player then incurs a loss of  $f_t(x_t)$ . Finally, the player moves to the next round  $t + 1$  and makes a decision  $x_{t+1}$  based on all previous information. Note that the losses  $f_{1:T}$  can be

*adversarially* chosen, and even depend on the action taken by the decision maker.

### Convexity and Smoothness of Functions

For analysis and algorithm design, the loss functions' convexity and smoothness properties play a key role, so we recap related concepts.

**Definition 8.1** (Convex). *A function  $f : \mathcal{K} \rightarrow \mathbb{R}$  is convex if for all  $x, y$  in the relative interior of the domain of  $f$  and  $\lambda \in (0, 1)$ , we have*

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y).$$

**Definition 8.2** ( $m$ -strongly convex). *A function  $f : \mathcal{K} \rightarrow \mathbb{R}$  is  $m$ -strongly convex with respect to a norm  $\|\cdot\|$  if for all  $x, y$  in the relative interior of the domain of  $f$  and  $\lambda \in (0, 1)$ , we have*

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y) - \frac{m}{2}\lambda(1 - \lambda)\|x - y\|^2.$$

Intuitively, the  $m$ -strongly convex property requires extra “curvature” on top of the standard convexity. There are three commonly used equivalent conditions:

- $f - \frac{m}{2}\|x\|^2$  is convex.
- (when  $f$  is differentiable) For all  $x, y$  we have  $f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{m}{2}\|y - x\|^2$ .
- (when  $f$  is twice differentiable) The Hessian is uniformly lower bounded by  $\nabla^2 f(x) \geq mI$ .

**Definition 8.3** ( $l$ -strongly smooth). *A function  $f : \mathcal{K} \rightarrow \mathbb{R}$  is  $l$ -strongly smooth with respect to a norm  $\|\cdot\|$  if  $f$  is everywhere differentiable and if for all  $x, y$  we have*

$$f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{l}{2}\|y - x\|^2.$$

Intuitively, compared to the second equivalent condition of the  $m$ -strongly convex property, the  $l$ -strongly smooth property is “limiting” the curvature of  $f$ . Similarly, if  $f$  is twice differentiable,  $f$  is  $l$ -strongly smooth if and only if  $lI \geq \nabla^2 f(x)$ .

We say a function is *well-conditioned* if it is simultaneously strongly convex and strongly smooth. For example, a quadratic function  $f(x) = \frac{1}{2}x^\top Qx$  is  $\lambda_{\min}(Q)$ -strongly convex and  $\lambda_{\max}(Q)$ -strongly smooth.

## Metrics Used in Online Learning

**Regret.** A commonly used performance metric in online learning is (static) regret. The static regret of an online algorithm is defined by

$$\text{regret} = \sum_{t=1}^T f_t(x_t) - \min_{x^* \in \mathcal{K}} \sum_{t=1}^T f_t(x^*). \quad (8.1)$$

Using this metric, an algorithm performs well if its regret is *sublinear* as a function of  $T$ , i.e.  $\text{regret} = o(T)$ , because this implies that the algorithm performs as well as the best fixed strategy in hindsight on the average.

**Dynamic regret (competitive difference).** Although the classic regret is the predominant metric used in previous work, it cannot fully capture the dynamic nature of  $f_{1:T}$ . In particular,  $\min_{x^* \in \mathcal{K}} \sum_{t=1}^T f_t(x^*)$  could be a weak benchmark if the minimizer of  $f_t$  is highly time-variant. For example, for robotic control applications,  $f_t$  depends on some time-variant environmental conditions (e.g., the wind condition in Chapter 5), and then the optimal policy at each time step is also time-variant.

Therefore, we also consider an alternative metric called dynamic regret (also known as competitive difference):

$$\text{dynamic regret} = \sum_{t=1}^T f_t(x_t) - \sum_{t=1}^T \min_{x \in \mathcal{K}} f_t(x) \quad (8.2)$$

which is the difference between the online algorithm cost and the offline optimal cost with the full knowledge of  $f_{1:T}$ . Intuitively, pursuing sublinear convergence is too good to be true using such a strong metric, so typically the goal is to bound the dynamic regret as a function of *path length*, which measures the variation of  $f_{1:T}$  (e.g., Y. Li et al. (2019)).

**Competitive ratio.** As mentioned before, dynamic regret analysis typically offers *problem-dependent* bounds (i.e., depending on the path length). Therefore, we also consider an even stronger variant called competitive ratio:

$$\text{competitive ratio} = \frac{\sum_{t=1}^T f_t(x_t)}{\sum_{t=1}^T \min_{x \in \mathcal{K}} f_t(x)} \quad (8.3)$$

which studies the ratio between the online algorithm cost and the offline optimal cost. Typically the theoretical goal is to have competitive ratios bounded by a *constant* (e.g., Lin et al. (2021)).

### Online Gradient Descent (OGD) and the Regret Analysis

In this subsection, we introduce a natural and canonical algorithm in online optimization, Online Gradient Descent (OGD), and analyze its regret.

The OGD algorithm runs as follows: it initializes  $x_1 \in \mathcal{K}$ . At time step  $t$ , it plays  $x_t$ , observes the cost function  $f_t$ , and then updates  $x_{t+1}$  by

$$x_{t+1} = \Pi_{\mathcal{K}}(x_t - \eta_t \nabla f_t(x_t))$$

where  $\Pi_{\mathcal{K}}$  is the projection onto  $\mathcal{K}$ , i.e.,  $\Pi_{\mathcal{K}}(y) = \arg \min_{x \in \mathcal{K}} \|x - y\|$ . Namely, OGD makes the decision by a projected gradient descent step with a time-variant learning rate  $\eta_t$ . We have the following well-known guarantee:

**Lemma 8.1** (Regret of OGD). *Suppose  $f_{1:T}(x)$  is a sequence of differentiable convex cost functions from  $\mathbb{R}^n$  to  $\mathbb{R}$ , and  $\mathcal{K}$  is a convex set in  $\mathbb{R}^n$  with diameter  $D$ , i.e.,  $\forall x_1, x_2 \in \mathcal{K}, \|x_1 - x_2\| \leq D$ . We denote by  $G > 0$  an upper bound on the norm of the gradients of  $f_{1:T}$  over  $\mathcal{K}$ , i.e.,  $\|\nabla f_t(x)\| \leq G$  for all  $t \in [1, T]$  and  $x \in \mathcal{K}$ . Then OGD with learning rates  $\{\eta_t = \frac{D}{G\sqrt{t}}\}$  guarantees the following:*

$$\sum_{t=1}^T f_t(x_t) - \min_{x^* \in \mathcal{K}} \sum_{t=1}^T f_t(x^*) \leq \frac{3}{2}GD\sqrt{T}. \quad (8.4)$$

The proof can be found in the Chapter 3.1 of Hazan (2019).

**Lower bounds.** Perhaps surprisingly, the OGD algorithm attains a tight regret bound up to a small constant factor in the worst case. Formally speaking, one can construct a counterexample to prove that any online algorithm for online optimization with convex cost functions incurs a  $GD\sqrt{T}$  regret in the worst case.

**Logarithmic regret with strongly convex functions.** Even though the  $\sqrt{T}$  dependence is unavoidable, it is possible to have much better regret bounds if the cost function has strong curvature, namely, being strongly convex. One can show that the same OGD algorithm with a different learning rate scheduling can achieve a logarithmic regret  $O(\log T)$  when  $f_{1:T}$  is strongly convex.

### 8.4 Preliminaries on Online Optimal Control

Similar to online learning and optimization, in online optimal control, an online policy (or agent) iteratively makes actions, and the goal is to minimize a sequence of cost functions. However, the main difference and extra challenge is from the *underlying dynamics*. Namely, in online optimal control, the policy can only decide



the action  $u_t$  instead of the state. The state  $x_{t+1}$  is a (deterministic or stochastic) function of  $x_t$  and  $u_t$ . Instead, in online optimization the player can directly select  $x_t$ . Therefore, generally speaking, non-asymptotic optimality guarantees (e.g., regret, dynamic regret, competitive ratio) in online optimal control are much more challenging to obtain, and most existing optimal control theoretical results focus on stability, robustness, and asymptotic convergence rather than those learning-theoretic non-asymptotic guarantees.

In this section, we briefly introduce two commonly studied optimal control problems: LQR and MPC. For more details, we refer to Anderson and Moore (2007), Kirk (2004), and Camacho and Alba (2013).

### Linear Quadratic Regulator (LQR)

Here we consider a generalized stochastic version of the classic LQR problem. In particular, we consider a linear system initialized with  $x_0 \in \mathbb{R}^n$  and controlled by  $u_t \in \mathbb{R}^m$ , with dynamics

$$x_{t+1} = Ax_t + Bu_t + w_t \quad \text{and cost} \quad J = \sum_{t=0}^{T-1} (x_t^\top Qx_t + u_t^\top Ru_t) + x_T^\top Q_f x_T \quad (8.5)$$

where  $T \geq 1$  is the total length of the control period,  $Q_f$  is the terminal cost matrix, and  $w_{0:T-1}$  are i.i.d. with  $\mathbb{E}[w_t] = 0$ ,  $\mathbb{E}[w_t w_t^\top] = W$ . The goal of the controller is to minimize the expectation of the cost  $J$  given  $A, B, Q, R, Q_f, x_0$  and  $W$ , i.e.,  $\min \mathbb{E}_{w_{0:T-1}} [J]$ .

In the stochastic LQR problem, the controller choose  $u_t$  *after* knowing the current state  $x_t$  but *before* knowing the current disturbance  $w_t$ . Or equivalently, at time step  $t$  the controller knows  $x_0$  and  $w_{0:t-1}$ . Therefore, at time step  $t$ , we need to solve an optimal policy as a function of  $x_0$  and  $w_{0:t-1}$ , namely,  $u_t^*(x_0, w_{0:t-1})$ .

Now let us solve Eq. (8.5) via dynamic programming. Define  $V_t(z)$  as the optimal value function (or cost-to-go) from time step  $t$  starting at  $x_t = z$ . Namely:

$$V_t(z) = \min_{u_{t:T-1}} \mathbb{E} \left[ \sum_{\tau=t}^{T-1} (x_\tau^\top Qx_\tau + u_\tau^\top Ru_\tau) + x_T^\top Q_f x_T \right].$$

Obviously we have  $V_T(z) = z^\top Q_f z$ , and the optimal total cost is  $V_0(x_0)$ . Moreover, via dynamic programming,  $V_t$  can be found by the following backward recursion

(for  $t = T - 1, \dots, 0$ ):

$$V_t(z) = z^\top Qz + \min_u \{u^\top Ru + \mathbb{E}_{w_t} [V_{t+1}(Az + Bu + w_t)]\}.$$

After solving  $V_{0:T}$ , the optimal policy at time step  $t$  becomes

$$u_t^* = \arg \min_u \{u^\top Ru + \mathbb{E}_{w_t} [V_{t+1}(Az + Bu + w_t)]\}.$$

Because the dynamics is linear and the cost is quadratic, one can show that the value function  $V_t$  has a nice quadratic form  $V_t(z) = z^\top P_t x_t + q_t$  for  $t = 0, \dots, T$ . And we have the following recursion:

$$\begin{aligned} P_T &= Q_f, q_T = 0 \\ P_t &= Q + A^\top P_{t+1} A - A^\top P B (R + B^\top P_{t+1} B)^{-1} B^\top P_{t+1} A \\ q_t &= q_{t+1} + \text{Tr}(W P_{t+1}). \end{aligned}$$

Moreover, the optimal policy  $u_t$  is in a linear state feedback form:  $u_t^* = -K_t x_t$  where

$$K_t = (R + B^\top P_{t+1} B)^{-1} B^\top P_{t+1} A.$$

Finally, the optimal cost is

$$V_0(x_0) = x_0^\top P_0 x_0 + q_0 = x_0^\top P_0 x_0 + \sum_{t=1}^T \text{Tr}(W P_t).$$

Some remarks:

- In classic (deterministic) LQR,  $W = 0$ , so we have the same  $P_t, K_t$  while  $q_t = 0, \forall t$ . Moreover, the optimal cost is  $x_0^\top P_0 x_0$ .
- Due to the structure of the LQR system, the optimal policy  $u_t^*$  has a nice *linear feedback form* and it is *independent* of  $x_0$  and  $W$ .
- **Infinite horizon.** When  $T \rightarrow \infty$ , we choose to minimize an average cost, i.e.,

$$\min \lim_{T \rightarrow \infty} \frac{1}{T} \mathbb{E}_{w_{0:T-1}} \left[ \sum_{t=0}^{T-1} (x_t^\top Q x_t + u_t^\top R u_t) + x_T^\top Q_f x_T \right].$$

In this case, the optimal average cost is given by  $\text{Tr}(WP)$ , where  $P$  satisfies

$$P = Q + A^\top P A - A^\top P B (R + B^\top P B)^{-1} B^\top P A. \quad (8.6)$$

Equation (8.6) is called discrete algebraic Riccati equation (DARE). Moreover, the optimal policy is a constant linear feedback policy  $u_t^* = -Kx_t$  where  $K = (R + B^T P B)^{-1} B^T P A$ .

- **Predictions and delays.** Note that in the standard LQR setting, at time step  $t$  the policy knows  $x_0$  and  $w_{0:t-1}$ , i.e., knowing the current state  $x_t$  but not knowing the current disturbance  $w_t$ . In Chapter 11, we will generalize it significantly to settings with predictions (knowing future  $w$ ) and delays.
- **I.i.d. disturbance.** The simple structure of the value function  $V_t$  and the policy  $u = -K_t x$  also relies on the fact that the disturbance  $w$  is i.i.d. and zero-mean. In Chapter 11, we will also generalize it significantly to general stochastic disturbance and adversarial disturbance.

### Model Predictive Control (MPC)

Since the 1980s, Model Predictive Control (MPC) has been one of the most influential and popular process control methods in industries. The key idea of MPC is straightforward: with a finite look-ahead window of the future, MPC optimizes a finite-time optimal control problem at each time step, but only implements/executes the current time slot and then optimizes again at the next time step, repeatedly. The second part “only implements the current time slot and reoptimizes at each time step” is one of the reasons MPC was not that popular before the 1980s, because iteratively solving complex optimal control problems at high frequency was such a luxury task before computational power took off.

In this subsection, we use a trajectory tracking problem to explain how MPC works (visualized in Fig. 8.2).

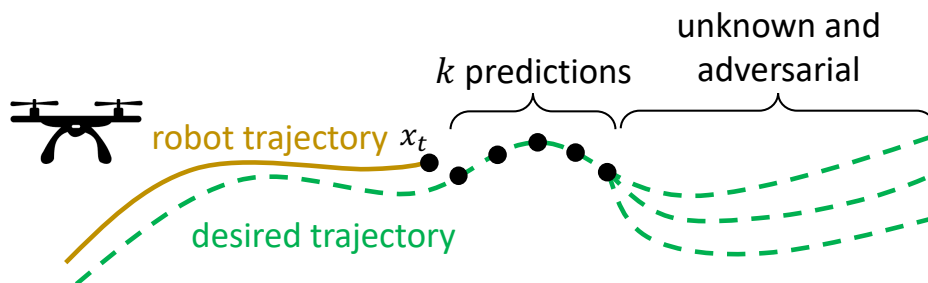


Figure 8.2: A trajectory tracking problem to explain how MPC works.

Suppose a robot is tracking a trajectory  $d_1, \dots, d_T$ , and the robot’s dynamics follows  $x_{t+1} = f_t(x_t, u_t) + w_t, 0 \leq t \leq T - 1$  where  $x_t, u_t, w_t$  are state, control input

and disturbance, respectively. At time step  $t$ , the cost the robot needs to pay is  $(x_t - d_t)^\top Q(x_t - d_t) + u_t^\top R u_t$  where  $Q$  penalizes the tracking error and  $R$  penalizes the fuel cost (control energy). At each time step, the robot has  $k$ -step future predictions:  $d_{t+1:t+k}$  (trajectory),  $f_{t:t+k-1}$  (dynamics) and  $w_{t:t+k-1}$  (disturbance).

At each time step, MPC solves the following  $k$ -step optimal control problem:

$$\begin{aligned} \min_{u_{t:t+k-1}} \quad & \sum_{i=1}^k ((x_{t+i} - d_{t+i})^\top Q(x_{t+i} - d_{t+i}) + u_{t+i-1}^\top R u_{t+i-1}) + V(x_{t+k}) \\ \text{s.t.} \quad & x_{t+i+1} = f_{t+i}(x_{t+i}, u_{t+i}) + w_{t+i}, \quad 0 \leq i \leq k-1 \\ & u_{t+i} \in \mathcal{U}, x_{t+i+1} \in \mathcal{X} \quad 0 \leq i \leq k-1 \end{aligned}$$

where the terminal cost  $V(\cdot)$  regularizes the terminal state. Having a proper  $V$  is critical for the stability and performance of MPC. Suppose the optimal solution from the above optimization problem is  $u_{t|t}^*, \dots, u_{t+k-1|t}^*$ . A key feature of MPC is that only the first solved action  $u_{t|t}^*$  is executed/used, and at the next step we need to solve another optimization problem for  $u_{t+1|t+1}^*$ .

### Why Beyond Regret in Online Control?

We finish this section by discussing why we prefer metrics beyond regret in Part II.

In Part II, we focus on non-asymptotic analyses of online control and learning problems. In Chapters 10 and 11, we use global optimality metrics beyond (static) regret, namely, dynamic regret and competitive ratio, as introduced in Section 8.3.

The reader may wonder why we choose those two metrics rather than the classic regret, especially considering that no-regret online control has become an extremely popular research area since 2011. However, we will use the following example to show that we should consider metrics beyond regret in *dynamic environments*.

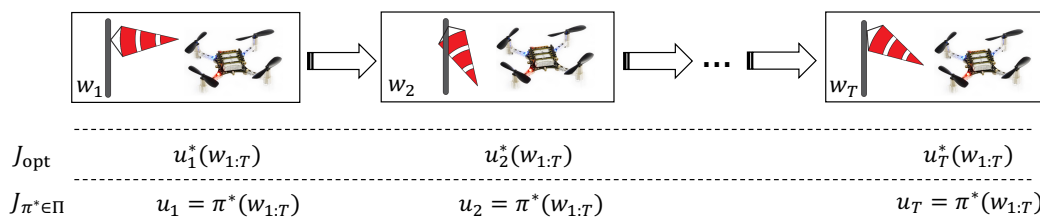


Figure 8.3: An example illustrating why beyond regret.

Figure 8.3 revisits the drone-flying-in-wind example in Chapter 5. At every time step, the drone experiences a new wind condition  $w_t$ . The first row in Fig. 8.3 is the benchmark we use in dynamic regret or competitive ratio, which compares

against the globally offline optimal policy's cost  $J_{\text{OPT}}$  given the full sequence  $w_{1:T}$  in hindsight. Note that the globally offline optimal policy might be highly nonlinear and adaptive.

On the other hands, the second row in Fig. 8.3 is the benchmark we use in classic regret, which compares against a fixed and stationary offline optimal policy  $\pi^*$ . Note that  $\pi^*$  is in some known and fixed policy class  $\Pi$ . The predominant  $\Pi$  used in previous work is the linear controller class (e.g., Agarwal et al. (2019), Dean et al. (2020), and Cohen et al. (2018)).

However, the cost of the optimal static policy  $J_{\pi^* \in \Pi}$  might be far from  $J_{\text{OPT}}$ . In Chapter 10, we will show that this gap could be arbitrarily large even in simple 1-d systems. Therefore, achieving small regret may still mean having a significantly larger cost than optimal.

Another reason we choose metrics beyond regret is that, it is in general intractable to design or characterize a reasonable comparator policy class  $\Pi$  in dynamic environments (i.e., time-variant systems). For example, MPC is exactly such a *dynamic* algorithm. If we use regret to analyze MPC, it is extremely hard (if not impossible) to define a reasonable comparator class  $\Pi$ .

## References

- Abbasi-Yadkori, Yasin and Csaba Szepesvári (2011). *Regret bounds for the adaptive control of linear quadratic systems*. In: *Proceedings of the 24th Annual Conference on Learning Theory*. JMLR Workshop and Conference Proceedings, pp. 1–26.
- Agarwal, Naman, Brian Bullins, Elad Hazan, Sham M. Kakade, and Karan Singh (2019). *Online control with adversarial disturbances*. In: *International Conference on Machine Learning (ICML)*.
- Anderson, Brian D. O. and John B. Moore (2007). *Optimal control: Linear quadratic methods*. Courier Corporation.
- Åström, Karl J. and Björn Wittenmark (2013). *Adaptive control*. Courier Corporation.
- Camacho, Eduardo F. and Carlos Bordons Alba (2013). *Model predictive control*. Springer Science & Business Media.
- Cohen, Alon, Avinatan Hasidim, Tomer Koren, Nevena Lazic, Yishay Mansour, and Kunal Talwar (2018). *Online linear quadratic control*. In: *International Conference on Machine Learning*. Proceedings of Machine Learning Research, pp. 1029–1038.

- Dean, Sarah, Horia Mania, Nikolai Matni, Benjamin Recht, and Stephen Tu (2020). *On the sample complexity of the linear quadratic regulator*. In: *Foundations of Computational Mathematics* 20.4, pp. 633–679.
- Hazan, Elad (2019). *Introduction to online convex optimization*. In: *arXiv preprint arXiv:1909.05207*.
- Kirk, Donald E. (2004). *Optimal control theory: An introduction*. Courier Corporation.
- Krstic, Miroslav, Petar V. Kokotovic, and Ioannis Kanellakopoulos (1995). *Nonlinear and adaptive control design*. John Wiley & Sons, Inc.
- Li, Yingying, Xin Chen, and Na Li (2019). *Online optimal control with linear dynamics and predictions: Algorithms and regret analysis*. In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 14858–14870.
- Lin, Yiheng, Yang Hu, Guanya Shi, Haoyuan Sun, Guannan Qu, and Adam Wierman (2021). *Perturbation-based regret analysis of predictive control in linear time varying systems*. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 34. Curran Associates, Inc., pp. 5174–5185. URL: <https://arxiv.org/abs/2106.10497>.
- Rawlings, James B. (2000). *Tutorial overview of model predictive control*. In: *IEEE Control Systems Magazine* 20.3, pp. 38–52.
- Richards, Spencer M., Navid Azizan, Jean-Jacques E. Slotine, and Marco Pavone (2021). *Adaptive-control-oriented meta-learning for nonlinear systems*. In: *arXiv preprint arXiv:2103.04490*.
- Slotine, Jean-Jacques E. and Weiping Li (1991). *Applied nonlinear control*. Vol. 199. 1. Prentice Hall, Englewood Cliffs, NJ.

## ONLINE META-ADAPTIVE CONTROL

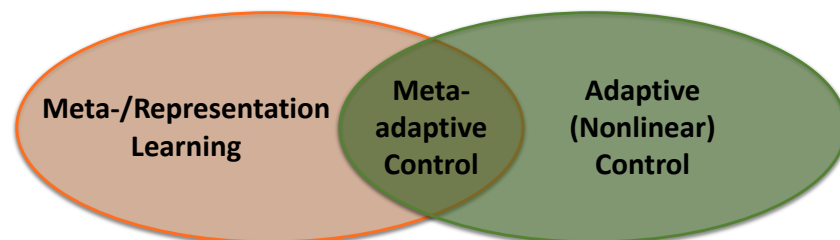


Figure 9.1: The motivation of Chapter 9.

In order to have rapidly adaptable autonomous systems operating in changing environments (e.g., varying wind conditions for drones in Chapter 5), it is crucial to extract common representations from all environments. However, as shown in Fig. 9.1, existing theoretical results focus on either representation/meta-learning with i.i.d. data (i.e., no dynamics) or adaptive control in a single environment. In this chapter, we propose a novel *meta-adaptive control* framework, where meta-learning optimizes a representation shared by all environments. Then the representation is fine-tuned by adaptive nonlinear control in a low-dimensional space. Chapter 9 jointly analyzes outer-loop learning and inner-loop adaptive control under a unified framework, and provides the first end-to-end non-asymptotic guarantee for multi-task nonlinear control. This chapter is mainly based on the following paper<sup>1</sup>:

Shi, Guanya, Kamyar Azizzadenesheli, Michael O’Connell, Soon-Jo Chung, and Yisong Yue (2021). *Meta-adaptive nonlinear control: Theory and algorithms*. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 34. Curran Associates, Inc., pp. 10013–10025. URL: <https://proceedings.neurips.cc/paper/2021/file/52fc2aee802efbad698503d28ebd3a1f-Paper.pdf>.

---

**Abstract.** We present an online multi-task learning approach for adaptive nonlinear control, which we call Online Meta-Adaptive Control (OMAC). The goal is to con-

<sup>1</sup>Code and video: <https://github.com/GuanyaShi/Online-Meta-Adaptive-Control>

control a nonlinear system subject to adversarial disturbance and unknown *environment-dependent* nonlinear dynamics, under the assumption that the environment-dependent dynamics can be well captured with some shared representation. Our approach is motivated by robot control, where a robotic system encounters a sequence of new environmental conditions that it must quickly adapt to. A key emphasis is to integrate online representation learning with established methods from control theory, in order to arrive at a unified framework that yields both control-theoretic and learning-theoretic guarantees. We provide instantiations of our approach under varying conditions, leading to the first non-asymptotic end-to-end convergence guarantee for multi-task nonlinear control. OMAC can also be integrated with deep representation learning. Experiments show that OMAC significantly outperforms conventional adaptive control approaches which do not learn the shared representation, in inverted pendulum and 6-DoF drone control tasks under varying wind conditions.

## 9.1 Introduction

One important goal in autonomy and artificial intelligence is to enable autonomous robots to learn from prior experience to quickly adapt to new tasks and environments. Examples abound in robotics, such as a drone flying in different wind conditions (Chapter 5), a manipulator throwing varying objects (Zeng et al., 2020), or a quadruped walking over changing terrains (Lee et al., 2020). Though those examples provide encouraging empirical evidence, when designing such adaptive systems, two important theoretical challenges arise, as discussed below.

First, from a learning perspective, the system should be able to learn an “efficient” representation from prior tasks, thereby permitting faster future adaptation, which falls into the categories of representation learning or meta-learning. Recently, a line of work has shown theoretically that learning representations (in the standard supervised setting) can significantly reduce sample complexity on new tasks (Du et al., 2020; Tripuraneni, Jin, et al., 2020; Maurer et al., 2016). Empirically, deep representation learning or meta-learning has achieved success in many applications (Bengio et al., 2013), including control, in the context of meta-reinforcement learning (Gupta et al., 2018; Finn, Abbeel, et al., 2017; Nagabandi et al., 2018). However, theoretical benefits (in the end-to-end sense) of representation learning or meta-learning for adaptive control remain unclear.

Second, from a control perspective, the agent should be able to handle parametric



model uncertainties with control-theoretic guarantees such as stability and tracking error convergence, which is a common adaptive control problem (Slotine and W. Li, 1991; Karl J. Åström and Wittenmark, 2013). For classic adaptive control algorithms, theoretical analysis often involves the use of Lyapunov stability and asymptotic convergence (Slotine and W. Li, 1991; Karl J. Åström and Wittenmark, 2013). Moreover, many recent studies aim to integrate ideas from learning, optimization, and control theory to design and analyze adaptive controllers using learning-theoretic metrics. Typical results guarantee non-asymptotic convergence in finite time horizons, such as regret (Boffi et al., 2021; Simchowitz and Foster, 2020; Dean et al., 2020; Kakade et al., 2020) and dynamic regret (Chapter 11 and Y. Li et al. (2019)). However, these results focus on a single environment or task. A multi-task extension, especially whether and how prior experience could benefit the adaptation in new tasks, remains an open problem.

**Main contributions.** In this chapter, we address both learning and control challenges in a unified framework and provide end-to-end guarantees. We derive a new method of Online Meta-Adaptive Control (OMAC) that controls uncertain nonlinear systems under a sequence of new environmental conditions. The underlying assumption is that the environment-dependent unknown dynamics can well be captured by a shared representation, which OMAC learns using a *meta-adapter*. OMAC then performs environment-specific updates using an *inner-adapter*.

We provide different instantiations of OMAC under varying assumptions and conditions. In the jointly and element-wise convex cases, we show sublinear cumulative control error bounds, which to our knowledge is the first non-asymptotic convergence result for multi-task nonlinear control. Compared to standard adaptive control approaches that do not have a meta-adapter, we show that OMAC possesses both stronger guarantees and empirical performance. We finally show how to integrate OMAC with deep representation learning, which further improves empirical performance.

## 9.2 Problem Statement

We consider the setting where a controller encounters a sequence of  $N$  environments, with each environment lasting  $T$  time steps. We use *outer iteration* to refer to the iterating over the  $N$  environments, and *inner iteration* to refer to the  $T$  time steps within an environment. We use superscripts (e.g.,  $(i)$  in  $x_t^{(i)}$ ) to denote the index of the outer iteration where  $1 \leq i \leq N$ , and subscripts (e.g.,  $t$  in  $x_t^{(i)}$ ) to denote the time

index of the inner iteration where  $1 \leq t \leq T$ . We use  $\text{step}(i, t)$  to refer to the inner time step  $t$  at the  $i^{\text{th}}$  outer iteration.

We consider a discrete-time nonlinear control-affine system (Murray et al., 2017; LaValle, 2006) with environment-dependent uncertainty  $f(x, c)$ . The dynamics model at the  $i^{\text{th}}$  outer iteration is:

$$x_{t+1}^{(i)} = f_0(x_t^{(i)}) + B(x_t^{(i)})u_t^{(i)} - f(x_t^{(i)}, c^{(i)}) + w_t^{(i)}, \quad 1 \leq t \leq T, \quad (9.1)$$

where the state  $x_t^{(i)} \in \mathbb{R}^n$ , the control  $u_t^{(i)} \in \mathbb{R}^m$ ,  $f_0 : \mathbb{R}^n \rightarrow \mathbb{R}^n$  is a known nominal dynamics model,  $B : \mathbb{R}^n \rightarrow \mathbb{R}^{n \times m}$  is a known state-dependent actuation matrix,  $c^{(i)} \in \mathbb{R}^h$  is the unknown parameter that indicates an environmental condition,  $f : \mathbb{R}^n \times \mathbb{R}^h \rightarrow \mathbb{R}^n$  is the unknown  $c^{(i)}$ -dependent dynamics model, and  $w_t^{(i)}$  is disturbance, potentially adversarial. For simplicity we define  $B_t^{(i)} = B(x_t^{(i)})$  and  $f_t^{(i)} = f(x_t^{(i)}, c^{(i)})$ .

**Interaction protocol.** We study the following adaptive nonlinear control problem under  $N$  unknown environments. At the beginning of outer iteration  $i$ , the environment first selects  $c^{(i)}$  (adaptively and adversarially), which is unknown to the controller, and then the controller makes decision  $u_{1:T}^{(i)}$  under unknown dynamics  $f(x_t^{(i)}, c^{(i)})$  and potentially adversarial disturbances  $w_t^{(i)}$ . To summarize:

1. **Outer iteration  $i$ .** A policy encounters environment  $i$  ( $i \in \{1, \dots, N\}$ ), associated with unobserved variable  $c^{(i)}$  (e.g., the wind condition for a flying drone). Run inner loop (Step 2).
2. **Inner loop.** Policy interacts with environment  $i$  for  $T$  time steps, observing  $x_t^{(i)}$  and taking action  $u_t^{(i)}$ , with state/action dynamics following (9.1).
3. Policy optionally observes  $c^{(i)}$  at the end of the inner loop (used for some variants of the analysis).
4. Increment  $i = i + 1$  and repeat from Step 1.

We use average control error (ACE) as our performance metric:

**Definition 9.1** (Average control error). *The average control error (ACE) of  $N$  outer iterations (i.e.,  $N$  environments) with each lasting  $T$  time steps, is defined as  $\text{ACE} = \frac{1}{TN} \sum_{i=1}^N \sum_{t=1}^T \|x_t^{(i)}\|$ .*

ACE can be viewed as a non-asymptotic generalization of steady-state error in control (Karl Johan Åström and Murray, 2021). We make the following assumptions on the actuation matrix  $B$ , the nominal dynamics  $f_0$ , and disturbance  $w_t^{(i)}$ :

**Assumption 9.1** (Full actuation, bounded disturbance, and e-ISS assumptions). *We consider fully-actuated systems, i.e., for all  $x$ ,  $\text{rank}(B(x)) = n$ .  $\|w_t^{(i)}\| \leq W, \forall t, i$ . Moreover, the nominal dynamics  $f_0$  is exponentially input-to-state stable (e-ISS): let constants  $\beta, \gamma \geq 0$  and  $0 \leq \rho < 1$ . For a sequence  $v_{1:t-1} \in \mathbb{R}^n$ , consider the dynamics  $x_{k+1} = f_0(x_k) + v_k, 1 \leq k \leq t-1$ .  $x_t$  satisfies:*

$$\|x_t\| \leq \beta \rho^{t-1} \|x_1\| + \gamma \sum_{k=1}^{t-1} \rho^{t-1-k} \|v_k\|. \quad (9.2)$$

With the e-ISS property in Assumption 9.1, we have the following bound that connects ACE with the average squared loss between  $B_t^{(i)} u_t^{(i)} + w_t^{(i)}$  and  $f_t^{(i)}$ .

**Lemma 9.1** (Connecting learning error with control error). *Assume  $x_1^{(i)} = 0, \forall i$ . The average control error (ACE) is bounded as:*

$$\frac{\sum_{i=1}^N \sum_{t=1}^T \|x_t^{(i)}\|}{TN} \leq \frac{\gamma}{1-\rho} \sqrt{\frac{\sum_{i=1}^N \sum_{t=1}^T \|B_t^{(i)} u_t^{(i)} - f_t^{(i)} + w_t^{(i)}\|^2}{TN}}. \quad (9.3)$$

*Proof.* Using the e-ISS property in Assumption 9.1, we have:

$$\begin{aligned} \frac{1}{TN} \sum_{i=1}^N \sum_{t=1}^T \|x_t^{(i)}\| &\leq \frac{1}{TN} \sum_{i=1}^N \sum_{t=1}^T \left( \gamma \sum_{k=1}^{t-1} \rho^{t-1-k} \|B_k^{(i)} u_k^{(i)} - f_k^{(i)} + w_k^{(i)}\| \right) \\ &\stackrel{(a)}{\leq} \frac{\gamma}{1-\rho} \frac{1}{TN} \sum_{i=1}^N \sum_{t=1}^{T-1} \|B_t^{(i)} u_t^{(i)} - f_t^{(i)} + w_t^{(i)}\| \\ &\stackrel{(b)}{\leq} \frac{\gamma}{1-\rho} \sqrt{\frac{1}{TN}} \sqrt{\sum_{i=1}^N \sum_{t=1}^T \|B_t^{(i)} u_t^{(i)} - f_t^{(i)} + w_t^{(i)}\|^2}, \end{aligned} \quad (9.4)$$

where (a) and (b) are from geometric series and Cauchy-Schwarz inequality, respectively.  $\square$

In Lemma 9.1, we assume  $x_1^{(i)} = 0$  for simplicity: the influence of non-zero and bounded  $x_1^{(i)}$  is a constant term in each outer iteration, from the e-ISS property (9.2).

**Remark on the e-ISS assumption and the ACE metric.** Note that an exponentially stable linear system  $f_0(x_t) = Ax_t$  (i.e., the spectral radius of  $A$  is  $< 1$ ) satisfies

the exponential ISS (e-ISS) assumption. However, in nonlinear systems e-ISS is a stronger assumption than exponential stability. For both linear and nonlinear systems, the e-ISS property of  $f_0$  is usually achieved by applying some stable feedback controller to the system<sup>2</sup>, i.e.,  $f_0$  is the closed-loop dynamics (Slotine and W. Li, 1991; Cohen et al., 2018). e-ISS assumption is standard in both online adaptive linear control (Cohen et al., 2018; Simchowicz and Foster, 2020) and nonlinear control (Boffi et al., 2021), and practical in robotic control such as drones (see Chapter 2). In ACE, we consider a regulation task, but it can also capture trajectory tracking task with time-variant nominal dynamics  $f_0$  under incremental stability assumptions (Boffi et al., 2021). We only consider the regulation task in this chapter for simplicity.

**Generality.** We would like to emphasize the generality of our dynamics model (9.1). The nominal control-affine part can model general fully-actuated robotic systems via Euler-Langrange equations (Murray et al., 2017; LaValle, 2006), and the unknown part  $f(x, c)$  is nonlinear in  $x$  and  $c$ . We only need to assume the disturbance  $w_t^{(i)}$  is bounded, which is more general than stochastic settings in linear (Simchowicz and Foster, 2020; Dean et al., 2020) and nonlinear (Boffi et al., 2021) cases. For example,  $w_t^{(i)}$  can model extra  $(x, u, c)$ -dependent uncertainties or adversarial disturbances. Moreover, the environment sequence  $c^{(1:N)}$  could also be adversarial. In term of the extension to under-actuated systems, all the results in this chapter hold for the *matched uncertainty* setting, i.e., in the form  $x_{t+1}^{(i)} = f_0(x_t^{(i)}) + B(x_t^{(i)})(u_t^{(i)} - f(x_t^{(i)}, c^{(i)})) + w_t^{(i)}$  where  $B(x_t^{(i)})$  is not necessarily full rank (e.g., drone and inverted pendulum experiments in Section 9.5). Generalizing to other under-actuated systems is interesting future work.

### 9.3 Online Meta-Adaptive Control (OMAC) Algorithm

The design of our online meta-adaptive control (OMAC) approach comprises four pieces: the policy class, the function class, the inner loop (within environment) adaptation algorithm  $\mathcal{A}_2$ , and the outer loop (between environment) online learning algorithm  $\mathcal{A}_1$ .

**Policy class.** We focus on the class of certainty-equivalence controllers (Boffi et al., 2021; Mania et al., 2019; Simchowicz and Foster, 2020), which is a general class of model-based controllers that also includes linear feedback controllers commonly studied in online control (Mania et al., 2019; Simchowicz and Foster, 2020). After

<sup>2</sup>For example, consider  $x_{t+1} = \frac{3}{2}x_t + 2 \sin x_t + \bar{u}_t$ . With a feedback controller  $\bar{u}_t = u_t - x_t - 2 \sin x_t$ , the closed-loop dynamics is  $x_{t+1} = \frac{1}{2}x_t + u_t$ , where  $f_0(x) = \frac{1}{2}x$  is e-ISS.

a model is learned from past data, a controller is designed by treating the learned model as the truth (Karl J. Åström and Wittenmark, 2013). Formally, at  $\text{step}(i, t)$ , the controller first estimates  $\hat{f}_t^{(i)}$  (an estimation of  $f_t^{(i)}$ ) based on past data, and then executes  $u_t^{(i)} = B_t^{(i)\dagger} \hat{f}_t^{(i)}$ , where  $(\cdot)^\dagger$  is the pseudo inverse. Note that from Lemma 9.1, the average control error of the *omniscient controller* using  $\hat{f}_t^{(i)} = f_t^{(i)}$  (i.e., the controller perfectly knows  $f(x, c)$ ) is upper bounded as<sup>3</sup>:

$$\text{ACE}(\text{omniscient}) \leq \gamma W / (1 - \rho).$$

ACE(omniscient) can be viewed as a fundamental limit of the certainty-equivalence policy class.

**Function class  $F$  for representation learning.** In OMAC, we need to define a function class  $F(\phi(x; \hat{\Theta}), \hat{c})$  to compute  $\hat{f}_t^{(i)}$  (i.e.,  $\hat{f}_t^{(i)} = F(\phi(x_t^{(i)}; \hat{\Theta}), \hat{c})$ ), where  $\phi$  (parameterized by  $\hat{\Theta}$ ) is a representation shared by all environments, and  $\hat{c}$  is an environment-specific latent state. From a theoretical perspective, the main consideration of the choice of  $F(\phi(x), \hat{c})$  is on how it effects the resulting learning objective. For instance,  $\phi$  represented by a Deep Neural Network (DNN) would lead to a highly non-convex learning objective. In this chapter, we focus on the setting  $\hat{\Theta} \in \mathbb{R}^p, \hat{c} \in \mathbb{R}^h$ , and  $p \gg h$ , i.e., it is much more expensive to learn the shared representation  $\phi$  (e.g., a DNN) than “fine-tuning” via  $\hat{c}$  in a specific environment, which is consistent with meta-learning (Finn, Abbeel, et al., 2017) and representation learning (Du et al., 2020; Bengio et al., 2013) practices.

**Inner loop adaptive control.** We take a modular approach in our algorithm design, in order to cleanly leverage state-of-the-art methods from online learning, representation learning, and adaptive control. When interacting with a single environment (for  $T$  time steps), we keep the learned representation  $\phi$  fixed, and use that representation for adaptive control by treating  $\hat{c}$  as an unknown low-dimensional parameter. We can utilize any adaptive control method such as online gradient descent, velocity gradient, or composite adaptation (Hazan, 2019; Boffi et al., 2021; Slotine and W. Li, 1991).

**Outer loop online learning.** We treat the outer loop (which iterates between environments) as an online learning problem, where the goal is to learn the shared representation  $\phi$  that optimizes the inner loop adaptation performance. Theoretic-

<sup>3</sup>This upper bound is tight. Consider a scalar system  $x_{t+1} = ax_t + u_t - f(x_t) + w$  with  $|a| < 1$  and  $w$  a constant. In this case  $\rho = |a|, \gamma = 1$ , and the omniscient controller  $u_t = f(x_t)$  yields  $\text{ACE} = \gamma|w|/(1 - \rho)$ .

cally, we can reason about the analysis by setting up a hierarchical or nested online learning procedure (adaptive control nested within online learning).

**Design goal.** Our goal is to design a meta-adaptive controller that has low ACE, ideally converging to ACE(omniscient) as  $T, N \rightarrow \infty$ . In other words, OMAC should converge to performing as good as the omniscient controller with perfect knowledge of  $f(x, c)$ .

---

**Algorithm 9.1:** Online Meta-Adaptive Control (OMAC) algorithm

---

**Input:** Meta-adapter  $\mathcal{A}_1$ ; inner-adapter  $\mathcal{A}_2$ ; model  $F(\phi(x; \hat{\Theta}), \hat{c})$ ; Boolean ObserveEnv

```

1 for  $i = 1, \dots, N$  do
2   The environment selects  $c^{(i)}$ 
3   for  $t = 1, \dots, T$  do
4     Compute  $\hat{f}_t^{(i)} = F(\phi(x_t^{(i)}; \hat{\Theta}^{(i)}), \hat{c}_t^{(i)})$ 
5     Execute  $u_t^{(i)} = B_t^{(i)\dagger} \hat{f}_t^{(i)}$  //certainty-equivalence policy
6     Observe  $x_{t+1}^{(i)}, y_t^{(i)} = f(x_t^{(i)}, c^{(i)}) - w_t^{(i)}$ , and
        $\ell_t^{(i)}(\hat{\Theta}, \hat{c}) = \|F(\phi(x_t^{(i)}; \hat{\Theta}), \hat{c}) - y_t^{(i)}\|^2$  // $y_t^{(i)}$  is a noisy
       measurement of  $f$  and  $\ell_t^{(i)}$  is the observed loss
7     Construct an inner cost function  $g_t^{(i)}(\hat{c})$  by  $\mathcal{A}_2$  // $g_t^{(i)}$  is a function of  $\hat{c}$ 
8     Inner-adaptation:  $\hat{c}_{t+1}^{(i)} \leftarrow \mathcal{A}_2(\hat{c}_t^{(i)}, g_{1:t}^{(i)})$ 
9     if ObserveEnv then Observe  $c^{(i)}$  //only used in some instantiations
10    Construct an outer cost function  $G^{(i)}(\hat{\Theta})$  by  $\mathcal{A}_1$  // $G^{(i)}$  is a function of  $\hat{\Theta}$ 
11    Meta-adaptation:  $\hat{\Theta}^{(i+1)} \leftarrow \mathcal{A}_1(\hat{\Theta}^{(i)}, G^{(1:i)})$ 

```

---

Algorithm 9.1 describes the OMAC algorithm. Since  $\phi$  is environment-invariant and  $p \gg h$ , we only adapt  $\hat{\Theta}$  at the end of each outer iteration. On the other hand, because  $c^{(i)}$  varies in different environments, we adapt  $\hat{c}$  at each inner step. As shown in Algorithm 9.1, at  $\text{step}(i, t)$ , after applying  $u_t^{(i)}$ , the controller observes the next state  $x_{t+1}^{(i)}$  and computes:  $y_t^{(i)} \triangleq f_0(x_t^{(i)}) + B_t^{(i)} u_t^{(i)} - x_{t+1}^{(i)} = f_t^{(i)} - w_t^{(i)}$ , which is a disturbed measurement of the ground truth  $f_t^{(i)}$ . We then define  $\ell_t^{(i)}(\hat{\Theta}, \hat{c}) \triangleq \|F(\phi(x_t^{(i)}; \hat{\Theta}), \hat{c}) - y_t^{(i)}\|^2$  as the observed loss at  $\text{step}(i, t)$ , which is a squared loss between the disturbed measurement of  $f_t^{(i)}$  and the model prediction  $F(\phi(x_t^{(i)}; \hat{\Theta}), \hat{c})$ .

**Instantiations.** Depending on  $\{F(\phi(x; \hat{\Theta}), \hat{c}), \mathcal{A}_1, \mathcal{A}_2, \text{ObserveEnv}\}$ , we consider three settings:

- **Convex case:** The observed loss  $\ell_t^{(i)}$  is convex with respect to  $\hat{\Theta}$  and  $\hat{c}$ .

- **Element-wise convex case:** Fixing  $\hat{\Theta}$  or  $\hat{c}$ ,  $\ell_t^{(i)}$  is convex with respect to the other.
- **Deep learning case:** We use a DNN with weight  $\hat{\Theta}$  to represent  $\phi$ .

#### 9.4 Different Instantiations of OMAC and Theoretical Analysis

##### Convex Case

In this subsection, we focus on a setting where the observed loss  $\ell_t^{(i)}(\hat{\Theta}, \hat{c}) = \|F(\phi(x; \hat{\Theta}), \hat{c}) - y_t^{(i)}\|^2$  is convex with respect to  $\hat{\Theta}$  and  $\hat{c}$ . We provide the following example to illustrate this case and highlight its difference between conventional adaptive control (e.g., Boffi et al. (2021) and Karl J. Åström and Wittenmark (2013)).

**Example 9.1.** We consider a model  $F(\phi(x; \hat{\Theta}), \hat{c}) = Y_1(x)\hat{\Theta} + Y_2(x)\hat{c}$  to estimate  $f$ :

$$\hat{f}_t^{(i)} = Y_1(x_t^{(i)})\hat{\Theta}^{(i)} + Y_2(x_t^{(i)})\hat{c}_t^{(i)}, \quad (9.5)$$

where  $Y_1 : \mathbb{R}^n \rightarrow \mathbb{R}^{n \times p}$ ,  $Y_2 : \mathbb{R}^n \rightarrow \mathbb{R}^{n \times h}$  are two known bases. Note that conventional adaptive control approaches typically concatenate  $\hat{\Theta}$  and  $\hat{c}$  and adapt on both at each time step, regardless of the environment changes (e.g., Boffi et al. (2021)). Since  $p \gg h$ , such concatenation is computationally much more expensive than OMAC, which only adapts  $\hat{\Theta}$  in outer iterations.

Because  $\ell_t^{(i)}(\hat{\Theta}, \hat{c})$  is jointly convex with respect to  $\hat{\Theta}$  and  $\hat{c}$ , the OMAC algorithm in this case falls into the category of Nested Online Convex Optimization (Nested OCO) (Agarwal et al., 2021). The choice of  $g_t^{(i)}$ ,  $G^{(i)}$ ,  $\mathcal{A}_1$ ,  $\mathcal{A}_2$  and `ObserveEnv` are depicted in Table 9.1. Note that in the convex case OMAC does not need to know  $c^{(i)}$  in the whole process (`ObserveEnv` = False).

$F(\phi(x; \hat{\Theta}), \hat{c})$	Any $F$ model such that $\ell_t^{(i)}(\hat{\Theta}, \hat{c})$ is convex (e.g., Example 9.1)
$g_t^{(i)}(\hat{c})$	$\nabla_{\hat{c}} \ell_t^{(i)}(\hat{\Theta}^{(i)}, \hat{c}_t^{(i)}) \cdot \hat{c}$
$G^{(i)}(\hat{\Theta})$	$\sum_{t=1}^T \nabla_{\hat{\Theta}} \ell_t^{(i)}(\hat{\Theta}^{(i)}, \hat{c}_t^{(i)}) \cdot \hat{\Theta}$
$\mathcal{A}_1$	With a convex set $\mathcal{K}_1$ , $\mathcal{A}_1$ initializes $\hat{\Theta}^{(1)} \in \mathcal{K}_1$ and returns $\hat{\Theta}^{(i+1)} \in \mathcal{K}_1, \forall i$ . $\mathcal{A}_1$ has sublinear regret, i.e., the total regret of $\mathcal{A}_1$ is $T \cdot o(N)$ (e.g., online gradient descent)
$\mathcal{A}_2$	With a convex set $\mathcal{K}_2$ , $\forall i$ , $\mathcal{A}_2$ initializes $\hat{c}_1^{(i)} \in \mathcal{K}_2$ and returns $\hat{c}_{t+1}^{(i)} \in \mathcal{K}_2, \forall t$ . $\mathcal{A}_2$ has sublinear regret, i.e., the total regret of $\mathcal{A}_2$ is $N \cdot o(T)$ (e.g., online gradient descent)
<code>ObserveEnv</code>	False

Table 9.1: OMAC with convex loss.

As shown in Table 9.1, at the end of step( $i, t$ ) we fix  $\hat{\Theta} = \hat{\Theta}^{(i)}$  and update  $\hat{c}_{t+1}^{(i)} \in \mathcal{K}_2$  using  $\mathcal{A}_2(\hat{c}_t^{(i)}, g_{1:t}^{(i)})$ , which is an OCO problem with linear costs  $g_{1:t}^{(i)}$ . On the other hand, at the end of outer iteration  $i$ , we update  $\hat{\Theta}^{(i+1)} \in \mathcal{K}_1$  using  $\mathcal{A}_1(\hat{\Theta}^{(i)}, G^{(1:i)})$ , which is another OCO problem with linear costs  $G^{(1:i)}$ . From Agarwal et al. (2021), we have the following regret relationship:

**Lemma 9.2** (Nested OCO regret bound, Agarwal et al. (2021)). *OMAC (Algorithm 9.1) specified by Table 9.1 has regret:*

$$\begin{aligned} & \sum_{i=1}^N \sum_{t=1}^T \ell_t^{(i)}(\hat{\Theta}^{(i)}, \hat{c}_t^{(i)}) - \min_{\Theta \in \mathcal{K}_1} \sum_{i=1}^N \min_{c^{(i)} \in \mathcal{K}_2} \sum_{t=1}^T \ell_t^{(i)}(\Theta, c^{(i)}) \\ & \leq \underbrace{\sum_{i=1}^N G^{(i)}(\hat{\Theta}^{(i)}) - \min_{\Theta \in \mathcal{K}_1} \sum_{i=1}^N G^{(i)}(\Theta)}_{\text{the total regret of } \mathcal{A}_1, T \cdot o(N)} + \underbrace{\sum_{i=1}^N \sum_{t=1}^T g_t^{(i)}(\hat{c}_t^{(i)}) - \sum_{i=1}^N \min_{c^{(i)} \in \mathcal{K}_2} \sum_{t=1}^T g_t^{(i)}(c^{(i)})}_{\text{the total regret of } \mathcal{A}_2, N \cdot o(T)}. \end{aligned} \quad (9.6)$$

Note that the total regret of  $\mathcal{A}_1$  is  $T \cdot o(N)$  because  $G^{(i)}$  is scaled up by a factor of  $T$ . With Lemmas 9.1 and 9.2, we have the following guarantee for the average control error.

**Theorem 9.1** (OMAC ACE bound with convex loss). *Assume the unknown dynamics model is  $f(x, c) = F(\phi(x; \Theta), c)$ . Assume the true parameters  $\Theta \in \mathcal{K}_1$  and  $c^{(i)} \in \mathcal{K}_2, \forall i$ . Then OMAC (Algorithm 9.1) specified by Table 9.1 ensures the following ACE guarantee:*

$$\text{ACE} \leq \frac{\gamma}{1-\rho} \sqrt{W^2 + \frac{o(T)}{T} + \frac{o(N)}{N}}.$$

*Proof.* Since  $\Theta \in \mathcal{K}_1$  and  $c^{(1:N)} \in \mathcal{K}_2$ , applying Lemma 9.2 we have

$$\sum_{i=1}^N \sum_{t=1}^T \ell_t^{(i)}(\hat{\Theta}^{(i)}, \hat{c}_t^{(i)}) - \sum_{i=1}^N \sum_{t=1}^T \ell_t^{(i)}(\Theta, c^{(i)}) \leq T \cdot o(N) + N \cdot o(T). \quad (9.7)$$

Recall that the definition of  $\ell_t^{(i)}$  is  $\ell_t^{(i)}(\hat{\Theta}, \hat{c}) = \|F(\phi(x_t^{(i)}; \hat{\Theta}), \hat{c}) - y_t^{(i)}\|^2$ , and  $y_t^{(i)} = f_t^{(i)} - w_t^{(i)}$ . Therefore we have

$$\begin{aligned} \ell_t^{(i)}(\hat{\Theta}^{(i)}, \hat{c}_t^{(i)}) &= \|\hat{f}_t^{(i)} - f_t^{(i)} + w_t^{(i)}\|^2 = \|B_t^{(i)} u_t^{(i)} - f_t^{(i)} + w_t^{(i)}\|^2 \\ \ell_t^{(i)}(\Theta, c^{(i)}) &= \|w_t^{(i)}\|^2 \leq W^2. \end{aligned} \quad (9.8)$$



Then applying Lemma 9.1, we have

$$\begin{aligned}
\text{ACE} &\leq \frac{\gamma}{1-\rho} \sqrt{\frac{\sum_{i=1}^N \sum_{t=1}^T \|B_t^{(i)} u_t^{(i)} - f_t^{(i)} + w_t^{(i)}\|^2}{TN}} \\
&= \frac{\gamma}{1-\rho} \sqrt{\frac{\sum_{i=1}^N \sum_{t=1}^T \ell_t^{(i)}(\hat{\Theta}^{(i)}, \hat{c}_t^{(i)})}{TN}} \\
&\stackrel{(a)}{\leq} \frac{\gamma}{1-\rho} \sqrt{\frac{T \cdot o(N) + N \cdot o(T) + \sum_{i=1}^N \sum_{t=1}^T \ell_t^{(i)}(\Theta, c^{(i)})}{TN}} \\
&\leq \frac{\gamma}{1-\rho} \sqrt{W^2 + \frac{o(T)}{T} + \frac{o(N)}{N}},
\end{aligned} \tag{9.9}$$

where (a) uses (9.7).  $\square$

To further understand Theorem 9.1 and compare OMAC with conventional adaptive control approaches, we provide the following corollary using the model in Example 9.1.

**Corollary 9.1.** *Suppose the unknown dynamics model is  $f(x, c) = Y_1(x)\Theta + Y_2(x)c$ , where  $Y_1 : \mathbb{R}^n \rightarrow \mathbb{R}^{n \times p}$ ,  $Y_2 : \mathbb{R}^n \rightarrow \mathbb{R}^{n \times h}$  are known bases. We assume  $\|\Theta\| \leq K_\Theta$  and  $\|c^{(i)}\| \leq K_c, \forall i$ . Moreover, we assume  $\|Y_1(x)\| \leq K_1$  and  $\|Y_2(x)\| \leq K_2, \forall x$ . In Table 9.1 we use  $\hat{f}_t^{(i)} = Y_1(x_t^{(i)})\hat{\Theta}^{(i)} + Y_2(x_t^{(i)})\hat{c}_t^{(i)}$ , and Online Gradient Descent (OGD) (Hazan, 2019) for both  $\mathcal{A}_1$  and  $\mathcal{A}_2$ , with learning rates  $\bar{\eta}^{(i)}$  and  $\eta_t^{(i)}$ , respectively. We set  $\mathcal{K}_1 = \{\hat{\Theta} : \|\hat{\Theta}\| \leq K_\Theta\}$  and  $\mathcal{K}_2 = \{\hat{c} : \|\hat{c}\| \leq K_c\}$ . If we schedule the learning rates as:*

$$\bar{\eta}^{(i)} = \frac{2K_\Theta}{\underbrace{(4K_1^2 K_\Theta + 4K_1 K_2 K_c + 2K_1 W)T\sqrt{i}}_{C_1}}, \quad \eta_t^{(i)} = \frac{2K_c}{\underbrace{(4K_2^2 K_c + 4K_1 K_2 K_\Theta + 2K_2 W)\sqrt{t}}_{C_2}},$$

then the average control performance is bounded as:

$$\text{ACE(OMAC)} \leq \frac{\gamma}{1-\rho} \sqrt{W^2 + 3 \left( K_\Theta C_1 \frac{1}{\sqrt{N}} + K_c C_2 \frac{1}{\sqrt{T}} \right)}.$$

Moreover, the baseline adaptive control which uses OGD to adapt  $\hat{\Theta}$  and  $\hat{c}$  at each time step satisfies:

$$\text{ACE(baseline adaptive control)} \leq \frac{\gamma}{1-\rho} \sqrt{W^2 + 3 \sqrt{K_\Theta^2 + K_c^2} \sqrt{C_1^2 + C_2^2} \frac{1}{\sqrt{T}}}.$$

*Proof.* Define  $\mathcal{R}(\mathcal{A}_1)$  as the total regret of the outer-adapter  $\mathcal{A}_1$ , and  $\mathcal{R}(\mathcal{A}_2)$  as the total regret of the inner-adapter  $\mathcal{A}_2$ . Recall that in Theorem 9.1 we show that  $\text{ACE}(\text{OMAC}) \leq \frac{\gamma}{1-\rho} \sqrt{W^2 + \frac{\mathcal{R}(\mathcal{A}_1) + \mathcal{R}(\mathcal{A}_2)}{TN}}$ . Now we will prove Corollary 9.1 by analyzing  $\mathcal{R}(\mathcal{A}_1)$  and  $\mathcal{R}(\mathcal{A}_2)$ , respectively.

Since the true dynamics  $f(x, c^{(i)}) = Y_1(x)\Theta + Y_2(x)c^{(i)}$ , we have

$$\ell_t^{(i)}(\hat{\Theta}, \hat{c}) = \|Y_1(x_t^{(i)})\hat{\Theta} + Y_2(x_t^{(i)})\hat{c} - Y_1(x_t^{(i)})\Theta - Y_2(x_t^{(i)})c^{(i)} + w_t^{(i)}\|^2. \quad (9.10)$$

Recall that  $g_t^{(i)}(\hat{c}) = \nabla_{\hat{c}} \ell_t^{(i)}(\hat{\Theta}^{(i)}, \hat{c}_t^{(i)}) \cdot \hat{c}$ , which is convex (linear) w.r.t.  $\hat{c}$ . The gradient of  $g_t^{(i)}$  is upper bounded as

$$\begin{aligned} \|\nabla_{\hat{c}} g_t^{(i)}\| &= \left\| 2Y_2(x_t^{(i)})^\top \left( Y_1(x_t^{(i)})\hat{\Theta}^{(i)} + Y_2(x_t^{(i)})\hat{c}_t^{(i)} - Y_1(x_t^{(i)})\Theta - Y_2(x_t^{(i)})c^{(i)} + w_t^{(i)} \right) \right\| \\ &\leq 2K_2K_1K_\Theta + 2K_2^2K_c + 2K_2K_1K_\Theta + 2K_2^2K_c + 2K_2W \\ &= \underbrace{4K_1K_2K_\Theta + 4K_2^2K_c + 2K_2W}_{C_2}. \end{aligned} \quad (9.11)$$

From Lemma 8.1, using learning rates  $\eta_t^{(i)} = \frac{2K_c}{C_2\sqrt{t}}$  for all  $i$ , the regret of  $\mathcal{A}_2$  at each outer iteration is upper bounded by  $3K_cC_2\sqrt{T}$ . Then the total regret of  $\mathcal{A}_2$  is bounded as

$$\mathcal{R}(\mathcal{A}_2) \leq 3K_cC_2N\sqrt{T}. \quad (9.12)$$

Now let us study  $\mathcal{A}_1$ . Similarly, recall that  $G^{(i)}(\hat{\Theta}) = \sum_{t=1}^T \nabla_{\hat{\Theta}} \ell_t^{(i)}(\hat{\Theta}^{(i)}, \hat{c}_t^{(i)}) \cdot \hat{\Theta}$ , which is convex (linear) w.r.t.  $\hat{\Theta}$ . The gradient of  $G^{(i)}$  is upper bounded as

$$\begin{aligned} \|\nabla_{\hat{\Theta}} G^{(i)}\| &= \left\| \sum_{t=1}^T 2Y_1(x_t^{(i)})^\top \left( Y_1(x_t^{(i)})\hat{\Theta}^{(i)} + Y_2(x_t^{(i)})\hat{c}_t^{(i)} - Y_1(x_t^{(i)})\Theta - Y_2(x_t^{(i)})c^{(i)} + w_t^{(i)} \right) \right\| \\ &\leq T \left( 2K_1^2K_\Theta + 2K_1K_2K_c + 2K_1^2K_\Theta + 2K_1K_2K_c + 2K_1W \right) \\ &= T \underbrace{(4K_1^2K_\Theta + 4K_1K_2K_c + 2K_1W)}_{C_1}. \end{aligned} \quad (9.13)$$

From Lemma 8.1, using learning rates  $\bar{\eta}^{(i)} = \frac{2K_\Theta}{TC_1\sqrt{i}}$ , the total regret of  $\mathcal{A}_1$  is upper bounded as

$$\mathcal{R}(\mathcal{A}_1) \leq 3K_\Theta TC_1\sqrt{N}. \quad (9.14)$$

Finally using Theorem 9.1 we have

$$\begin{aligned} \text{ACE}(\text{OMAC}) &\leq \frac{\gamma}{1-\rho} \sqrt{W^2 + \frac{\mathcal{R}(\mathcal{A}_1) + \mathcal{R}(\mathcal{A}_2)}{TN}} \\ &\leq \frac{\gamma}{1-\rho} \sqrt{W^2 + 3(K_\Theta C_1 \frac{1}{\sqrt{N}} + K_c C_2 \frac{1}{\sqrt{T}})}. \end{aligned} \quad (9.15)$$

Now let us analyze ACE(baseline adaptive control). To simplify notations, we define  $\bar{Y}(x) = [Y_1(x) \ Y_2(x)] : \mathbb{R}^n \rightarrow \mathbb{R}^{n \times (p+h)}$  and  $\hat{\alpha} = [\hat{\Theta}; \hat{c}] \in \mathbb{R}^{p+h}$ . The baseline adaptive controller updates the whole vector  $\hat{\alpha}$  at every time step. We denote the ground truth parameter by  $\alpha^{(i)} = [\Theta; c^{(i)}]$ , and the estimation by  $\hat{\alpha}_t^{(i)} = [\hat{\Theta}_t^{(i)}; \hat{c}_t^{(i)}]$ . We have  $\|\alpha^{(i)}\| \leq \sqrt{K_\Theta^2 + K_c^2}$ . Define  $\bar{\mathcal{K}} = \{\hat{\alpha} = [\hat{\Theta}; \hat{c}] : \|\hat{\Theta}\| \leq \mathcal{K}_\Theta, \|\hat{c}\| \leq \mathcal{K}_c\}$ , which is a convex set in  $\mathbb{R}^{p+h}$ .

Note that the loss function for the baseline adaptive control is  $\bar{\ell}_t^{(i)}(\hat{\alpha}) = \|\bar{Y}(x_t^{(i)})\hat{\alpha} - Y_1(x_t^{(i)})\Theta - Y_2(x_t^{(i)})c^{(i)} + w_t^{(i)}\|^2$ . The gradient of  $\bar{\ell}_t^{(i)}$  is

$$\nabla_{\hat{\alpha}} \bar{\ell}_t^{(i)}(\hat{\alpha}) = 2 \begin{bmatrix} Y_1(x_t^{(i)})^\top \\ Y_2(x_t^{(i)})^\top \end{bmatrix} (Y_1(x_t^{(i)})\hat{\Theta} + Y_2(x_t^{(i)})\hat{c} - Y_1(x_t^{(i)})\Theta - Y_2(x_t^{(i)})c^{(i)} + w_t^{(i)}), \quad (9.16)$$

whose norm on  $\bar{\mathcal{K}}$  is bounded by

$$\sqrt{4(K_1^2 + K_2^2)(2K_1K_\Theta + 2K_2K_c + W)^2} = \sqrt{C_1^2 + C_2^2}. \quad (9.17)$$

Therefore, from Lemma 8.1, running OGD on  $\bar{\mathcal{K}}$  with learning rates  $\frac{2\sqrt{K_\Theta^2 + K_c^2}}{\sqrt{C_1^2 + C_2^2}\sqrt{t}}$  gives the following guarantee at each outer iteration:

$$\sum_{t=1}^T \bar{\ell}_t^{(i)}(\hat{\alpha}_t^{(i)}) - \bar{\ell}_t^{(i)}(\alpha^{(i)}) \leq 3\sqrt{K_\Theta^2 + K_c^2}\sqrt{C_1^2 + C_2^2}\sqrt{T}. \quad (9.18)$$

Finally, similar as (9.9) we have

$$\begin{aligned} \text{ACE}(\text{baseline adaptive control}) &\leq \frac{\gamma}{1-\rho} \sqrt{\frac{\sum_{i=1}^N \sum_{t=1}^T \bar{\ell}_t^{(i)}(\hat{\alpha}_t^{(i)})}{TN}} \\ &\leq \frac{\gamma}{1-\rho} \sqrt{\frac{\sum_{i=1}^N 3\sqrt{K_\Theta^2 + K_c^2}\sqrt{C_1^2 + C_2^2}\sqrt{T} + \sum_{i=1}^N \sum_{t=1}^T \bar{\ell}_t^{(i)}(\alpha^{(i)})}{TN}} \\ &\leq \frac{\gamma}{1-\rho} \sqrt{W^2 + 3\sqrt{K_\Theta^2 + K_c^2}\sqrt{C_1^2 + C_2^2}\frac{1}{\sqrt{T}}}. \end{aligned} \quad (9.19)$$

□

Note that ACE(baseline adaptive control) does not improve as  $N$  increases (i.e., encountering more environments has no benefit). If  $p \gg h$ , we potentially have  $K_1 \gg K_2$  and  $K_\Theta \gg K_c$ , so  $C_1 \gg C_2$ . Therefore, the ACE upper bound of OMAC is better than the baseline adaptation if  $N > T$ , which is consistent with recent representation learning results (Du et al., 2020; Tripuraneni, Jin, et al., 2020). It is also

worth noting that the baseline adaptation is much more computationally expensive, because it needs to adapt both  $\hat{\Theta}$  and  $\hat{c}$  at each time step. Intuitively, OMAC improves convergence because the meta-adapter  $\mathcal{A}_1$  approximates the environment-invariant part  $Y_1(x)\Theta$ , which makes the inner-adapter  $\mathcal{A}_2$  much more efficient.

### Element-wise Convex Case

In this subsection, we consider the setting where the loss function  $\ell_t^{(i)}(\hat{\Theta}, \hat{c})$  is element-wise convex with respect to  $\hat{\Theta}$  and  $\hat{c}$ , i.e., when one of  $\hat{\Theta}$  or  $\hat{c}$  is fixed,  $\ell_t^{(i)}$  is convex with respect to the other one. For instance,  $F$  could be function as depicted in Example 9.2. Outside the context of control, such bilinear models are also studied in representation learning (Du et al., 2020; Tripuraneni, Jin, et al., 2020) and factorization bandits (Wang et al., 2017; Kawale et al., 2015).

**Example 9.2.** We consider a model  $F(\phi(x; \hat{\Theta}), \hat{c}) = Y(x)\hat{\Theta}\hat{c}$  to estimate  $f$ :

$$\hat{f}_t^{(i)} = Y(x_t^{(i)})\hat{\Theta}^{(i)}\hat{c}_t^{(i)}, \quad (9.20)$$

where  $Y : \mathbb{R}^n \rightarrow \mathbb{R}^{n \times \bar{p}}$  is a known basis,  $\hat{\Theta}^{(i)} \in \mathbb{R}^{\bar{p} \times h}$ , and  $\hat{c}_t^{(i)} \in \mathbb{R}^h$ . Note that the dimensionality of  $\hat{\Theta}$  is  $p = \bar{p}h$ . Conventional adaptive control typically views  $\hat{\Theta}\hat{c}$  as a vector in  $\mathbb{R}^{\bar{p}}$  and adapts it at each time step regardless of the environment changes (Boffi et al., 2021).

Compared to the Convex Case, the element-wise convex case poses new challenges: i) the coupling between  $\hat{\Theta}$  and  $\hat{c}$  brings inherent non-identifiability issues; and ii) statistical learning guarantees typical need i.i.d. assumptions on  $c^{(i)}$  and  $x_t^{(i)}$  (Du et al., 2020; Tripuraneni, Jin, et al., 2020). These challenges are further amplified by the underlying unknown nonlinear system (9.1). Therefore in this section we set `ObserveEnv = True`, i.e., the controller has access to the true environmental condition  $c^{(i)}$  at the end of the  $i^{\text{th}}$  outer iteration, which is practical in many systems when  $c^{(i)}$  has a concrete physical meaning, e.g., drones with wind disturbances (Chapter 5).

The inputs to OMAC for the element-wise convex case are specified in Table 9.2. Compared to the convex case in Table 9.1, difference includes i) the cost functions  $g_t^{(i)}$  and  $G^{(i)}$  are convex, not necessarily linear; and ii) since `ObserveEnv = True`, in  $G^{(i)}$  we use the true environmental condition  $c^{(i)}$ . With inputs specified in Table 9.2, Algorithm 9.1 has ACE guarantees in the following theorem.

**Theorem 9.2** (OMAC ACE bound with element-wise convex loss). *Assume the unknown dynamics model is  $f(x, c) = F(\phi(x; \Theta), c)$ . Assume the true parameter*

$F(\phi(x; \hat{\Theta}), \hat{c})$	Any $F$ model such that $\ell_t^{(i)}(\hat{\Theta}, \hat{c})$ is element-wise convex (e.g., Example 9.2)
$g_t^{(i)}(\hat{c})$	$\ell_t^{(i)}(\hat{\Theta}^{(i)}, \hat{c})$
$G^{(i)}(\hat{\Theta})$	$\sum_{t=1}^T \ell_t^{(i)}(\hat{\Theta}, c^{(i)})$
$\mathcal{A}_1$ & $\mathcal{A}_2$	Same as Table 9.1
ObserveEnv	True

Table 9.2: OMAC with element-wise convex loss.

$\Theta \in \mathcal{K}_1$  and  $c^{(i)} \in \mathcal{K}_2, \forall i$ . Then OMAC (Algorithm 9.1) specified by Table 9.2 ensures the following ACE guarantee:

$$\text{ACE} \leq \frac{\gamma}{1-\rho} \sqrt{W^2 + \frac{o(T)}{T} + \frac{o(N)}{N}}.$$

*Proof.* For any  $\Theta \in \mathcal{K}_1$  and  $c^{(1:N)} \in \mathcal{K}_2$ , we have

$$\begin{aligned} & \sum_{i=1}^N \sum_{t=1}^T \ell_t^{(i)}(\hat{\Theta}^{(i)}, \hat{c}_t^{(i)}) - \sum_{i=1}^N \sum_{t=1}^T \ell_t^{(i)}(\Theta, c^{(i)}) \\ &= \sum_{i=1}^N \sum_{t=1}^T \left[ \ell_t^{(i)}(\hat{\Theta}^{(i)}, \hat{c}_t^{(i)}) - \ell_t^{(i)}(\hat{\Theta}^{(i)}, c^{(i)}) \right] + \sum_{i=1}^N \sum_{t=1}^T \left[ \ell_t^{(i)}(\hat{\Theta}^{(i)}, c^{(i)}) - \ell_t^{(i)}(\Theta, c^{(i)}) \right] \\ &= \underbrace{\sum_{i=1}^N \sum_{t=1}^T \left[ g_t^{(i)}(\hat{c}_t^{(i)}) - g_t^{(i)}(c^{(i)}) \right]}_{\leq o(T)} + \underbrace{\sum_{i=1}^N \left[ G^{(i)}(\hat{\Theta}^{(i)}) - G^{(i)}(\Theta) \right]}_{\leq T \cdot o(N)}. \end{aligned} \tag{9.21}$$

Then combining with Lemma 9.1 results in the ACE bound.  $\square$

### Faster convergence with sub-Gaussian and environment diversity assumptions.

Since the cost functions  $g_t^{(i)}$  and  $G^{(i)}$  in Table 9.2 are not necessarily strongly convex, the ACE upper bound in Theorem 9.2 is typically  $\frac{\gamma}{1-\rho} \sqrt{W^2 + O(1/\sqrt{T}) + O(1/\sqrt{N})}$  (e.g., using OGD for both  $\mathcal{A}_1$  and  $\mathcal{A}_2$ ). However, for the bilinear model in Example 9.2, it is possible to achieve faster convergence with extra sub-Gaussian and environment diversity assumptions.

With a bilinear model, the inputs to the OMAC algorithm are shown in Table 9.3. With extra assumptions on  $w_t^{(i)}$  and the environment, we have the following ACE guarantees.

**Theorem 9.3** (OMAC ACE bound with bilinear model). *Consider an unknown dynamics model  $f(x, c) = Y(x)\Theta c$  where  $Y : \mathbb{R}^n \rightarrow \mathbb{R}^{n \times \bar{p}}$  is a known basis and*

$F(\phi(x; \hat{\Theta}), \hat{c})$	The bilinear model in Example 9.2
$g_t^{(i)}(\hat{c})$	$\ell_t^{(i)}(\hat{\Theta}^{(i)}, \hat{c})$
$G^{(i)}(\hat{\Theta})$	$\lambda \ \hat{\Theta}\ _F^2 + \sum_{j=1}^i \sum_{t=1}^T \ell_t^{(j)}(\hat{\Theta}, c^{(j)})$ with some $\lambda > 0$
$\mathcal{A}_1$	$\hat{\Theta}^{(i+1)} = \arg \min_{\hat{\Theta}} G^{(i)}(\hat{\Theta})$ (i.e., Ridge regression)
$\mathcal{A}_2$	Same as Table 9.1
ObserveEnv	True

Table 9.3: OMAC with bilinear model.

$\Theta \in \mathbb{R}^{\bar{p} \times h}$ . Assume each component of the disturbance  $w_t^{(i)}$  is  $R$ -sub-Gaussian, the true parameters  $\|\Theta\|_F \leq K_\Theta$ ,  $\|c^{(i)}\| \leq K_c, \forall i$ , and  $\|Y(x)\|_F \leq K_Y, \forall x$ . Define  $Z_t^{(j)} = c^{(j)\top} \otimes Y(x_t^{(j)}) \in \mathbb{R}^{n \times \bar{p}h}$  and assume environment diversity:

$$\lambda_{\min} \left( \sum_{j=1}^i \sum_{t=1}^T Z_t^{(j)\top} Z_t^{(j)} \right) = \Omega(i).$$

Then OMAC (Algorithm 9.1) specified by Table 9.3 has the following ACE guarantee (with probability at least  $1 - \delta$ ):

$$\text{ACE} \leq \frac{\gamma}{1 - \rho} \sqrt{W^2 + \frac{o(T)}{T} + O\left(\frac{\log(NT/\delta) \log(N)}{N}\right)}. \quad (9.22)$$

If we relax the environment diversity condition to  $\Omega(\sqrt{i})$ , the last term becomes  $O\left(\frac{\log(NT/\delta)}{\sqrt{N}}\right)$ .

*Proof.* Note that in this case the available measurement of  $f$  at the end of the outer iteration  $i$  is:

$$y_t^{(j)} = Y(x_t^{(j)})\Theta c^{(j)} - w_t^{(j)}, \quad 1 \leq j \leq i, 1 \leq t \leq T. \quad (9.23)$$

Recall that the Ridge-regression estimation of  $\hat{\Theta}$  is given by

$$\begin{aligned} \hat{\Theta}^{(i+1)} &= \arg \min_{\hat{\Theta}} \lambda \|\hat{\Theta}\|_F^2 + \sum_{j=1}^i \sum_{t=1}^T \|Y(x_t^{(j)})\hat{\Theta}c^{(j)} - y_t^{(j)}\|^2 \\ &= \arg \min_{\hat{\Theta}} \lambda \|\hat{\Theta}\|_F^2 + \sum_{j=1}^i \sum_{t=1}^T \|Z_t^{(j)} \text{vec}(\hat{\Theta}) - y_t^{(j)}\|^2. \end{aligned} \quad (9.24)$$

Note that  $y_t^{(j)} = (c^{(j)\top} \otimes Y(x_t^{(j)})) \cdot \text{vec}(\Theta) - w_t^{(j)} = Z_t^{(j)} \text{vec}(\Theta) - w_t^{(j)}$ . Define  $V_i = \lambda I + \sum_{j=1}^i \sum_{t=1}^T Z_t^{(j)\top} Z_t^{(j)}$ . Then from the Theorem 2 of Abbasi-Yadkori, Pál, et al. (2011) we have

$$\|\text{vec}(\hat{\Theta}^{(i+1)} - \Theta)\|_{V_i} \leq R \sqrt{\bar{p}h \log\left(\frac{1 + iT \cdot nK_Y^2 K_c^2 / \lambda}{\delta}\right)} + \sqrt{\lambda} K_\Theta \quad (9.25)$$

for all  $i$  with probability at least  $1 - \delta$ . Note that the environment diversity condition implies:  $V_i > \Omega(i)I$ . Finally we have

$$\|\hat{\Theta}^{(i+1)} - \Theta\|_F^2 = \|\text{vec}(\hat{\Theta}^{(i+1)} - \Theta)\|^2 \leq O\left(\frac{1}{i}\right)O(\log(iT/\delta)) = O\left(\frac{\log(iT/\delta)}{i}\right). \quad (9.26)$$

Then with a fixed  $\hat{\Theta}^{(i+1)}$ , in outer iteration  $i + 1$  we have

$$g_t^{(i+1)}(\hat{c}) = \|Y(x_t^{(i+1)})\hat{\Theta}^{(i+1)}\hat{c} - Y(x_t^{(i+1)})\Theta c^{(i+1)} + w_t^{(i+1)}\|^2. \quad (9.27)$$

Since  $\mathcal{A}_2$  gives sublinear regret, we have

$$\begin{aligned} & \sum_{t=1}^T \|Y(x_t^{(i+1)})\hat{\Theta}^{(i+1)}\hat{c}_t^{(i+1)} - Y(x_t^{(i+1)})\Theta c^{(i+1)} + w_t^{(i+1)}\|^2 \\ & - \min_{\hat{c} \in \mathcal{K}_2} \sum_{t=1}^T \|Y(x_t^{(i+1)})\hat{\Theta}^{(i+1)}\hat{c} - Y(x_t^{(i+1)})\Theta c^{(i+1)} + w_t^{(i+1)}\|^2 = o(T). \end{aligned} \quad (9.28)$$

Note that

$$\begin{aligned} & \min_{\hat{c} \in \mathcal{K}_2} \sum_{t=1}^T \|Y(x_t^{(i+1)})\hat{\Theta}^{(i+1)}\hat{c} - Y(x_t^{(i+1)})\Theta c^{(i+1)} + w_t^{(i+1)}\|^2 \\ & \leq \sum_{t=1}^T \|Y(x_t^{(i+1)})\hat{\Theta}^{(i+1)}c^{(i+1)} - Y(x_t^{(i+1)})\Theta c^{(i+1)} + w_t^{(i+1)}\|^2 \\ & \stackrel{(a)}{\leq} TW^2 + T \cdot K_Y^2 \cdot O\left(\frac{\log(iT/\delta)}{i}\right) \cdot K_c^2, \end{aligned} \quad (9.29)$$

where (a) uses (9.26). Finally we have

$$\begin{aligned} & \sum_{t=1}^T \|\hat{f}_t^{(i+1)} - f_t^{(i+1)} + w_t^{(i+1)}\|^2 \\ & = \sum_{t=1}^T \|Y(x_t^{(i+1)})\hat{\Theta}^{(i+1)}\hat{c}_t^{(i+1)} - Y(x_t^{(i+1)})\Theta c^{(i+1)} + w_t^{(i+1)}\|^2 \\ & \stackrel{(b)}{\leq} o(T) + TW^2 + O\left(T \frac{\log(iT/\delta)}{i}\right) \end{aligned} \quad (9.30)$$

for all  $i$  with probability at least  $1 - \delta$ . (b) is from (9.28) and (9.29). Then with

Lemma 9.1 we have (with probability at least  $1 - \delta$ )

$$\begin{aligned}
\text{ACE} &\leq \frac{\gamma}{1 - \rho} \sqrt{\frac{\sum_{i=1}^N o(T) + TW^2 + O\left(T \frac{\log(iT/\delta)}{i}\right)}{TN}} \\
&\leq \frac{\gamma}{1 - \rho} \sqrt{W^2 + \frac{o(T)}{T} + \frac{O(\log(NT/\delta))}{N} \sum_{i=1}^N \frac{1}{i}} \quad (9.31) \\
&\leq \frac{\gamma}{1 - \rho} \sqrt{W^2 + \frac{o(T)}{T} + O\left(\frac{\log(NT/\delta) \log(N)}{N}\right)}.
\end{aligned}$$

If we relax the environment diversity condition to  $\Omega(\sqrt{i})$ , in (9.26) we will have  $O\left(\frac{\log(iT/\delta)}{\sqrt{i}}\right)$ . Therefore in (9.31) the last term becomes  $\frac{O(\log(NT/\delta))}{N} \sum_{i=1}^N \frac{1}{\sqrt{i}} \leq \frac{O(\log(NT/\delta))}{\sqrt{N}}$ .  $\square$

The sub-Gaussian assumption is widely used in statistical learning theory to obtain concentration bounds (Abbasi-Yadkori, Pál, et al., 2011; Du et al., 2020). The environment diversity assumption states that  $c^{(i)}$  provides “new information” in every outer iteration such that the minimum eigenvalue of  $\sum_{j=1}^i \sum_{t=1}^T Z_t^{(j)\top} Z_t^{(j)}$  grows linearly as  $i$  increases. Note that we do not need  $\lambda_{\min}(\sum_{j=1}^i \sum_{t=1}^T Z_t^{(j)\top} Z_t^{(j)})$  to increase as  $T$  goes up. Moreover, if we relax the condition to  $\Omega(\sqrt{i})$ , the ACE bound becomes worse than the general element-wise convex case (the last term is  $O(1/\sqrt{N})$ ), which implies the importance of “linear” environment diversity  $\Omega(i)$ . Task diversity has been shown to be important for representation learning (Du et al., 2020; Tripuraneni, Jordan, et al., 2020).

## Deep OMAC

We now introduce deep OMAC, a deep representation learning based OMAC algorithm. Table 9.4 shows the instantiation. As shown in Table 9.4, Deep OMAC employs a DNN to represent  $\phi$ , and uses gradient descent for optimization. With the model<sup>4</sup>,  $\phi(x; \hat{\Theta}) \cdot \hat{c}$ , the meta-adapter  $\mathcal{A}_1$  updates the representation  $\phi$  via deep learning, and the inner-adapter  $\mathcal{A}_2$  updates a linear layer  $\hat{c}$ .

## 9.5 Simulations

To demonstrate the performance of OMAC, we consider two sets of experiments:

- **Inverted pendulum with external wind, gravity mismatch, and unknown damping.** The continuous-time model is  $ml^2\ddot{\theta} - ml\hat{g} \sin \theta = u + f(\theta, \dot{\theta}, c)$ ,

<sup>4</sup>This structure generalizes the meta-learning and adaptive control method covered in Chapter 5



$F(\phi(x; \hat{\Theta}), \hat{c})$	$\phi(x; \hat{\Theta}) \cdot \hat{c}$ , where $\phi(x; \hat{\Theta}) : \mathbb{R}^n \rightarrow \mathbb{R}^{n \times h}$ is a neural network with weight $\hat{\Theta}$
$g_t^{(i)}(\hat{c})$	$\ell_t^{(i)}(\hat{\Theta}^{(i)}, \hat{c})$
$\mathcal{A}_1$	Gradient descent: $\hat{\Theta}^{(i+1)} \leftarrow \hat{\Theta}^{(i)} - \eta \nabla_{\hat{\Theta}} \sum_{t=1}^T \ell_t^{(i)}(\hat{\Theta}, \hat{c}_t^{(i)})$
$\mathcal{A}_2$	Same as Table 9.1
ObserveEnv	False

Table 9.4: OMAC with deep learning.

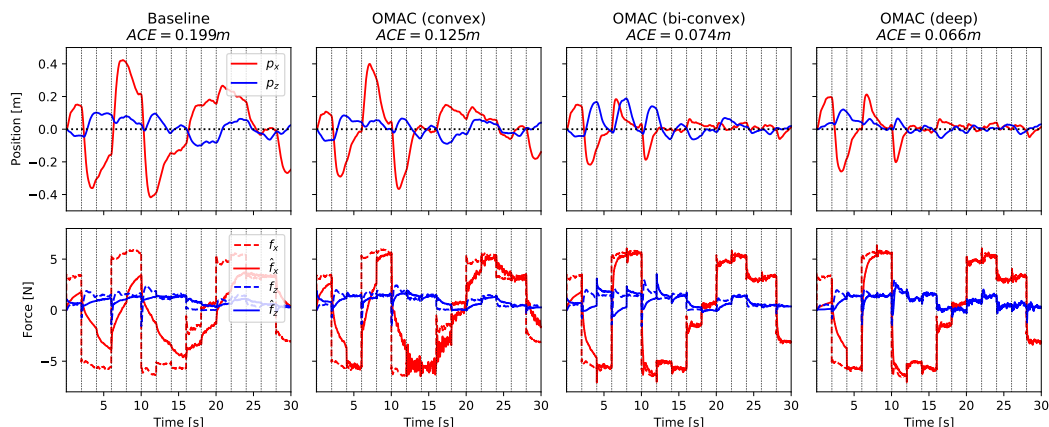
where  $\theta$  is the pendulum angle,  $\dot{\theta}/\ddot{\theta}$  is the angular velocity/acceleration,  $m$  is the pendulum mass and  $l$  is the length,  $\hat{g}$  is the gravity estimation,  $c$  is the unknown parameter that indicates the external wind condition, and  $f(\theta, \dot{\theta}, c)$  is the unknown dynamics which depends on  $c$ , but also includes  $c$ -invariant terms such as damping and gravity mismatch. This model generalizes Liu et al. (2020) by considering damping and gravity mismatch.

- **6-DoF quadrotor with 3-D wind disturbances.** We consider a 6-DoF quadrotor model with unknown wind-dependent aerodynamic force  $f(x, c) \in \mathbb{R}^3$ , where  $x \in \mathbb{R}^{13}$  is the drone state (including position, velocity, attitude, and angular velocity) and  $c$  is the unknown parameter indicating the wind condition. We incorporate a realistic high-fidelity aerodynamic model from Shi et al. (2020).

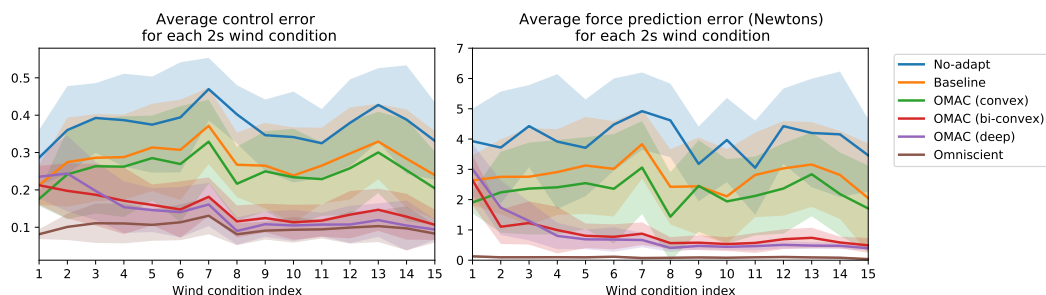
We consider 6 different controllers in the experiment:

- **No-adapt** is simply using  $\hat{f}_t^{(i)} = 0$ , and **omniscient** is using  $\hat{f}_t^{(i)} = f_t^{(i)}$ .
- **OMAC (convex)** is based on Example 9.1, where  $\hat{f}_t^{(i)} = Y_1(x_t^{(i)})\hat{\Theta}^{(i)} + Y_2(x_t^{(i)})\hat{c}_t^{(i)}$ . We use random Fourier features (Rahimi and Recht, 2007; Boffi et al., 2021) to generate both  $Y_1$  and  $Y_2$ . We use OGD for both  $\mathcal{A}_1$  and  $\mathcal{A}_2$  in Table 9.1.
- **OMAC (bi-convex)** is based on Example 9.2, where  $\hat{f}_t^{(i)} = Y(x_t^{(i)})\hat{\Theta}^{(i)}\hat{c}_t^{(i)}$ . Similarly, we use random Fourier features to generate  $Y$ . Although the theoretical result in the Element-wise Convex Case requires `ObserveEnv = True`, we relax this in our experiments and use  $G^{(i)}(\hat{\Theta}) = \sum_{t=1}^T \ell_t^{(i)}(\hat{\Theta}, \hat{c}_t^{(i)})$  in Table 9.2, instead of  $\sum_{t=1}^T \ell_t^{(i)}(\hat{\Theta}, c^{(i)})$ . We also deploy OGD for  $\mathcal{A}_1$  and  $\mathcal{A}_2$ . **Baseline** has the same procedure except with  $\hat{\Theta}^{(i)} \equiv \hat{\Theta}^{(1)}$ , i.e., it calls the inner-adapter  $\mathcal{A}_2$  at every step and does not update  $\hat{\Theta}$ , which is standard in adaptive control (Boffi et al., 2021; Karl J. Åström and Wittenmark, 2013).

- **OMAC (deep learning)** is based on Table 9.4. We use a DNN with spectral normalization (see details in Chapter 2) to represent  $\phi$ , and use the Adam optimizer (Kingma and Ba, 2014) for  $\mathcal{A}_1$ . Same as other methods,  $\mathcal{A}_2$  is also an OGD algorithm.



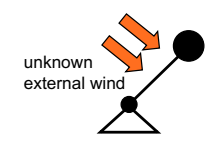
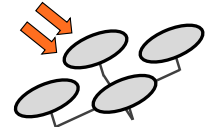
(a) xz-position trajectories (top) and force predictions (bottom) in the quadrotor experiment from one random seed. The wind condition is switched randomly every 2 s (indicated by the dotted vertical lines).



(b) The evolution of control error (left) and prediction error (right) in each wind condition. The solid lines average 10 random seeds and the shade areas show standard deviations. The performance OMAC improves as the number of wind conditions increases (especially the bi-convex and deep variants) while the baseline not.

Figure 9.2: Drone experiment results.

For all methods, we randomly switch the environment (wind)  $c$  every 2 s. To make a fair comparison, except no-adapt or omniscient, all methods have the same learning rate for the inner-adapter  $\mathcal{A}_2$  and the dimensions of  $\hat{c}$  are also same ( $\dim(\hat{c}) = 20$  for the pendulum and  $\dim(\hat{c}) = 30$  for the drone). ACE results from 10 random seeds are depicted in Table 9.5. Figure 9.2 visualizes the drone experiment results. We observe that i) OMAC significantly outperforms the baseline; ii) OMAC adapts faster and faster as it encounters more environments but baseline cannot benefit from prior experience, especially for the bi-convex and deep versions (see Figure

	no-adapt	baseline	OMAC (convex)
	$0.663 \pm 0.142$	$0.311 \pm 0.112$	$0.147 \pm 0.047$
	OMAC (bi-convex)	OMAC (deep)	omniscient
	$0.129 \pm 0.044$	$0.093 \pm 0.027$	$0.034 \pm 0.017$
	no-adapt	baseline	OMAC (convex)
	$0.374 \pm 0.044$	$0.283 \pm 0.043$	$0.251 \pm 0.043$
	OMAC (bi-convex)	OMAC (deep)	omniscient
	$0.150 \pm 0.019$	$0.141 \pm 0.024$	$0.100 \pm 0.018$

Table 9.5: ACE results in pendulum (top) and drone (bottom) experiments from 10 random seeds.

9.2), and iii) Deep OMAC achieves the best ACE due to the representation power of DNN.

We note that in the drone experiments the performance of OMAC (convex) is only marginally better than the baseline. This is because the aerodynamic disturbance force in the quadrotor simulation is a multiplicative combination of the relative wind speed, the drone attitude, and the motor speeds; thus, the superposition structure  $\hat{f}_t^{(i)} = Y_1(x_t^{(i)})\hat{\Theta}^{(i)} + Y_2(x_t^{(i)})\hat{C}_t^{(i)}$  cannot easily model the unknown force, while the bi-convex and deep learning variants both learn good controllers. In particular, OMAC (bi-convex) achieves similar performance as the deep learning case with much fewer parameters. On the other hand, in the pendulum experiments, OMAC (convex) is relatively better because a large component of the  $c$ -invariant part in the unknown dynamics is in superposition with the  $c$ -dependent part.

## 9.6 Related Work

**Meta-learning and representation learning.** Empirically, representation learning and meta-learning have shown great success in various domains (Bengio et al., 2013). In terms of control, meta-reinforcement learning is able to solve challenging multi-task RL problems (Gupta et al., 2018; Finn, Abbeel, et al., 2017; Nagabandi et al., 2018). We remark that learning representation for control also refers to learn *state* representation from rich observations (Lesort et al., 2018; Gelada et al., 2019; Zhang et al., 2019), but we consider *dynamics* representation in this chapter. On the theoretic side, Du et al. (2020), Tripuraneni, Jin, et al. (2020), and Tripuraneni, Jordan, et al. (2020) have shown representation learning reduces sample complexity on new tasks, and “task diversity” is critical. Consistently, we show OMAC enjoys better convergence theoretically (Corollary 9.1) and empirically, and Theorem 9.3 also implies the importance of *environment diversity*. Another relevant line of

theoretical work (Finn, Rajeswaran, et al., 2019; Denevi et al., 2019; Khodak et al., 2019) uses tools from online convex optimization to show guarantees for gradient-based meta-learning, by assuming closeness of all tasks to a single fixed point in parameter space. We eliminate this assumption by considering a hierarchical or nested online optimization procedure.

**Adaptive control and online control.** There is a rich body of literature studying Lyapunov stability and asymptotic convergence in adaptive control theory (Slotine and W. Li, 1991; Karl J. Åström and Wittenmark, 2013). Recently, there has been increased interest in studying online adaptive control with non-asymptotic metrics (e.g., regret) from learning theory, largely for settings with linear systems such as online LQR or LQG with unknown dynamics (Simchowitz and Foster, 2020; Dean et al., 2020; Cohen et al., 2018; Abbasi-Yadkori and Szepesvári, 2011). The most relevant work (Boffi et al., 2021) gives the first regret bound of adaptive nonlinear control with unknown nonlinear dynamics and stochastic noise. Another relevant work studies online robust control of nonlinear systems with a mistake guarantee on the number of robustness failures (Ho et al., 2021). However, all these results focus on the single-task case. To our knowledge, we show the first non-asymptotic convergence result for multi-task adaptive control. On the empirical side, Chapter 5 and Richards et al. (2021) combine adaptive nonlinear control with meta-learning, yielding encouraging experimental results.

**Online matrix factorization.** Our work bears affinity to online matrix factorization, particularly the bandit collaborative filtering setting (Kawale et al., 2015; Bresler et al., 2016; Wang et al., 2017). In this setting, one typically posits a linear low-rank projection as the target representation (e.g., a low-rank factorization of the user-item matrix), which is similar to our bilinear case. Setting aside the significant complexity introduced by nonlinear control, a key similarity comes from viewing different users as “tasks” and recommended items as “actions.” Prior work in this area has by and large not been able to establish strong regret bounds, in part due to the non-identifiability issue inherent in matrix factorization. In contrast, in our setting, one set of latent variables (e.g., the wind condition) has a concrete physical meaning that we are allowed to measure (ObserveEnv in Algorithm 9.1), thus side-stepping this non-identifiability issue.

## References

- Abbasi-Yadkori, Yasin, Dávid Pál, and Csaba Szepesvári (2011). *Improved algorithms for linear stochastic bandits*. In: *Neural Information Processing Systems*. Vol. 11, pp. 2312–2320.
- Abbasi-Yadkori, Yasin and Csaba Szepesvári (2011). *Regret bounds for the adaptive control of linear quadratic systems*. In: *Proceedings of the 24th Annual Conference on Learning Theory*. JMLR Workshop and Conference Proceedings, pp. 1–26.
- Agarwal, Naman, Elad Hazan, Anirudha Majumdar, and Karan Singh (2021). *A regret minimization approach to iterative learning control*. In: *arXiv preprint arXiv:2102.13478*.
- Åström, Karl J. and Björn Wittenmark (2013). *Adaptive control*. Courier Corporation.
- Åström, Karl Johan and Richard M. Murray (2021). *Feedback systems: An introduction for scientists and engineers*. Princeton University Press.
- Bengio, Yoshua, Aaron Courville, and Pascal Vincent (2013). *Representation learning: A review and new perspectives*. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8, pp. 1798–1828.
- Boffi, Nicholas M., Stephen Tu, and Jean-Jacques E. Slotine (2021). *Regret bounds for adaptive nonlinear control*. In: *Learning for Dynamics and Control*. Proceedings of Machine Learning Research, pp. 471–483.
- Bresler, Guy, Devavrat Shah, and Luis Filipe Voloch (2016). *Collaborative filtering with low regret*. In: *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science*.
- Cohen, Alon, Avinatan Hasidim, Tomer Koren, Nevena Lazic, Yishay Mansour, and Kunal Talwar (2018). *Online linear quadratic control*. In: *International Conference on Machine Learning*. Proceedings of Machine Learning Research, pp. 1029–1038.
- Dean, Sarah, Horia Mania, Nikolai Matni, Benjamin Recht, and Stephen Tu (2020). *On the sample complexity of the linear quadratic regulator*. In: *Foundations of Computational Mathematics* 20.4, pp. 633–679.
- Denevi, Giulia, Carlo Ciliberto, Riccardo Grazi, and Massimiliano Pontil (2019). *Learning-to-learn stochastic gradient descent with biased regularization*. In: *International Conference on Machine Learning*. Proceedings of Machine Learning Research, pp. 1566–1575.
- Du, Simon S., Wei Hu, Sham M. Kakade, Jason D. Lee, and Qi Lei (2020). *Few-shot learning via learning the representation, provably*. In: *arXiv preprint arXiv:2002.09434*.

- Finn, Chelsea, Pieter Abbeel, and Sergey Levine (2017). *Model-agnostic meta-learning for fast adaptation of deep networks*. In: *International Conference on Machine Learning*. Proceedings of Machine Learning Research, pp. 1126–1135.
- Finn, Chelsea, Aravind Rajeswaran, Sham Kakade, and Sergey Levine (2019). *On-line meta-learning*. In: *International Conference on Machine Learning*. Proceedings of Machine Learning Research, pp. 1920–1930.
- Gelada, Carles, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G. Belle-mare (2019). *Deepmdp: Learning continuous latent space models for representation learning*. In: *International Conference on Machine Learning*. Proceedings of Machine Learning Research, pp. 2170–2179.
- Gupta, Abhishek, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine (2018). *Meta-reinforcement learning of structured exploration strategies*. In: *arXiv preprint arXiv:1802.07245*.
- Hazan, Elad (2019). *Introduction to online convex optimization*. In: *arXiv preprint arXiv:1909.05207*.
- Ho, Dimitar, Hoang Le, John Doyle, and Yisong Yue (2021). *Online robust control of nonlinear systems with large uncertainty*. In: *International Conference on Artificial Intelligence and Statistics*.
- Kakade, Sham, Akshay Krishnamurthy, Kendall Lowrey, Motoya Ohnishi, and Wen Sun (2020). *Information theoretic regret bounds for online nonlinear control*. In: *arXiv preprint arXiv:2006.12466*.
- Kawale, Jaya, Hung H. Bui, Branislav Kveton, Long Tran-Thanh, and Sanjay Chawla (2015). *Efficient thompson sampling for online matrix-factorization recommendation*. In: *Advances in Neural Information Processing Systems*, pp. 1297–1305.
- Khodak, Mikhail, Maria-Florina Balcan, and Ameet Talwalkar (2019). *Adaptive gradient-based meta-learning methods*. In: *arXiv preprint arXiv:1906.02717*.
- Kingma, Diederik P. and Jimmy Ba (2014). *Adam: A method for stochastic optimization*. In: *arXiv preprint arXiv:1412.6980*.
- LaValle, Steven M. (2006). *Planning algorithms*. Cambridge University Press.
- Lee, Joonho, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter (2020). *Learning quadrupedal locomotion over challenging terrain*. In: *Science Robotics* 5.47.
- Lesort, Timothée, Natalia Díaz-Rodríguez, Jean-Francois Goudou, and David Fil-liat (2018). *State representation learning for control: An overview*. In: *Neural Networks* 108, pp. 379–392.
- Li, Yingying, Xin Chen, and Na Li (2019). *Online optimal control with linear dynamics and predictions: Algorithms and regret analysis*. In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 14858–14870.

- Liu, Anqi, Guanya Shi, Soon-Jo Chung, Anima Anandkumar, and Yisong Yue (2020). *Robust regression for safe exploration in control*. In: *Learning for Dynamics and Control*. PMLR, pp. 608–619. URL: <https://proceedings.mlr.press/v120/liu20a.html>.
- Mania, Horia, Stephen Tu, and Benjamin Recht (2019). *Certainty equivalence is efficient for linear quadratic control*. In: *arXiv preprint arXiv:1902.07826*.
- Maurer, Andreas, Massimiliano Pontil, and Bernardino Romera-Paredes (2016). *The benefit of multitask representation learning*. In: *Journal of Machine Learning Research* 17.81, pp. 1–32.
- Murray, Richard M., Zexiang Li, and S. Shankar Sastry (Dec. 2017). *A mathematical introduction to robotic manipulation*. 1st ed. CRC Press. ISBN: 978-1-315-13637-0. DOI: 10.1201/9781315136370. URL: <https://www.taylorfrancis.com/books/9781351469791>.
- Nagabandi, Anusha, Ignasi Clavera, Simin Liu, Ronald S. Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn (2018). *Learning to adapt in dynamic, real-world environments through meta-reinforcement learning*. In: *arXiv preprint arXiv:1803.11347*.
- Rahimi, Ali and Benjamin Recht (2007). *Random features for large-scale kernel machines*. In: *Proceedings of the 20th International Conference on Neural Information Processing Systems*, pp. 1177–1184.
- Richards, Spencer M., Navid Azizan, Jean-Jacques E. Slotine, and Marco Pavone (2021). *Adaptive-control-oriented meta-learning for nonlinear systems*. In: *arXiv preprint arXiv:2103.04490*.
- Shi, Xichen, Patrick Spieler, Ellande Tang, Elena-Sorina Lupu, Phillip Tokumaru, and Soon-Jo Chung (May 2020). *Adaptive nonlinear control of fixed-wing VTOL with airflow vector sensing*. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. Paris, France: IEEE, pp. 5321–5327. ISBN: 978-1-72817-395-5. DOI: 10.1109/ICRA40945.2020.9197344. URL: <https://ieeexplore.ieee.org/document/9197344/>.
- Simchowitz, Max and Dylan Foster (2020). *Naive exploration is optimal for online LQR*. In: *International Conference on Machine Learning*. Proceedings of Machine Learning Research, pp. 8937–8948.
- Slotine, Jean-Jacques E. and Weiping Li (1991). *Applied nonlinear control*. Vol. 199. 1. Prentice Hall, Englewood Cliffs, NJ.
- Tripuraneni, Nilesh, Chi Jin, and Michael I. Jordan (2020). *Provable meta-learning of linear representations*. In: *arXiv preprint arXiv:2002.11684*.
- Tripuraneni, Nilesh, Michael I. Jordan, and Chi Jin (2020). *On the theory of transfer learning: The importance of task diversity*. In: *arXiv preprint arXiv:2006.11650*.

- Wang, Huazheng, Qingyun Wu, and Hongning Wang (2017). *Factorization bandits for interactive recommendation*. In: *Proceedings of the AAAI Conference on Artificial Intelligence*.
- Zeng, Andy, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser (2020). *Tossingbot: Learning to throw arbitrary objects with residual physics*. In: *IEEE Transactions on Robotics* 36.4, pp. 1307–1319.
- Zhang, Amy, Zachary C. Lipton, Luis Pineda, Kamyar Azizzadenesheli, Anima Anandkumar, Laurent Itti, Joelle Pineau, and Tommaso Furlanello (2019). *Learning causal state representations of partially observable environments*. In: *arXiv preprint arXiv:1906.10437*.



## COMPETITIVE ONLINE OPTIMIZATION AND CONTROL

The recent progress in learning-theoretic analyses for online control begs the question: What learning-theoretic guarantees do we need for real-world systems? Existing results focus on linear systems with classic no-regret guarantees. However, no-regret policies compare with the optimal static controller in a specific class (typically linear policy class), which could be arbitrarily suboptimal in real-world nonlinear or time-varying systems. Therefore, this chapter builds interfaces between *competitive online optimization* and control, which uses stronger metrics beyond regret, i.e., competitive ratio and dynamic regret (competitive difference). Those metrics directly compare with the global optimum, thus naturally suitable for real-world systems. This chapter is mainly based on the following papers:

Pan, Weici, Guanya Shi, Yiheng Lin, and Adam Wierman (2022). *Online optimization with feedback delay and nonlinear switching cost*. In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 6.1, pp. 1–34. DOI: [10.1145/3508037](https://doi.org/10.1145/3508037).

Shi, Guanya, Yiheng Lin, Soon-Jo Chung, Yisong Yue, and Adam Wierman (2020). *Online optimization with memory and competitive control*. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 33. Curran Associates, Inc. URL: <https://arxiv.org/abs/2002.05318>.

---

## 10.1 Introduction

This chapter studies the problem of Online Convex Optimization (OCO) with *memory*, a variant of classic OCO (Hazan, 2019) where an online learner iteratively picks an action  $y_t$  and then suffers a convex loss  $g_t(y_{t-p}, \dots, y_t)$ , depending on current and *previous* actions. Incorporating memory into OCO has seen increased attention recently, due to both its theoretical implications, such as to convex body chasing problems (Bubeck, Lee, et al., 2019; Argue et al., 2020; Sellke, 2020; Bubeck, Klartag, et al., 2020), and its wide applicability to settings such as data centers (M. Lin, Liu, et al., 2012), power systems (Li, Qu, et al., 2018b; Badiei et al., 2015; Kim and Giannakis, 2016), and electric vehicle charging (Kim and Giannakis, 2016; S.

Chen and Tong, 2012). Of particular relevance to this chapter is the considerable recent effort studying connections between OCO with memory with online control in dynamical systems, leading to online algorithms that enjoy sublinear static regret (N. Agarwal, Bullins, et al., 2019; N. Agarwal, Hazan, et al., 2019), low dynamic regret (Li, Qu, et al., 2018a; Li, X. Chen, et al., 2019), constant competitive ratio (Goel and Wierman, 2019), and the ability to boost weak controllers (N. Agarwal, Brukhim, et al., 2019).

Prior work on OCO with memory is typically limited in one of two ways. First, algorithms with the strongest guarantees, a constant *competitive ratio*, are restricted to a special case known as Smoothed Online Convex Optimization (SOCO), or OCO with switching costs (N. Chen, Goel, et al., 2018; M. Lin, Liu, et al., 2012; Goel, Y. Lin, et al., 2019), which considers only one step of memory and assumes cost functions can be observed before actions are chosen. Second, algorithms proposed for the general case typically only enjoy sublinear *static regret* (Anava et al., 2015), which is a much weaker guarantee, because static regret compares to the offline optimal static solution while competitive ratio directly compares to the true offline optimal. It is known that algorithms that achieve sublinear static regret can be arbitrarily worse than the true offline optimal (Goel and Hassibi, 2020), and also may have unbounded competitive ratios (Andrew et al., 2013). The pursuit of general-purpose constant-competitive algorithms for OCO with memory remains open.

This chapter is also motivated by establishing theoretical connections between online optimization and control (see more discussions in Chapter 8). Recently a line of work has shown the applicability of tools from online optimization for control, albeit in limited settings (N. Agarwal, Bullins, et al., 2019; N. Agarwal, Hazan, et al., 2019; Lale et al., 2020; Goel and Wierman, 2019). Deepening these connections can potentially be impactful since most prior work studies how to achieve sublinear regret compared to the best static linear controller (Dean et al., 2020; N. Agarwal, Hazan, et al., 2019; N. Agarwal, Bullins, et al., 2019; Cohen et al., 2018). However, the best static linear controller is a weak benchmark compared to the true optimal controller (Goel and Hassibi, 2020), which may be neither linear nor static. To achieve stronger guarantees, one must seek to bound either the competitive ratio (Goel and Wierman, 2019) or dynamic regret (Li, Qu, et al., 2018a; Li, X. Chen, et al., 2019), and connections to online optimization can provide such results. However, prior attempts either have significant caveats (e.g., bounds depend on the

path length of the instance (Li, Qu, et al., 2018a; Li, X. Chen, et al., 2019)) or only apply to very restricted control systems (e.g., invertible control actuation matrices and perfect knowledge of disturbances (Goel and Wierman, 2019)). As such, the potential to obtain constant-competitive policies for general control systems via online optimization remains unrealized.

## 10.2 Problem Statement: OCO with Structured Memory

In this section, we first survey prior work on Online Convex Optimization (OCO) with memory and then introduce our new model of OCO with *structured* memory.

### OCO with Memory

OCO with memory is a variant of classic OCO that was first introduced by Anava et al. (2015). In contrast to classic OCO, in OCO with memory, the loss function depends on previous actions in addition to the current action. At time step  $t$ , the online agent picks  $y_t \in \mathcal{K} \subset \mathbb{R}^d$  and then a loss function  $g_t : \mathcal{K}^{p+1} \rightarrow \mathbb{R}$  is revealed. The agent incurs a loss of  $g_t(y_{t-p:t})$ . Thus,  $p$  quantifies the length of the memory in the loss function. Within this general model of OCO with memory, Anava et al. (2015) focus on developing policies with small *policy regret*, which is defined as:

$$\text{policy regret} = \sum_{t=p}^T g_t(y_{t-p:t}) - \min_{y \in \mathcal{K}} \sum_{t=0}^T g_t(y, \dots, y).$$

The main result presents a memory-based online gradient descent algorithm that achieves  $O(\sqrt{T})$  regret under some moderate assumptions on the diameter of  $\mathcal{K}$  and the gradient of the loss functions.

### OCO with Switching Costs (SOCO)

While the general form of OCO with memory was introduced only recently, specific forms of OCO problems involving memory have been studied for decades. The most prominent example is OCO with switching costs, often termed Smoothed Online Convex Optimization (SOCO) (M. Lin, Liu, et al., 2012; N. Chen, A. Agarwal, et al., 2015; N. Chen, Goel, et al., 2018; Goel and Wierman, 2019; Li, Qu, et al., 2018b; Goel, Y. Lin, et al., 2019). In SOCO, the loss function is separated into two pieces:

1. A *hitting cost*  $f_t$ , which depends on only the current action  $y_t$ .
2. A *switching cost*  $c(y_t, y_{t-1})$ , which penalizes big changes in the action between rounds.

Often the hitting cost is assumed to be of the form  $\|y_t - v_t\|$  for some (squared) norm, motivated by tracking some unknown trajectory  $v_t$ , and the switching cost  $c$  is a (squared) norm motivated by penalizing switching in proportion to the (squared) distance between the actions, e.g., a common choice  $c(y_t, y_{t-1}) = \frac{1}{2} \|y_t - y_{t-1}\|_2^2$ . The goal of the online learner is to minimize its total cost over  $T$  rounds:

$$\text{cost}(ALG) = \sum_{t=1}^T f_t(y_t) + c(y_t, y_{t-1}).$$

Under SOCO, results characterizing the policy regret are straightforward, and the goal is instead to obtain stronger results that characterize the *competitive ratio*. The competitive ratio is the worst-case ratio of total cost incurred by the online learner and the offline optimal. The cost of the offline optimal is defined as the minimal cost of an algorithm if it has full knowledge of the sequence  $\{f_t\}$ , i.e.:

$$\text{cost}(OPT) = \min_{y_1 \dots y_T} \sum_{t=1}^T f_t(y_t) + c(y_t, y_{t-1}).$$

Using this, the *competitive ratio* is defined as:

$$\text{competitive ratio}(ALG) = \sup_{f_{1:T}} \frac{\text{cost}(ALG)}{\text{cost}(OPT)}.$$

Bounds for competitive ratio are stronger than for policy regret, since the dynamic offline optimal can change its decisions on different time steps.

In the context of SOCO, the first results bounding the competitive ratio focused on one-dimensional action sets (M. Lin, Wierman, et al., 2013; Bansal et al., 2015), but after a long series of papers there now exist algorithms that provide constant competitive ratios in high dimensional settings (N. Chen, Goel, et al., 2018; Goel and Wierman, 2019; Goel, Y. Lin, et al., 2019). Among different choices of switching cost  $c$ , we are particularly interested in  $c(y_t, y_{t-1}) = \frac{1}{2} \|y_t - y_{t-1}\|_2^2$  due to the connection to quadratic costs in control problems. The state-of-the-art algorithm for this switching cost is Regularized Online Balanced Descent (ROBD), introduced by Goel, Y. Lin, et al. (2019), which achieves the lowest possible competitive ratio of any online algorithm. Other recent results study the case where  $c(y_t, y_{t-1}) = \|y_t - y_{t-1}\|$  (Bubeck, Lee, et al., 2019; Argue et al., 2020; Sellke, 2020; Bubeck, Klartag, et al., 2020). Variants of the problem with predictions (N. Chen, A. Agarwal, et al., 2015; N. Chen, Comden, et al., 2016; Li, Qu, et al., 2018b), non-convex cost functions

(Y. Lin et al., 2020), and constraints (Q. Lin et al., 2019; Yang et al., 2020) have been studied as well.

### OCO with Structured Memory

Though competitive algorithms have been proposed for many SOCO instances, the SOCO setting has two limitations. First, the hitting cost  $f_t$  is revealed before making action  $y_t$ , i.e., SOCO requires one step exact prediction of  $f_t$ . Second, the switching cost in SOCO only depends on one previous action in the form  $c(y_t, y_{t-1})$ , so only one step of memory is considered. In this chapter, our goal is to derive competitive algorithms (as exist for SOCO) in more general settings where more than one step of memory is considered. Working with the general model of OCO with memory is too ambitious for this goal. Instead, we introduce a model of OCO with *structured* memory that generalizes SOCO, and is motivated by a nontrivial connection with online control (see Example 10.1).

Specifically, we consider a loss function  $g_t$  at time step  $t$  that can be decomposed as the sum of a hitting cost function  $f_t : \mathbb{R}^d \rightarrow \mathbb{R}^+ \cup \{0\}$  and a switching cost function  $c : \mathbb{R}^{d \times (p+1)} \rightarrow \mathbb{R}^+ \cup \{0\}$ . Additionally, we assume that the switching cost has the form:

$$c(y_{t:t-p}) = \frac{1}{2} \left\| y_t - \sum_{i=1}^p C_i y_{t-i} \right\|_2^2,$$

with known  $C_i \in \mathbb{R}^{d \times d}$ ,  $i = 1, \dots, p$ . Note that SOCO is a special case  $p = 1$  and  $C_1 = I$ .

**Example 10.1** (Relations between the structured memory and control). *Intuitively, the hitting cost penalizes the agent for deviating from an optimal point sequence, while the switching cost captures the cost of implementing a control action. Specifically, suppose  $y_t$  is a robot's position at  $t$ , and then the classic SOCO switching cost  $\|y_t - y_{t-1}\|_2$  is approximately its "velocity." Under our new switching cost, we can represent acceleration by  $\|y_t - 2y_{t-1} + y_{t-2}\|_2$ , and many other higher-order dynamics. In other word, the classic SOCO models the single integrator dynamics where the control input can directly decide the robot's velocity, but the  $\frac{1}{2} \|y_t - \sum_{i=1}^p C_i y_{t-i}\|_2^2$  structure can model higher-order systems, such as the double integrator dynamics where the control input can only directly decide the robot's acceleration. We present a formal statement of this connection in Section 10.4.*

To summarize, we consider an online agent and an offline adversary interacting as follows in each time step  $t$ , and we assume  $y_i$  is already fixed for  $i = -p, -(p -$

1),  $\dots$ , 0.

1. The adversary reveals a function  $h_t$  and a convex *estimation set*  $\Omega_t \subseteq \mathbb{R}^d$ . We assume  $h_t$  is both  $m$ -strongly convex and  $l$ -strongly smooth, and that  $\arg \min_y h_t(y) = 0$ .
2. The agent picks  $y_t \in \mathbb{R}^d$ .
3. The adversary picks  $v_t \in \Omega_t$ .
4. The agent incurs *hitting cost*  $f_t(y_t) = h_t(y_t - v_t)$  and *switching cost*  $c(y_{t:t-p})$ .

Notice that the hitting cost  $f_t$  is revealed to the online agent in two separate steps. The geometry of  $f_t$  (given by  $h_t$  whose minimizer is at 0) is revealed before the agent picks  $y_t$ . After  $y_t$  is picked, the minimizer  $v_t$  of  $f_t$  is revealed.

Unlike SOCO, due to the uncertainty about  $v_t$ , the agent cannot determine the exact value of the hitting cost it incurs at time step  $t$  when determining its action  $y_t$ . To keep the problem tractable, we assume an estimation set  $\Omega_t$ , which contains all possible  $v_t$ 's, is revealed to bound the uncertainty. The agent can leverage this information when picking  $y_t$ . SOCO is a special case where  $\Omega_t$  contains only one point, i.e.,  $\Omega_t = \{v_t\}$ , and then the agent has a precise estimate of the minimizer  $v_t$  when choosing its action (Goel and Wierman, 2019; Goel, Y. Lin, et al., 2019). Like SOCO, the offline optimal cost in the structured memory model is defined as  $\text{cost}(OPT) = \min_{y_1 \dots y_T} \sum_{t=1}^T f_t(y_t) + c(y_{t:t-p})$  given the full sequence  $\{f_t\}_{t=1}^T$ .

### 10.3 Algorithms and Theoretical Analysis

In OCO with structured memory, there is a key differentiation depending on whether the agent has knowledge of the hitting cost function (both  $h_t$  and  $v_t$ ) when choosing its action or not, i.e., whether the estimation set  $\Omega_t$  is a single point,  $v_t$ , or not. We deal with each case in turn in the following.

#### **Case 1: Exact Prediction of $v_t$ ( $\Omega_t = \{v_t\}$ )**

We first study the simplest case where  $\Omega_t = \{v_t\}$ . Recall that  $\Omega_t$  is the convex set which contains all possible  $v_t$  and so, in this case, the agent has exact knowledge of the hitting cost when picking action. This assumption, while strict, is standard in the SOCO literature, e.g., Goel and Wierman (2019) and Goel, Y. Lin, et al. (2019). It is appropriate for situations where the cost function can be observed before choosing

---

**Algorithm 10.1:** Regularized OBD (ROBD), Goel, Y. Lin, et al. (2019)

---

**Parameter:**  $\lambda_1 \geq 0, \lambda_2 \geq 0$

1 **for**  $t = 1$  **to**  $T$  **do**

**Input:** Hitting cost function  $f_t$ , previous decision points  $y_{t-p}, \dots, y_{t-1}$

2      $v_t \leftarrow \arg \min_y f_t(y)$

3      $y_t \leftarrow \arg \min_y f_t(y) + \lambda_1 c(y, y_{t-1:t-p}) + \frac{\lambda_2}{2} \|y - v_t\|_2^2$

**Output:**  $y_t$

---

an action, e.g., Li, Qu, et al. (2018b), Kim and Giannakis (2016), and Goel and Wierman (2019).

Our main result in this setting is the following theorem, which shows that the ROBD algorithm (Algorithm 10.1), which is the state-of-the-art algorithm for SOCO, performs well in the more general case of structured memory. Note that, in this setting, the smoothness parameter  $l$  of hitting cost functions is not involved in the competitive ratio bound.

**Theorem 10.1.** *Suppose the hitting cost functions are  $m$ -strongly convex and the switching cost is given by  $c(y_{t:t-p}) = \frac{1}{2} \|y_t - \sum_{i=1}^p C_i y_{t-i}\|_2^2$ , where  $C_i \in \mathbb{R}^{d \times d}$  and  $\sum_{i=1}^p \|C_i\|_2 = \alpha$ . The competitive ratio of ROBD with parameters  $\lambda_1$  and  $\lambda_2$  is upper bounded by:*

$$\max \left\{ \frac{m + \lambda_2}{m\lambda_1}, \frac{\lambda_1 + \lambda_2 + m}{(1 - \alpha^2)\lambda_1 + \lambda_2 + m} \right\},$$

if  $\lambda_1 > 0$  and  $(1 - \alpha^2)\lambda_1 + \lambda_2 + m > 0$ . If  $\lambda_1$  and  $\lambda_2$  satisfy  $m + \lambda_2 = \frac{m + \alpha^2 - 1 + \sqrt{(m + \alpha^2 - 1)^2 + 4m}}{2}$ ,  $\lambda_1$ , then the competitive ratio is:

$$\frac{1}{2} \left( 1 + \frac{\alpha^2 - 1}{m} + \sqrt{\left( 1 + \frac{\alpha^2 - 1}{m} \right)^2 + \frac{4}{m}} \right).$$

The proof of Theorem 10.1 is given in the appendix (Section 10.7). To get insight into Theorem 10.1, first consider the case when  $\alpha$  is a constant. In this case, the competitive ratio is of order  $O(1/m)$ , which highlights that the challenging setting is when  $m$  is small. It is easy to see that this upper bound is in fact tight. To see this, note that the case of SOCO with  $\ell_2$  squared switching cost considered in Goel and Wierman (2019) and Goel, Y. Lin, et al. (2019) is a special case where  $p = 1, C_1 = I, \alpha = 1$ . Substituting these parameters into Theorem 10.1 gives exactly the same upper bound (including constants) as Goel, Y. Lin, et al. (2019), which has been shown to match a lower bound on the achievable cost of any online algorithm, including constant factors. On the other hand, if we instead assume that  $m$  is a fixed

positive constant. The competitive ratio can be expressed as  $1 + O(\alpha^2)$ . Therefore, the competitive ratio gets worse quickly as  $\alpha$  increases. This is also the best possible scaling, achievable via any online algorithm, as we show in the original paper (see Shi et al., 2020, Appendix D).

Perhaps surprisingly, the memory length  $p$  does not appear in the competitive ratio bound, which contradicts the intuition that the online optimization problem should get harder as the memory length increases. However, it is worth noting that  $\alpha$  becomes larger as  $p$  increases, so the memory length implicitly impacts the competitive ratio. For example, an interesting form of switching cost is

$$c(y_{t:t-p}) = \frac{1}{2} \left\| \sum_{i=0}^p (-1)^i \binom{p}{i} y_{t-i} \right\|_2^2,$$

which corresponds to the  $p^{\text{th}}$  derivative of  $y$  and generalizes SOCO ( $p = 1$ ). In this case, we have  $\alpha = 2^p - 1$ . Hence  $\alpha$  grows exponentially in  $p$ .

### Case 2: Inexact Prediction of $v_t$ ( $v_t \in \Omega_t$ )

For general  $\Omega_t$ , ROBD is no longer enough. It needs to be adapted to handle the uncertainty that results from the estimation set  $\Omega_t$ . Note that this uncertainty set is crucial for many applications, such as online control with adversarial disturbances in Section 10.4.

---

#### Algorithm 10.2: Optimistic ROBD

---

**Parameter:**  $\lambda \geq 0$

1 **for**  $t = 1$  **to**  $T$  **do**

- 2     **Input:**  $v_{t-1}, h_t, \Omega_t$
- 3     Initialize a ROBD instance with  $\lambda_1 = \lambda, \lambda_2 = 0$
- 4     Recover  $f_{t-1}(y) = h_{t-1}(y - v_{t-1})$
- 5      $\hat{y}_{t-1} \leftarrow \text{ROBD}(f_{t-1}, \hat{y}_{t-p-1:t-2})$
- 6      $\tilde{v}_t \leftarrow \arg \min_{v \in \Omega_t} \min_y h_t(y - v) + \lambda c(y, \hat{y}_{t-1:t-p})$
- 7     Estimate  $\tilde{f}_t(y) = h_t(y - \tilde{v}_t)$
- 8      $y_t \leftarrow \text{ROBD}(\tilde{f}_t, \hat{y}_{t-p:t-1})$

**Output:**  $y_t$  (the decision at time step  $t$ )

---

To handle this additional complexity, we propose Optimistic ROBD (Algorithm 10.2). Optimistic ROBD is based on two key ideas. The first is to ensure that the algorithm tracks the sequence of actions it would have made if given observations of the true cost functions before choosing an action. To formalize this, we define the *accurate sequence*  $\{\hat{y}_1, \dots, \hat{y}_T\}$  to be the choices of ROBD (Algorithm 10.1) with



$\lambda_1 = \lambda$ ,  $\lambda_2 = 0$  when each hitting cost  $f_t$  is revealed before picking  $\hat{y}_t$ . The goal of Optimistic ROBD (Algorithm 10.2) is to approximate the accurate sequence. In order to track the accurate sequence, the first step is to recover it up to time step  $t - 1$  at time step  $t$ . To do this, after we observe the previous minimizer  $v_{t-1}$ , we can compute the accurate choice of ROBD as if both  $h_{t-1}$  and  $v_{t-1}$  are observed before picking  $y_{t-1}$ . Therefore, Algorithm 10.2 can compute the *accurate subsequence*  $\{\hat{y}_1, \dots, \hat{y}_{t-1}\}$  at time step  $t$ . Picking  $y_t$  based on the accurate sequence  $\{\hat{y}_1, \dots, \hat{y}_{t-1}\}$  instead of the noisy sequence  $\{y_1, \dots, y_{t-1}\}$  ensures that the actions do not drift too far from the accurate sequence.

The second key idea is to be optimistic by assuming the adversary will give it  $v \in \Omega_t$  that minimizes the cost it will experience. Specifically, before  $v_t$  is revealed, the algorithm assumes it is the point in  $\Omega_t$  which minimizes the weighted sum  $h_t(y - v) + \lambda c(y, \hat{y}_{t-1:t-p})$  if ROBD is implemented with parameter  $\lambda$  to pick  $y$ . This ensures that additional cost is never taken unnecessarily, which could be exploited by the adversary. Note that  $\min_y h_t(y - v) + \lambda c(y)$  is strongly convex with respect to  $v$  (see Shi et al., 2020, Appendix E), so it is tractable even if  $\Omega_t$  is unbounded.

The following theorem bounds the competitive ratio of Optimistic ROBD.

**Theorem 10.2** (The upper bound of the competitive ratio of Optimistic ROBD). *Suppose the hitting cost functions are both  $m$ -strongly convex and  $l$ -strongly smooth and the switching cost is given by  $c(y_{t:t-p}) = \frac{1}{2} \|y_t - \sum_{i=1}^p C_i y_{t-i}\|_2^2$ , where  $C_i \in \mathbb{R}^{d \times d}$  and  $\sum_{i=1}^p \|C_i\|_2 = \alpha$ . For arbitrary  $\eta > 0$ , the cost of Optimistic ROBD with parameter  $\lambda > 0$ , is upper bounded by  $K_1 \text{cost}(OPT) + K_2$ , where:*

$$K_1 = (1 + \eta) \max \left\{ \frac{1}{\lambda}, \frac{\lambda + m}{(1 - \alpha^2)\lambda + m} \right\},$$

$$K_2 = \lambda \left( \frac{l}{1 + \eta - \lambda} + \frac{4\alpha^2}{\eta} - \frac{m}{\lambda + m} \right) \sum_{t=1}^T \frac{\|v_t - \tilde{v}_t\|^2}{2}.$$

The proof of Theorem 10.2 is given in (Shi et al., 2020, Appendix E). This proof relies on the two key ideas we mentioned before. Note that we can choose  $\eta$  to balance  $K_1$  and  $K_2$  and obtain a competitive ratio, in particular the smallest  $\eta$  such that:

$$\lambda \left( \frac{l}{1 + \eta - \lambda} + \frac{4\alpha^2}{\eta} - \frac{m}{\lambda + m} \right) \leq 0.$$

Therefore, we have  $\eta = O(l + \alpha^2)$  and  $K_2 \leq 0$ . So the competitive ratio is upper bounded by:

$$O \left( (l + \alpha^2) \max \left\{ \frac{1}{\lambda}, \frac{\lambda + m}{(1 - \alpha^2)\lambda + m} \right\} \right).$$

However, the reason we present Theorem 10.2 in terms of  $K_1$  and  $K_2$  is that, when the diameter of  $\Omega_t$  is small, we can pick a small  $\eta$  so that the ratio coefficient  $K_1$  will be close to the competitive ratio of ROBD when  $v_t$  is known before picking  $y_t$ . This “beyond-the-worst-case” analysis is useful in many applications.

#### 10.4 Connections to Competitive Control

Goel and Wierman (2019) show a connection between SOCO and online control in the setting where disturbance is perfectly known at time step  $t$  and the control actuation matrix  $B$  is invertible, which leads to the only constant-competitive control policy as far as we know. Since the new proposed OCO with structured memory generalizes SOCO, one may expect its connects to more general dynamical systems. In this section, we present a nontrivial reduction from *Input-Disturbed Squared Regulators (IDSRs)* to OCO with structured memory, leading to the first constant-competitive policy in online control with adversarial disturbance.

#### Control Problem Setting

$$A = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & & & \ddots & \\ 0 & 0 & 0 & \dots & 1 \\ * & * & * & \dots & * & * & * & \dots & * & \dots & * \\ & & & & 0 & 1 & 0 & \dots & 0 & \dots & \\ & & & & 0 & 0 & 1 & \dots & 0 & \dots & \\ & & & & \vdots & & & \ddots & & & \\ & & & & 0 & 0 & 0 & \dots & 1 & \dots & \\ * & * & * & \dots & * & * & * & \dots & * & \dots & * \\ & & & & \vdots & & & & & & \\ & & & & & & 0 & 1 & 0 & \dots & 0 \\ & & & & & & 0 & 0 & 1 & \dots & 0 \\ & & & & & & \vdots & & & \ddots & \\ & & & & & & 0 & 0 & 0 & \dots & 1 \\ * & & & \dots & * & * & * & \dots & * & & \end{bmatrix}, B = \begin{bmatrix} 0 & 0 & \dots \\ 0 & 0 & \dots \\ \vdots & & \\ 0 & 0 & \dots \\ 1 & 0 & \dots \\ 0 & 0 & \dots \\ 0 & 0 & \dots \\ \vdots & & \\ 0 & 0 & \dots \\ 0 & 1 & 0 & \dots \\ \vdots & & \\ 0 & 0 & \dots \\ 0 & 0 & \dots \\ \vdots & & \\ 0 & 0 & \dots \\ 0 & 0 & \dots & 1 \end{bmatrix},$$

Figure 10.1: Controllable canonical form.

**Input-disturbed systems.** We focus on systems in controllable canonical form defined by:

$$x_{t+1} = Ax_t + B(u_t + w_t), \quad (10.1)$$

where  $x_t \in \mathbb{R}^n$  is the state,  $u_t \in \mathbb{R}^d$  is the control, and  $w_t \in \mathbb{R}^d$  is a potentially adversarial disturbance to the system. We further assume that  $(A, B)$  is in controllable canonical form (see Fig. 10.1), where each  $*$  represents a (possibly) non-zero entry, and the rows of  $B$  with 1 are the same rows of  $A$  with  $*$  (Luenberger, 1967). It is well-known that any controllable system can be linearly transformed to the canonical form. This system is more restrictive than the general form in linear

systems. We call these *Input-Disturbed* systems, since the disturbance  $w_t$  is in the control input/action space. There are many corresponding real-world applications that are well-described by Input-Disturbed systems, e.g., external/disturbance force in robotics (see Part I).

**Squared regulator costs.** We consider the following cost model for the controller:

$$c_t(x_t, u_t) = \frac{q_t}{2} \|x_t\|_2^2 + \frac{1}{2} \|u_t\|_2^2, \quad (10.2)$$

where  $q_t$  is a positive scalar. The sequence  $q_{0:T}$  is picked by the adversary and revealed online. The objective of the controller is to minimize the total control cost  $\sum_{t=0}^T c_t(x_t, u_t)$ . We call this cost model the *Squared Regulator* model, which is a restriction of the classical quadratic cost model. This class of costs is general enough to address a fundamental trade-off in optimal control: the trade-off between the state cost and the control effort (Kirk, 2004).

**Disturbances.** In the online control literature, a variety of assumptions have been made about the noise  $w_t$ . In most works, the assumption is that the exact noise  $w_t$  is not known before  $u_t$  is taken. Many assume  $w_t$  is drawn from a certain known distribution, e.g., N. Agarwal, Hazan, et al. (2019). Others assume  $w_t$  is chosen adversarially subject to  $\|w_t\|_2$  being upper bounded by a constant  $W$ , e.g., N. Agarwal, Bullins, et al. (2019). In a closely related paper, Goel and Wierman (2019) connect SOCO with online control under the assumption that  $w_t$  can be observed before picking the control action  $u_t$ . In contrast, in this chapter we assume that the exact  $w_t$  is not observable before the agent picks  $u_t$ . Instead, we assume a convex estimation set  $W_t$  (not necessarily bounded) that contains all possible  $w_t$  is revealed to the online agent to help the agent decide  $u_t$ . Our assumption is a generalization of Goel and Wierman (2019), where  $W_t$  is a one-point set, and N. Agarwal, Bullins, et al. (2019), where  $W_t$  is a ball of radius  $W$  centered at 0. Our setting can also naturally model time-Lipschitz noise, where  $w_t$  is chosen adversarially subject to  $\|w_t - w_{t-1}\|_2 \leq \epsilon$ , by picking  $W_t$  as a sphere of radius  $\epsilon$  centered at  $w_{t-1}$ , which has many real-applications such as smooth disturbances in robotics. Moreover, note that our setting is naturally adaptive because of the estimation set  $W_t$  (e.g., controller may choose more aggressive action if  $W_t$  is small), which is different from the classic  $\mathcal{H}_\infty$  control setting (Zhou and Doyle, 1998).

**Competitive ratio.** Our goal is to develop policies with constant (small) competitive ratios. This is a departure from the bulk of the literature (N. Agarwal, Hazan, et al., 2019; N. Agarwal, Bullins, et al., 2019; Dean et al., 2020; Cohen et al.,

2018), which focuses on designing policies that have low regret compared to the optimal linear controller. We show the optimal linear controller can have cost arbitrarily larger than the offline optimal, via an analytic example (see Shi et al., 2020, Appendix B). We again denote the offline optimal cost, with full knowledge of the sequence  $w_{0:T}$ , as  $\text{cost}(OPT) = \min_{u_{0:T}} \sum_{t=0}^T c_t(x_t, u_t)$ . For an online algorithm  $ALG$ , let  $\text{cost}(ALG)$  be its cost on the same disturbance sequence  $w_{0:T}$ . The competitive ratio is then the worst-case ratio of  $\text{cost}(ALG)$  and  $\text{cost}(OPT)$  over any disturbance sequence, i.e.,  $\sup_{w_{0:T}} \text{cost}(ALG)/\text{cost}(OPT)$ . To the best of our knowledge, the only prior work that studies competitive algorithms for online control is Goel and Wierman (2019), which considers a very restricted system with invertible  $B$  and known  $w_t$  at step  $t$ . A related line of online optimization research studies *dynamic regret*, or *competitive difference*, defined as the difference between online algorithm cost and the offline optimal (Li, X. Chen, et al., 2019; Li, Qu, et al., 2018a). For example, Li, X. Chen, et al. (2019) bound the dynamic regret of online control with time-varying convex costs with no noise. However, results for the dynamic regret depend on the path-length or variation budget, not just system properties. Bounding the competitive ratio is typically more challenging.

### A Reduction to OCO with Structured Memory

---

**Algorithm 10.3:** Reduction from Online Control to OCO with Structured Memory

---

**Input:** Transition matrix  $A$  and control matrix  $B$

**Solver:** OCO with structured memory algorithm  $ALG$

```

1 for  $t = 0$  to  $T - 1$  do
    Observe:  $x_t, W_t$ , and  $q_{t:t+p-1}$ 
2   if  $t > 0$  then
3      $w_{t-1} \leftarrow \psi(x_t - Ax_{t-1} - Bu_{t-1})$ 
4      $\zeta_{t-1} \leftarrow w_{t-1} + \sum_{i=1}^p C_i \zeta_{t-1-i}$ 
5      $v_{t-1} \leftarrow -\zeta_{t-1}$ 
6   Define  $h_t(y) = \frac{1}{2} \sum_{i=1}^d \left( \sum_{j=1}^{p_i} q_{t+j} \right) \left( y^{(i)} \right)^2$ 
7   Define  $\Omega_t = \{-w - \sum_{i=1}^p C_i \zeta_{t-i} \mid w \in W_t\}$ 
8   Feed  $v_{t-1}, h_t, \Omega_t$  into  $ALG$ 
9   Obtain  $ALG$ 's output  $y_t$ 
10   $u_t \leftarrow y_t - \sum_{i=1}^p C_i y_{t-i}$ 
    Output:  $u_t$ 
Output:  $u_T = 0$ 

```

---

We now present a reduction from *IDSR* to OCO with structured memory. This reduction allows us to inherit the competitive ratio bounds on Optimistic ROBD

for this class of online control problems. Before presenting the reduction, we first introduce some important notations. The indices of non-zero rows in matrix  $B$  in (10.1) are denoted as  $\{k_1, \dots, k_d\} := \mathcal{I}$ . We define operator  $\psi : \mathbb{R}^n \rightarrow \mathbb{R}^d$  as:

$$\psi(x) = \left( x^{(k_1)}, \dots, x^{(k_d)} \right)^\top,$$

which extracts the dimensions in  $\mathcal{I}$ . Moreover, let  $p_i = k_i - k_{i-1}$  for  $1 \leq i \leq n$ , where  $k_0 = 0$ . The controllability index of the canonical-form  $(A, B)$  is defined as  $p = \max\{p_1, \dots, p_d\}$ . We assume that the initial state is zero, i.e.,  $x_0 = 0$ . In the reduction, we also need to use matrices  $C_i \in \mathbb{R}^{d \times d}$ ,  $i = 1, \dots, p$ , which regroup the columns of  $A(\mathcal{I}, :)$ . We define  $C_i$  for  $i = 1, \dots, p$  formally by constructing each of its columns. For  $j = 1, \dots, d$ , if  $i \leq p_j$ , the  $j$ th column of  $C_i$  is the  $(k_j + 1 - i)$ th column of  $A(\mathcal{I}, :)$ ; otherwise, the  $j$ th column of  $C_i$  is 0. Formally, for  $i \in \{1, \dots, p\}$ ,  $j \in \{1, \dots, d\}$ , we have:

$$C_i(:, j) = \begin{cases} A(\mathcal{I}, k_j + 1 - i) & \text{if } i \leq p_j \\ 0 & \text{otherwise.} \end{cases}$$

Based on coefficients  $q_{0:T}$ , we define:

$$q_{\min} = \min_{0 \leq t \leq T-1, 1 \leq i \leq d} \sum_{j=1}^{p_i} q_{t+j}, \quad q_{\max} = \max_{0 \leq t \leq T-1, 1 \leq i \leq d} \sum_{j=1}^{p_i} q_{t+j},$$

where we assume  $q_t = 0$  for all  $t > T$ .

**Theorem 10.3.** *Consider IDSR where the cost function and dynamics are specified by (10.2) and (10.1). We assume the coefficients  $q_{t:t+p-1}$  are observable at step  $t$ . Any instance of IDSR in controllable canonical form can be reduced to an instance of OCO with structured memory by Algorithm 10.3.*

A proof of Theorem 10.3 are given in (Shi et al., 2020, Appendix G) and an example are given in Example 10.2. Notably,  $cost(OPT)$  and  $cost(ALG)$  remain unchanged in the reduction described by Algorithm 10.3. In fact, Algorithm 10.3, when instantiated with Optimistic ROBD, provides an efficient algorithm for online control. It only requires  $O(p)$  memory to compute the recursive sequences.

**Example 10.2** (Reduction for a control system with  $\dim(x) = 5, \dim(u) = 2$ ). To illustrate the reduction, consider the following example:

$$\begin{bmatrix} x_{t+1}^{(1)} \\ x_{t+1}^{(2)} \\ x_{t+1}^{(3)} \\ x_{t+1}^{(4)} \\ x_{t+1}^{(5)} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ a_1 & a_2 & a_3 & a_4 & a_5 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ b_1 & b_2 & b_3 & b_4 & b_5 \end{bmatrix} \begin{bmatrix} x_t^{(1)} \\ x_t^{(2)} \\ x_t^{(3)} \\ x_t^{(4)} \\ x_t^{(5)} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \left( \begin{bmatrix} u_t^{(1)} \\ u_t^{(2)} \end{bmatrix} + \begin{bmatrix} w_t^{(1)} \\ w_t^{(2)} \end{bmatrix} \right). \quad (10.3)$$

Notice that since  $x_{t+1}^{(1)} = x_t^{(2)}, x_{t+1}^{(3)} = x_t^{(4)}$ , we can rewrite (10.3) in a more compact form:

$$\underbrace{\begin{bmatrix} x_{t+1}^{(2)} \\ x_{t+1}^{(5)} \end{bmatrix}}_{z_{t+1}} = \underbrace{\begin{bmatrix} a_2 & a_5 \\ b_2 & b_5 \end{bmatrix}}_{C_1} \begin{bmatrix} x_t^{(2)} \\ x_t^{(5)} \end{bmatrix} + \underbrace{\begin{bmatrix} a_1 & a_4 \\ b_1 & b_4 \end{bmatrix}}_{C_2} \begin{bmatrix} x_{t-1}^{(2)} \\ x_{t-1}^{(5)} \end{bmatrix} + \underbrace{\begin{bmatrix} 0 & a_3 \\ 0 & b_3 \end{bmatrix}}_{C_3} \begin{bmatrix} x_{t-2}^{(2)} \\ x_{t-2}^{(5)} \end{bmatrix} + \begin{bmatrix} u_t^{(1)} \\ u_t^{(2)} \end{bmatrix} + \begin{bmatrix} w_t^{(1)} \\ w_t^{(2)} \end{bmatrix}. \quad (10.4)$$

In this example  $p_1 = 2, p_2 = 3, \mathcal{I} = \{k_1, k_2\} = \{2, 5\}$  and thus  $p = 3$  and  $n = 2$ . From (10.4) we have

$$z_{t+1} = C_1 z_t + C_2 z_{t-1} + C_3 z_{t-2} + u_t + w_t. \quad (10.5)$$

Recall the definition of  $y_t$  and  $\zeta_t$ :

$$y_t = u_t + \sum_{i=1}^3 C_i y_{t-i}, \forall t \geq 0, \quad \zeta_t = w_t + \sum_{i=1}^3 C_i \zeta_{t-i}, \forall t \geq 0. \quad (10.6)$$

Then the original system could be translated to the compact form:

$$z_{t+1} = y_t + \zeta_t. \quad (10.7)$$

If the objective is given as  $\frac{1}{2} \sum_{t=0}^T (q_t \|x_t\|^2 + \|u_t\|^2)$ , we have that

$$h_t(z) = \frac{1}{2} (q_{t+1} + q_{t+2}) \left( z^{(1)} \right)^2 + \frac{1}{2} (q_{t+1} + q_{t+2} + q_{t+3}) \left( z^{(2)} \right)^2.$$

### Competitive Policy

The reduction in Algorithm 10.3 immediately translates the competitive ratio guarantees in Section 10.3 into competitive policies. As Theorem 10.2 suggests, we can tune  $\eta$  in Optimistic ROBD based on the quality of prediction. As a result, we present two forms of upper bounds for Algorithm 10.3 in Corollaries 10.1 and 10.2. Notably, Corollary 10.1 gives a tighter bound where good estimations are

available, while Corollary 10.2 gives a bound that does not depend on the quality of the estimations.

In the first case, we assume that a good estimation of  $w_t$  is available before picking  $u_t$ . Specifically, we assume the diameter of set  $W_t$  is upper bounded by  $\epsilon_t$  at time step  $t$ , where  $\epsilon_t$  is a small positive constant. We derive Corollary 10.1 by setting  $\eta = 1 + \lambda$  in Theorem 10.2.

**Corollary 10.1.** *In IDSR, assume that coefficients  $q_{t:t+p-1}$  are observable at time step  $t$ . Let  $\alpha = \sum_{i=1}^q \|C_i\|_2$ . When the diameter of  $W_t$  is upper bounded by  $\epsilon_t$  at time step  $t$ , the total cost incurred by Algorithm 10.3 (using Optimistic ROBD with parameter  $\lambda$ ) in the online control problem is upper bounded by  $K_1 \text{cost}(OPT) + K_2$ , where:*

$$K_1 = (2 + \lambda) \cdot \max \left\{ \frac{1}{\lambda}, \frac{\lambda + q_{\min}}{(1 - \alpha^2)\lambda + q_{\min}} \right\},$$

$$K_2 = \lambda \left( \frac{q_{\max}}{2} + \frac{4\alpha^2}{1 + \lambda} - \frac{q_{\min}}{\lambda + q_{\min}} \right) \cdot \sum_{t=0}^{T-1} \frac{1}{2} \epsilon_t^2.$$

The residue term  $K_2$  in Corollary 10.1 becomes negligible when the total estimation error  $\sum_{t=0}^{T-1} \epsilon_t^2$  is small, leading to a pure competitive ratio guarantee. Further, if we ignore  $K_2$ , the coefficient  $K_1$  is only constant factor worse than the ratio we obtain when exact prediction of  $w_t$  is available.

However, the bound in Corollary 10.1 can be significantly worse than the case where exact prediction is available when the diameter of  $W_t$  is large or unbounded. Hence we introduce a second corollary that does not use any information about  $w_t$  when picking  $u_t$ . Specifically, we assume the diameter of set  $W_t$  cannot be bounded, so the upper bound given in Corollary 10.1 is meaningless. By picking the parameter  $\eta$  such that  $\lambda \left( \frac{1}{1+\eta-\lambda} + \frac{4\alpha^2}{\eta} - \frac{m}{\lambda+m} \right) \leq 0$  in Theorem 10.2, we obtain the following result.

**Corollary 10.2.** *In IDSR, assume that coefficients  $q_{t:t+p-1}$  are observable at time step  $t$ . Let  $\alpha = \sum_{i=1}^q \|C_i\|_2$ . The competitive ratio of Algorithm 10.3, using Optimistic ROBD with  $\lambda$ , is upper bounded by:*

$$O \left( (q_{\max} + 4\alpha^2) \max \left\{ \frac{1}{\lambda}, \frac{\lambda + q_{\min}}{(1 - \alpha^2)\lambda + q_{\min}} \right\} \right).$$

Compared with Corollary 10.1, Corollary 10.2 gives an upper bound that is independent of the size of  $W_t$ . It is also a pure constant competitive ratio, without any

additive term. However, the ratio is worse than the case where exact prediction of  $w_t$  is available, especially when  $q_{\max}$  or  $\alpha$  is large.

**Contrasting no-regret and constant-competitive guarantees.** The predominant benchmark used in previous work on online control via learning is *static regret* relative to the best linear controller in hindsight, i.e.,  $u_t = -K^*x_t$  (Cohen et al., 2018; Abeille and Lazaric, 2018; N. Agarwal, Bullins, et al., 2019; N. Agarwal, Hazan, et al., 2019; Dean et al., 2020; Abbasi-Yadkori et al., 2018). For example, N. Agarwal, Hazan, et al. (2019) achieve logarithmic regret under stochastic noise and strongly convex loss, and N. Agarwal, Bullins, et al. (2019) achieve  $O(\sqrt{T})$  regret under adversarial noise and convex loss. However, the cost of the optimal linear controller may be far from the true offline optimal cost. Goel and Hassibi (2020) recently show that there is a linear regret between the optimal offline linear policy and the true offline optimal policy in online LQR control. Thus, achieving small regret may still mean having a significantly larger cost than optimal. We illustrate this difference and our algorithm’s performance by a 1-d analytic example (see Shi et al., 2020, Appendix B), and also numerical experiments in higher dimensions (Section 10.5). In particular, we see that the optimal linear controller can be significantly more costly than the offline optimal controller and that Optimistic ROBD can significantly outperform the optimal linear controller.

## 10.5 Simulations

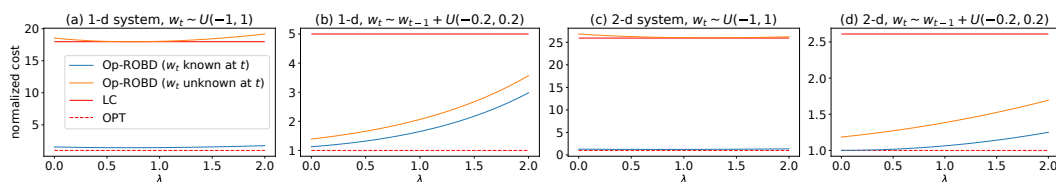


Figure 10.2: Numerical results of Optimistic ROBD in 1-d and 2-d systems, with different  $\lambda$ . LC means the best linear controller in hindsight and OPT means the global optimal controller in hindsight. LC is numerically searched in stable linear controller space. We consider two different types of  $w_t$ :  $w_t$  is i.i.d. random/random walk, and also two different settings:  $w_t$  is known/unknown at step  $t$ .

In this section, we use simple numerical examples to illustrate the contrast between the best linear controller in hindsight and the optimal offline controller. We also test our algorithm, Optimistic ROBO, and then numerically illustrate that Optimistic ROBD can obtain near-optimal cost and outperform the offline optimal linear controller.



In the first example we consider a simple 1-d system, where the object function is  $\sum_{t=0}^{200} 8|x_t|^2 + |u_t|^2$  and the dynamics is  $x_{t+1} = 2x_t + u_t + w_t$ . For the sequence  $\{w_t\}_{t=0}^T$ , we consider two cases, in the first case  $\{w_t\}_{t=0}^T$  is generated by  $w_t \sim \mathcal{U}(-1, 1)$  i.i.d., and in the second case the sequence is generated by  $w_{t+1} = w_t + \psi_t$  where  $\psi_t \sim \mathcal{U}(-0.2, 0.2)$  i.i.d.. The first case corresponds to unpredictable disturbances, where the estimation set  $W_t = (-1, 1)$ , and the second to smooth disturbances (i.e., a random walk), where  $W_t = w_{t-1} + (-0.2, 0.2)$ . For both types of  $\{w_t\}_{t=0}^T$ , we test Optimistic ROBD algorithms in two settings:  $w_t$  is known/unknown at step  $t$ . In the first setting,  $w_t$  is directly given to the algorithm, and in the latter setting, only  $W_t$  is given at time step  $t$ .

The results are shown in Fig. 10.2 (a-b). We see that if  $w_t$  is known at step  $t$ , Optimistic ROBD is much better than the best linear controller in hindsight, and almost matches the true optimal when  $w_t$  is smooth. In fact, when  $w_t$  is smooth, Optimistic ROBD is much better than the best linear controller even if it does not know  $w_t$  at step  $t$ . Even in the case when  $w_t \sim \mathcal{U}(-1, 1)$ , and so is extremely unpredictable, Optimistic ROBD's performance still matches the best linear controller, which uses perfect hindsight.

Our second example considers a 2-d system with the following objective and dynamics:

$$\min_{u_t} \sum_{t=0}^{200} 8\|x_t\|_2^2 + \|u_t\|_2^2, \quad \text{s.t. } x_{t+1} = \begin{bmatrix} 0 & 1 \\ -1 & 2 \end{bmatrix} x_t + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_t + \begin{bmatrix} 0 \\ 1 \end{bmatrix} w_t,$$

where  $(A, B)$  is the canonical form of double integrator dynamics. For this 2-d system, similarly, we test the performance of Optimistic ROBD with two types of  $w_t$ .

The results are shown in Fig. 10.2 (c-d) and reinforce the same observations we observed in the 1-d system. In particular, we see that the optimal linear controller can be significantly more costly than the offline optimal controller and that Optimistic ROBD can outperform the optimal linear controller, sometimes by a significant margin.

## 10.6 Extension: Nonlinear Switching Cost and Feedback Delay

Even though the proposed OCO with structured memory setting significantly generalizes the classic SOCO setting, and enables new competitive policies in online control with adversarial disturbance, there are two limitations.

First, the online learner observes the hitting cost function  $f_t$  either *before* (classic SOCO setting) or *one step after* (OCO with structured memory) picking the action  $y_t$ . However, in many applications, such as trajectory tracking problems in robotics,  $f_t$  is revealed after a multi-round delay due to communication and process delays, i.e., multiple rounds of actions must be taken *without* feedback on their hitting costs. Delay is known to be very challenging in practice and even *one-step* delay requires non-trivial algorithmic modifications (i.e., from ROBO to Optimistic ROBD). The impact of multi-round delay has been recognized as a challenging open question for the design of online algorithms (Joulani et al., 2013; Shamir and Szlak, 2017) and broadly in applications. For example, Chapter 4 highlights that a three-step delay (around 30 milliseconds) can already cause catastrophic crashes in drone tracking control using a standard controller without algorithmic adjustments for delay.

Second, either SOCO or OCO with structured memory allows only *linear* forms of switching cost functions, where the switching cost  $c$  is some (squared) norm of a linear combination of current and previous actions. However, in many practical scenarios the costs to move from  $y_{t-1}$  to  $y_t$  are non-trivial nonlinear functions. For example, consider  $y_t \in \mathbb{R}$  as the vertical velocity of a drone in a velocity control task. Hovering the drone (i.e., holding the position such that  $y_t = 0, \forall t$ ) is not free, due to gravity. In this case, the cost to move from  $y_{t-1}$  to  $y_t$  is  $(y_t - y_{t-1} + g(y_{t-1}))^2$  where the nonlinear term  $g(y_{t-1})$  accounts for the gravity and aerodynamic drag. Such non-linearities create significant algorithmic challenges because (i) in contrast to the linear setting, small movement between decisions does not necessarily imply small switching cost (e.g., the aforementioned drone control example), and (ii) a small error in a decision can lead to large non-linear penalties in the switching cost in future steps, which is further amplified by the multi-round delay. Addressing such challenges is well-known to be a challenging open question for the design of online algorithms.

Therefore, in this section we extend the proposed OCO with structured memory setting with a  $k$ -round multi-round feedback delay and nonlinear switching costs. In this extended setting, we introduce a new algorithm, Iterative Regularized Online Balanced Descent (iROBD) and prove that it maintains a dimension-free constant competitive ratio that is  $O(L^{2k})$ , where  $L$  is the Lipschitz constant of the non-linear switching cost and  $k$  is the delay. This is the first constant competitive algorithm in the case of either feedback delay or nonlinear switching cost and we show, via lower bounds, that the dependencies on both  $L$  and  $k$  are tight. These lower bounds

further serve to emphasize how the algorithmic difficulties increase as the length of delay becomes longer. Moreover, we show that online optimization with feedback delay and nonlinear switching costs can be connected with linear control problems more general than IDSR and a class of nonlinear control problems. Thus iROBD immediately provides novel competitive policies in those cases. This section is mainly based on (Pan et al., 2022).

### **An Online Optimization Model Considering Nonlinearity and Delay**

To incorporate feedback delay, we consider a situation where the online learner only knows the geometry of the hitting cost function at each round, i.e.,  $h_t$ , but that the minimizer of  $f_t$  is revealed only after a delay of  $k$  steps, i.e., at time  $t + k$ . This captures practical scenarios where the form of the loss function or tracking function is known by the online learner, but the target moves over time and measurement lag means that the position of the target is not known until some time after an action must be taken. To incorporate nonlinear (and potentially nonconvex) switching costs, we consider a known nonlinear function  $\delta$  from  $\mathbb{R}^{d \times p}$  to  $\mathbb{R}^d$  in the switching cost. Specifically, we have

$$c(y_{t:t-p}) = \frac{1}{2} \|y_t - \delta(y_{t-1:t-p})\|^2. \quad (10.8)$$

In summary, we consider an online agent that interacts with the environment as follows:

1. The adversary reveals a function  $h_t$ , which is the geometry of the  $t^{\text{th}}$  hitting cost, and a point  $v_{t-k}$ , which is the minimizer of the  $(t - k)^{\text{th}}$  hitting cost. Assume that  $h_t$  is  $m$ -strongly convex and  $l$ -strongly smooth, and that  $\arg \min_y h_t(y) = 0$ .
2. The online learner picks  $y_t$  as its decision point at time step  $t$  after observing  $h_t, v_{t-k}$ .
3. The adversary picks the minimizer of the hitting cost at time step  $t$ :  $v_t$ .
4. The learner pays hitting cost  $f_t(y_t) = h_t(y_t - v_t)$  and switching cost  $c(y_{t:t-p})$  of the form (10.8).

### **Iterative ROBD Algorithm**

Iterative ROBD (iROBD, Algorithm 10.4) is the first competitive algorithm for online optimization with multi-step delay and nonlinear switching costs. Generally

**Algorithm 10.4:** Iterative ROBD (iROBD)

---

```

1: Parameter:  $\lambda \geq 0$ 
2: Initialize a ROBD instance with  $\lambda_1 = \lambda, \lambda_2 = 0$ 
3: for  $t = 1$  to  $T$  do
4:   Input:  $h_t, v_{t-k}$ 
5:   Observe  $f_{t-k}(y) = h_{t-k}(y - v_{t-k})$ 
6:    $\hat{y}_{t-k} = \text{ROBD}(f_{t-k}, \hat{y}_{t-k-p:t-k-1})$ 
7:   Initialize a temporary sequence  $s_{1:t}$ 
8:    $s_{1:t-k} \leftarrow \hat{y}_{1:t-k}$ 
9:   for  $i = t - k + 1$  to  $t$  do
10:     $\tilde{v}_i = \arg \min_v \min_y h_i(y - v) + \lambda c(y, s_{i-1:i-p})$ 
11:    Set  $\tilde{f}_i(y) = h_i(y - \tilde{v}_i)$ 
12:     $s_i \leftarrow \text{ROBD}(\tilde{f}_i, s_{i-p:i-1})$ 
13:   end for
14:    $y_t = s_t$ 
15:   Output:  $y_t$  (the action at time step  $t$ )
16: end for

```

---

speaking, the design of iROBD deals with the  $k$ -round delay via a novel iterative process of estimating the unknown cost function optimistically, i.e., iteratively assuming that the unknown cost functions will lead to minimal cost for the algorithm. This approach is different than a one-shot optimistic approach focused on the whole trajectory of unknown cost functions, and the iterative nature is crucial for bounding the competitive ratio. More interpretations can be found in Pan et al. (2022). iROBD enjoys the following competitive ratio bound:

**Theorem 10.4.** *Suppose the hitting costs are  $m$ -strongly convex and  $l$ -strongly smooth, and the switching cost is given by  $c(y_{t:t-p}) = \frac{1}{2} \|y_t - \delta(y_{t-1:t-p})\|^2$ , where  $\delta : \mathbb{R}^{d \times p} \rightarrow \mathbb{R}^d$ . If there is a  $k$ -round-delayed feedback on the minimizers, and for any  $1 \leq i \leq p$  there exists a constant  $L_i > 0$ , such that for any given  $y_{t-1}, \dots, y_{t-i-1}, y_{t-i+1}, \dots, y_{t-p} \in \mathbb{R}^d$ , we have:*

$$\|\theta(a) - \theta(b)\| \leq L_i \|a - b\|, \forall a, b \in \mathbb{R}^d,$$

where  $\theta(x) = \delta(y_{t-1}, \dots, y_{t-i-1}, x, y_{t-i+1}, \dots, y_{t-p})$ , then the competitive ratio of iROBD( $\lambda$ ) is bounded by

$$O \left( (l + 2p^2 L^2)^k \max \left\{ \frac{1}{\lambda}, \frac{m + \lambda}{m + (1 - p^2 L^2) \lambda} \right\} \right),$$

where  $L = \max_i \{L_i\}$ ,  $\lambda > 0$  and  $m + (1 - p^2 L^2) \lambda > 0$ .

The detailed analysis and a proof can be found in Pan et al. (2022). The key idea to our competitive ratio proof is to bound the error that accumulates in the iterations by leveraging a Lipschitz property on the nonlinear component of the switching cost. This analytic approach is novel and a contribution in its own right.

Note that the competitive ratio of iROBD in Theorem 10.4 is  $O(L^{2k})$ . In other words, feedback delay ( $k$ ) leads to an exponential degradation of the competitive ratio while memory ( $p$ ) and the Lipschitz constant of the nonlinear switching cost ( $L$ ) impact the competitive ratio only in a polynomial manner. Thus, one may wonder if this dependence is a function of the iROBD algorithm or if it is fundamental. In Pan et al. (2022), we provide a  $\Omega(L^{2k})$  to show that it is a fundamental limit.

### Connections to Competitive Control

As expected, online optimization with feedback delay and nonlinear switching costs can be connected with much more general control problems than classic SOCO or OCO with structured memory settings. In particular, we show two examples here. For both examples, we can show exact reductions from them to online optimization with delay and nonlinearity, and iROBD immediately provides competitive policies. For detailed reductions and analysis, we refer to Pan et al. (2022).

**Linear systems with matched and unmatched disturbance.** We consider

$$\begin{aligned} \min_{u_t} \quad & \sum_{t=1}^T \frac{q_t}{2} \|x_t\|^2 + \sum_{t=0}^{T-1} \frac{1}{2} \|u_t\|^2 \\ \text{s.t.} \quad & x_{t+1} = Ax_t + Bu_t + w_t, \end{aligned}$$

where  $(A, B)$  are in controllable canonical form and  $w_t$  is a potentially adversarial disturbance. Note that it considers both matched and *unmatched* disturbance while Section 10.4 only considers the matched disturbance setting  $B(u_t + w_t)$ .

**Nonlinear systems with delay and time-variant costs.** Consider the following class of online nonlinear control problems:

$$\begin{aligned} \min_{u_t} \quad & \sum_{t=1}^T f_t(x_t) + \sum_{t=0}^{T-1} \frac{1}{2} \|u_t\|^2 \\ \text{s.t.} \quad & x_{t+1} = Ax_t + u_t + g(x_t) \end{aligned}$$

where  $\{f_t\}_{t=1}^T$  are time-variant well-conditioned costs (e.g., trajectory tracking costs), and  $g(x_t)$  is the nonlinear dynamics term. At time step  $t$ , only  $f_{1:t-k}$  is known due to communication delays. Many robotic systems can be viewed as special cases of this

form, such as pendulum dynamics and quadrotor position dynamics (see Chapter 2). It is immediate to see that, by defining  $y_t = x_t$ , this online control problem can be converted into an online optimization problem with hitting cost  $f_t$  and nonlinear switching cost  $c(y_t, y_{t-1}) = \frac{1}{2} \|y_t - Ay_{t-1} - g(y_{t-1})\|^2$ .

## 10.7 Appendix

### Proof of Theorem 10.1

Our approach is to make use of strong convexity and properties of the hitting cost, the switching cost, and the regularization term to derive an inequality in the form of  $H_t + M_t + \Delta\phi_t \leq C(H_t^* + M_t^*)$  for some positive constant  $C$ , where  $\Delta\phi_t$  is the change in potential, which satisfies  $\sum_{t=1}^T \Delta\phi_t \geq 0$ . We will give the formal definition of  $\Delta\phi_t$  later. The constant  $C$  is then an upper bound for the competitive ratio.

By assumption, we have  $y_i = y_i^*$  for  $i = 0, -1, \dots, -(p-1)$ .

For convenience, we define

$$\phi_t = \frac{\lambda_1 + \lambda_2 + m}{2} \|y_t - y_t^*\|^2.$$

Recall that we define  $v_t = \arg \min_y f_t(y)$ . Since the function

$$g_t(y) = f_t(y) + \frac{\lambda_1}{2} \left\| y - \sum_{i=1}^p C_i y_{t-i} \right\|^2 + \frac{\lambda_2}{2} \|y - v_t\|^2$$

is  $(m + \lambda_1 + \lambda_2)$ -strongly convex and ROBD selects  $y_t = \arg \min_y g_t(y)$ , we see that

$$g_t(y_t) + \frac{m + \lambda_1 + \lambda_2}{2} \|y_t - y_t^*\|^2 \leq g_t(y_t^*),$$

which implies

$$\begin{aligned} & H_t + \lambda_1 M_t + \left( \phi_t - \sum_{i=1}^p \frac{\|C_i\|}{\alpha} \phi_{t-i} \right) \\ & \leq \left( H_t^* + \frac{\lambda_2}{2} \|y_t^* - v_t\|^2 \right) + \left( \frac{\lambda_1}{2} \left\| y_t^* - \sum_{i=1}^p C_i y_{t-i} \right\|^2 - \sum_{i=1}^p \frac{\|C_i\|}{\alpha} \phi_{t-i} \right). \end{aligned} \quad (10.9)$$

In the following steps, we bound the second term in the right-hand side of (10.9) by

the switching cost of the offline optimal.

$$\begin{aligned}
& \sum_{i=1}^p \frac{\|C_i\|}{\alpha} \phi_{t-i} \\
&= \frac{\lambda_1 + \lambda_2 + m}{2\alpha} \sum_{i=1}^p \|C_i\| \cdot \|y_{t-i} - y_{t-i}^*\|^2 \\
&\geq \frac{\lambda_1 + \lambda_2 + m}{2\alpha^2} \left( \sum_{i=1}^p \|C_i\| \cdot \|y_{t-i} - y_{t-i}^*\| \right)^2 \tag{10.10a}
\end{aligned}$$

$$\geq \frac{\lambda_1 + \lambda_2 + m}{2\alpha^2} \left( \sum_{i=1}^p \|C_i y_{t-i} - C_i y_{t-i}^*\| \right)^2 \tag{10.10b}$$

$$\geq \frac{\lambda_1 + \lambda_2 + m}{2\alpha^2} \left\| \sum_{i=1}^p C_i y_{t-i} - \sum_{i=1}^p C_i y_{t-i}^* \right\|^2, \tag{10.10c}$$

where we use Jensen's Inequality in (10.10a); the definition of the matrix norm in (10.10b); the triangle inequality in (10.10c).

For notation convenience, we define

$$\delta_t = \sum_{i=1}^p C_i y_{t-i} - \sum_{i=1}^p C_i y_{t-i}^*.$$

Therefore, we obtain that

$$\begin{aligned} & \frac{\lambda_1}{2} \left\| y_t^* - \sum_{i=1}^p C_i y_{t-i} \right\|^2 - \sum_{i=1}^p \frac{\|C_i\|}{\alpha} \phi_{t-i} \\ & \leq \frac{\lambda_1}{2} \left\| y_t^* - \sum_{i=1}^p C_i y_{t-i} \right\|^2 - \frac{\lambda_1 + \lambda_2 + m}{2\alpha^2} \cdot \|\delta_t\|^2 \end{aligned} \quad (10.11a)$$

$$\begin{aligned} & = \frac{\lambda_1}{2} \left\| \left( y_t^* - \sum_{i=1}^p C_i y_{t-i}^* \right) - \delta_t \right\|^2 - \frac{\lambda_1 + \lambda_2 + m}{2\alpha^2} \cdot \|\delta_t\|^2 \\ & \leq \frac{\lambda_1}{2} \left\| y_t^* - \sum_{i=1}^p C_i y_{t-i}^* \right\|^2 + \lambda_1 \left\| y_t^* - \sum_{i=1}^p C_i y_{t-i}^* \right\| \cdot \|\delta_t\| \\ & \quad + \frac{\lambda_1}{2} \|\delta_t\|^2 - \frac{\lambda_1 + \lambda_2 + m}{2\alpha^2} \|\delta_t\|^2 \end{aligned} \quad (10.11b)$$

$$\begin{aligned} & = \frac{\lambda_1}{2} \left\| y_t^* - \sum_{i=1}^p C_i y_{t-i}^* \right\|^2 + \lambda_1 \left\| y_t^* - \sum_{i=1}^p C_i y_{t-i}^* \right\| \cdot \|\delta_t\| \\ & \quad - \frac{(1 - \alpha^2)\lambda_1 + \lambda_2 + m}{2\alpha^2} \|\delta_t\|^2 \\ & \leq \frac{\lambda_1}{2} \left\| y_t^* - \sum_{i=1}^p C_i y_{t-i}^* \right\|^2 + \frac{\alpha^2 \lambda_1^2}{2((1 - \alpha^2)\lambda_1 + \lambda_2 + m)} \left\| y_t^* - \sum_{i=1}^p C_i y_{t-i}^* \right\|^2 \\ & \quad + \frac{(1 - \alpha^2)\lambda_1 + \lambda_2 + m}{2\alpha^2} \|\delta_t\|^2 - \frac{(1 - \alpha^2)\lambda_1 + \lambda_2 + m}{2\alpha^2} \|\delta_t\|^2 \end{aligned} \quad (10.11c)$$

$$= \frac{\lambda_1(\lambda_1 + \lambda_2 + m)}{(1 - \alpha^2)\lambda_1 + \lambda_2 + m} M_t^*,$$

where we use (10.10) in (10.11a); the triangle inequality in (10.11b); the AM-GM inequality in (10.11c).

We also notice that since  $f_t$  is  $m$ -strongly convex, the first term in the right-hand side of (10.9) can be bounded by

$$H_t^* + \frac{\lambda_2}{2} \|y_t^* - v_t\|^2 \leq \frac{m + \lambda_2}{m} H_t^*. \quad (10.12)$$

Substituting (10.11) and (10.12) into (10.9), we obtain that

$$\begin{aligned} & H_t + \lambda_1 M_t + \phi_t - \sum_{i=1}^q \frac{\|C_i\|}{\alpha} \phi_{t-i} \\ & \leq \frac{m + \lambda_2}{m} H_t^* + \frac{\lambda_1(\lambda_1 + \lambda_2 + m)}{(1 - \alpha^2)\lambda_1 + \lambda_2 + m} M_t^*. \end{aligned} \quad (10.13)$$



Define  $\Delta\phi_t = \phi_t - \sum_{i=1}^q \frac{\|C_i\|}{\alpha} \phi_{t-i}$ . We see that

$$\sum_{t=1}^T \Delta\phi_t = \frac{1}{\alpha} \sum_{i=0}^{q-1} \left( \sum_{j=i+1}^q \|C_j\| \right) \phi_{T-i} - \frac{1}{\alpha} \sum_{i=0}^{q-1} \left( \sum_{j=i+1}^q \|C_j\| \right) \phi_{-i}.$$

Since  $\phi_t \geq 0, \forall t$  and  $\phi_0 = \phi_{-1} = \dots = \phi_{-q+1} = 0$ , we have

$$\sum_{t=1}^T \Delta\phi_t \geq 0. \quad (10.14)$$

Summing (10.13) over timesteps  $t = 1, 2, \dots, T$ , we see that

$$\sum_{t=1}^T (H_t + \lambda_1 M_t) + \sum_{t=1}^T \Delta\phi_t \leq \sum_{t=1}^T \left( \frac{m + \lambda_2}{m} H_t^* + \frac{\lambda_1(\lambda_1 + \lambda_2 + m)}{(1 - \alpha^2)\lambda_1 + \lambda_2 + m} M_t^* \right).$$

Using (10.14), we obtain that

$$\sum_{t=1}^T (H_t + \lambda_1 M_t) \leq \sum_{t=1}^T \left( \frac{m + \lambda_2}{m} H_t^* + \frac{\lambda_1(\lambda_1 + \lambda_2 + m)}{(1 - \alpha^2)\lambda_1 + \lambda_2 + m} M_t^* \right), \quad (10.15)$$

which implies

$$\sum_{t=1}^T (H_t + M_t) \leq \sum_{t=1}^T \left( \frac{m + \lambda_2}{m\lambda_1} H_t^* + \frac{\lambda_1 + \lambda_2 + m}{(1 - \alpha^2)\lambda_1 + \lambda_2 + m} M_t^* \right).$$

## References

- Abbasi-Yadkori, Yasin, Nevena Lazic, and Csaba Szepesvári (2018). *Regret bounds for model-free linear quadratic control*. In: *arXiv preprint arXiv:1804.06021*.
- Abeille, Marc and Alessandro Lazaric (2018). *Improved regret bounds for thompson sampling in linear quadratic control problems*. In: *International Conference on Machine Learning (ICML)*.
- Agarwal, Naman, Nataly Brukhim, Elad Hazan, and Zhou Lu (2019). *Boosting for dynamical systems*. In: *arXiv preprint arXiv:1906.08720*.
- Agarwal, Naman, Brian Bullins, Elad Hazan, Sham M. Kakade, and Karan Singh (2019). *Online control with adversarial disturbances*. In: *International Conference on Machine Learning (ICML)*.
- Agarwal, Naman, Elad Hazan, and Karan Singh (2019). *Logarithmic regret for online control*. In: *Neural Information Processing Systems (NeurIPS)*.
- Anava, Oren, Elad Hazan, and Shie Mannor (2015). *Online learning for adversaries with memory: Price of past mistakes*. In: *Advances in Neural Information Processing Systems*, pp. 784–792.

- Andrew, Lachlan, Siddharth Barman, Katrina Ligett, Minghong Lin, Adam Meyerson, Alan Roytman, and Adam Wierman (2013). *A tale of two metrics: Simultaneous bounds on competitiveness and regret*. In: *Conference on Learning Theory*, pp. 741–763.
- Argue, C. J., Anupam Gupta, Guru Guruganesh, and Ziyue Tang (2020). *Chasing convex bodies with linear competitive ratio*. In: *SIAM Symposium on Discrete Algorithms (SODA)*.
- Badiei, Masoud, Na Li, and Adam Wierman (2015). *Online convex optimization with ramp constraints*. In: *IEEE Conference on Decision and Control (CDC)*.
- Bansal, Nikhil, Anupam Gupta, Ravishankar Krishnaswamy, Kirk Pruhs, Kevin Schewior, and Cliff Stein (2015). *A 2-competitive algorithm for online convex optimization with switching costs*. In: *Proceedings of the Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*.
- Bubeck, Sébastien, Bo'az Klartag, Yin Tat Lee, Yuanzhi Li, and Mark Sellke (2020). *Chasing nested convex bodies nearly optimally*. In: *SIAM Symposium on Discrete Algorithms (SODA)*.
- Bubeck, Sébastien, Yin Tat Lee, Yuanzhi Li, and Mark Sellke (2019). *Competitively chasing convex bodies*. In: *ACM Symposium on Theory of Computing (STOC)*.
- Chen, Niangjun, Anish Agarwal, Adam Wierman, Siddharth Barman, and Lachlan Andrew (2015). *Online convex optimization using predictions*. In: *Proceedings of ACM SIGMETRICS*, pp. 191–204.
- Chen, Niangjun, Joshua Comden, Zhenhua Liu, Anshul Gandhi, and Adam Wierman (2016). *Using predictions in online optimization: Looking forward with an eye on the past*. In: *Proceedings of ACM SIGMETRICS*, pp. 193–206.
- Chen, Niangjun, Gautam Goel, and Adam Wierman (2018). *Smoothed online convex optimization in high dimensions via online balanced descent*. In: *Proceedings of Conference On Learning Theory (COLT)*, pp. 1574–1594.
- Chen, Shiyao and Lang Tong (2012). *iEMS for large scale charging of electric vehicles: Architecture and optimal online scheduling*. In: *2012 IEEE Third International Conference on Smart Grid Communications (SmartGridComm)*. IEEE, pp. 629–634.
- Cohen, Alon, Avinatan Hasidim, Tomer Koren, Nevena Lazic, Yishay Mansour, and Kunal Talwar (2018). *Online linear quadratic control*. In: *International Conference on Machine Learning*. Proceedings of Machine Learning Research, pp. 1029–1038.
- Dean, Sarah, Horia Mania, Nikolai Matni, Benjamin Recht, and Stephen Tu (2020). *On the sample complexity of the linear quadratic regulator*. In: *Foundations of Computational Mathematics* 20.4, pp. 633–679.

- Goel, Gautam and Babak Hassibi (2020). *The power of linear controllers in LQR control*. In: *arXiv preprint arXiv:2002.02574*.
- Goel, Gautam, Yiheng Lin, Haoyuan Sun, and Adam Wierman (2019). *Beyond online balanced descent: An optimal algorithm for smoothed online optimization*. In: *Neural Information Processing Systems (NeurIPS)*.
- Goel, Gautam and Adam Wierman (2019). *An online algorithm for smoothed regression and LQR control*. In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*.
- Hazan, Elad (2019). *Introduction to online convex optimization*. In: *arXiv preprint arXiv:1909.05207*.
- Joulani, Pooria, András György, and Csaba Szepesvári (2013). *Online learning under delayed feedback*. In: *International Conference on Machine Learning*. Proceedings of Machine Learning Research, pp. 1453–1461.
- Kim, Seung-Jun and Geogios B. Giannakis (2016). *An online convex optimization approach to real-time energy pricing for demand response*. In: *IEEE Transactions on Smart Grid* 8.6, pp. 2784–2793.
- Kirk, Donald E. (2004). *Optimal control theory: An introduction*. Courier Corporation.
- Lale, Sahin, Kamyar Azizzadenesheli, Babak Hassibi, and Anima Anandkumar (2020). *Logarithmic regret bound in partially observable linear dynamical systems*. In: *arXiv preprint arXiv:2003.11227*.
- Li, Yingying, Xin Chen, and Na Li (2019). *Online optimal control with linear dynamics and predictions: Algorithms and regret analysis*. In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 14858–14870.
- Li, Yingying, Guannan Qu, and Na Li (2018a). *Online optimization with predictions and switching costs: Fast algorithms and the fundamental limit*. In: *arXiv preprint arXiv:1801.07780*.
- (2018b). *Using predictions in online optimization with switching costs: A fast algorithm and a fundamental limit*. In: *2018 Annual American Control Conference (ACC)*. IEEE, pp. 3008–3013.
- Lin, Minghong, Zhenhua Liu, Adam Wierman, and Lachlan Andrew (2012). *Online algorithms for geographical load balancing*. In: *Proceedings of the International Green Computing Conference (IGCC)*, pp. 1–10.
- Lin, Minghong, Adam Wierman, Lachlan Andrew, and Eno Thereska (2013). *Dynamic right-sizing for power-proportional data centers*. In: *IEEE/ACM Transactions on Networking (TON)* 21.5, pp. 1378–1391.

- Lin, Qiulin, Hanling Yi, John Pang, Minghua Chen, Adam Wierman, Michael Honig, and Yuanzhang Xiao (2019). *Competitive online optimization under inventory constraints*. In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 3.1, pp. 1–28.
- Lin, Yiheng, Gautam Goel, and Adam Wierman (2020). *Online optimization with predictions and non-convex losses*. In: *Proceedings of ACM SIGMETRICS*.
- Luenberger, David (1967). *Canonical forms for linear multivariable systems*. In: *IEEE Transactions on Automatic Control* 12.3, pp. 290–293.
- Pan, Weici, Guanya Shi, Yiheng Lin, and Adam Wierman (2022). *Online optimization with feedback delay and nonlinear switching cost*. In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 6.1, pp. 1–34. DOI: 10.1145/3508037.
- Sellke, Mark (2020). *Chasing convex bodies optimally*. In: *SIAM Symposium on Discrete Algorithms (SODA)*.
- Shamir, Ohad and Liran Szlak (2017). *Online learning with local permutations and delayed feedback*. In: *International Conference on Machine Learning*. Proceedings of Machine Learning Research, pp. 3086–3094.
- Shi, Guanya, Yiheng Lin, Soon-Jo Chung, Yisong Yue, and Adam Wierman (2020). *Online optimization with memory and competitive control*. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 33. Curran Associates, Inc. URL: <https://arxiv.org/abs/2002.05318>.
- Yang, Lin, Mohammad H. Hajiesmaili, Ramesh Sitaraman, Enrique Mallada, Wing S. Wong, and Adam Wierman (2020). *Online inventory management with application to energy procurement in data centers*. In: *Proceedings of ACM Sigmetrics*.
- Zhou, Kemin and John Comstock Doyle (1998). *Essentials of robust control*. Vol. 104. Prentice Hall Upper Saddle River, NJ.

## ONLINE LEARNING PERSPECTIVES ON MODEL PREDICTIVE CONTROL

Another critical question is begged in online learning and control: Do established control methods have strong learning guarantees? In particular, Model Predictive Control (MPC) has been one of the most successful methods in industrial control since the 1980s. However, many learning theorists are studying RL algorithms, but few are analyzing MPC and why it is so powerful. To close this gap, this chapter studies MPC from learning-theoretic perspectives, and proves the first non-asymptotic guarantee for MPC, showing that MPC is *near-optimal* in the sense of dynamic regret in online LQR control with predictable disturbance. This chapter also extends to settings with inexact predictions and linear time-variant (LTV) systems. These results found common ground for learning and control theory and imply fundamental algorithmic principles. This chapter is mainly based on the following papers<sup>1</sup>:

- Li, Tongxin, Ruixiao Yang, Guannan Qu, Guanya Shi, Chenkai Yu, Adam Wierman, and Steven Low (2022). *Robustness and consistency in linear quadratic control with untrusted predictions*. In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 6.1, pp. 1–35. DOI: [10.1145/3508038](https://doi.org/10.1145/3508038).
- Lin, Yiheng, Yang Hu, Guanya Shi, Haoyuan Sun, Guannan Qu, and Adam Wierman (2021). *Perturbation-based regret analysis of predictive control in linear time varying systems*. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 34. Curran Associates, Inc., pp. 5174–5185. URL: <https://arxiv.org/abs/2106.10497>.
- Yu, Chenkai, Guanya Shi, Soon-Jo Chung, Yisong Yue, and Adam Wierman (2020). *The power of predictions in online control*. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 33. Curran Associates, Inc., pp. 1994–2004. URL: <https://proceedings.neurips.cc/paper/2020/file/155fa09596c7e18e50b58eb7e0c6ccb4-Paper.pdf>.
- (2022). *Competitive control with delayed imperfect information*. In: *American Control Conference (ACC)*. URL: <https://arxiv.org/abs/2010.11637>.

---

<sup>1</sup>Blog post: <https://www.gshi.me/blog/CompetitiveMPC/>

## 11.1 Introduction

This chapter studies the effect of using predictions for online control in a linear dynamical system governed by  $x_{t+1} = A_t x_t + B_t u_t + w_t$ , where  $x_t$ ,  $u_t$ , and  $w_t$  are the state, control, and disturbance (or exogenous input), respectively. At each time step  $t$ , the controller incurs a cost  $c_t(x_t, u_t)$ . Recently, considerable effort has been made to leverage and integrate ideas from learning, optimization and control theory to study the design of optimal controllers under various performance criteria, such as static regret (Dean et al., 2020; Agarwal, Bullins, et al., 2019; Agarwal, Hazan, et al., 2019; Cohen et al., 2019; Hazan et al., 2020; D. J. Foster and Simchowitz, 2020; Simchowitz and D. Foster, 2020), dynamic regret (Y. Li et al., 2019; Goel and Hassibi, 2020) and competitive ratio (Chapter 10 and Goel and Wierman (2019)). However, the study of online convergence when incorporating predictions has been largely absent.

Indeed, a key aspect of online control is considering the amount of available information when making decisions. Most recent studies focus on the basic setting where only past information,  $x_0, w_0, \dots, w_{t-1}$ , is available for  $u_t$  at every time step (Dean et al., 2020; Agarwal, Bullins, et al., 2019; D. J. Foster and Simchowitz, 2020). However, this basic setting does not effectively characterize situations where we have accurate predictions, e.g., when  $x_0, w_0, \dots, w_{t-1+k}$  are available at step  $t$ . These types of accurate predictions are often available in many applications, including robotics (Baca et al., 2018), energy systems (Vazquez et al., 2016), and data center management (Lazic et al., 2018). Moreover, there are many practical algorithms that leverage predictions, such as the popular Model Predictive Control (MPC) (Baca et al., 2018; Camacho and Alba, 2013; Angeli, Casavola, et al., 2016; Angeli, Amrit, et al., 2011; Grüne and Pirkelmann, 2018; Grüne and Stieler, 2014).

While there has been increased interest in studying online guarantees for control with predictions, to our knowledge, there has been no such study for the case of a finite-time horizon with disturbances. Several previous works studied the economic MPC problem by analyzing the asymptotic performance without disturbances (Angeli, Casavola, et al., 2016; Angeli, Amrit, et al., 2011; Grüne and Pirkelmann, 2018; Grüne and Stieler, 2014). Rosolia and Borrelli (2019) and Rosolia and Borrelli (2017) studied learning for MPC but focused on the episodic setting with asymptotic convergence guarantees. Y. Li et al. (2019) considered a linear system where finite predictions of costs are available, and analyzed the dynamic regret of their new algorithm; however, they neither consider disturbances nor study the more practically

relevant MPC approach. Goel and Hassibi (2020) characterized the offline optimal policy (i.e., with infinite predictions) and cost in LQR control with i.i.d. zero-mean stochastic disturbances, but those results do not apply to limited predictions or non-i.i.d. disturbances. Other prior works analyze the power of predictions in online optimization (Lin, Goel, et al., 2020; Chen et al., 2015), but the connection to online control in dynamical systems is unclear.

From this literature, fundamental questions about online control with predictions have emerged:

1. *What are the cost-optimal and regret-minimizing policies when given  $k$  predictions? What are the corresponding cost and regret of these policies?*
2. *What is the marginal benefit from each additional prediction used by the policy, and how many predictions are needed to achieve (near-)optimal performance?*
3. *How well does MPC with  $k$  predictions perform compared to cost-optimal and regret-minimizing policies?*

### **Contributions and Organizations**

In Sections 11.2 to 11.4, we systematically address each of the questions above in the context of LQR systems with general stochastic and adversarial disturbances in the dynamics. In the stochastic case, we explicitly derive the cost-optimal and dynamic regret minimizing policies with  $k$  predictions. In both the stochastic and adversarial cases, we derive (mostly tight) upper bounds for the optimal cost and minimum dynamic regret given access to  $k$  predictions. We also show that the marginal benefit of an extra prediction exponentially decays as  $k$  increases. We further show that MPC is near-optimal in terms of dynamic regret, and needs only  $O(\log T)$  predictions to achieve  $O(1)$  dynamic regret (the same order as is needed by the dynamic regret minimizing policy) in both settings.

Then in Section 11.5, we generalize the analysis framework and results from perfect predictions to settings with *inexact predictions* and *delays*. Section 11.5 also analyzes MPC's competitive ratios.

Finally in Section 11.6, we generalize the setting from standard LQR to Linear Time-Variant (LTV) systems with general well-conditioned costs.

Perhaps surprisingly, for all these settings (LQR, inexact predictions, or LTV), we have shown that classic MPC, which is a simple greedy policy (up to the prediction horizon), is near-optimal even with adversarial disturbances in the dynamics. Our results thus highlight the power of predictions to reduce the need for algorithmic sophistication. In that sense, our results somewhat mirror recent developments in the study of exploration strategies in online LQR control with unknown dynamics  $\{A, B\}$ : after a decade's research beginning with the work of Abbasi-Yadkori and Szepesvári (2011), Simchowicz and D. Foster (2020) recently show that naive exploration is optimal. Taken together with the result from Simchowicz and D. Foster (2020), we provide additional evidence for the idea that the structure of LQR allows simple algorithmic ideas to be effective, which sheds light on key algorithmic principles and fundamental limits in continuous control.

## 11.2 Problem Statement

Throughout Sections 11.2 to 11.4, we consider the *Linear Quadratic Regulator (LQR)* optimal control problem with disturbances in the dynamics (introduced in Chapter 8). In particular, we consider a linear system initialized with  $x_0 \in \mathbb{R}^n$  and controlled by  $u_t \in \mathbb{R}^d$ , with dynamics

$$x_{t+1} = Ax_t + Bu_t + w_t \quad \text{and cost} \quad J = \sum_{t=0}^{T-1} (x_t^\top Q x_t + u_t^\top R u_t) + x_T^\top Q_f x_T,$$

where  $T \geq 1$  is the total length of the control period. The goal of the controller is to minimize the cost given  $A, B, Q, R, Q_f, x_0$ , and the characterization of the disturbance  $w_t$ . We make the following assumption about the system.

**Assumption 11.1.** *We assume  $Q, Q_f \geq 0, R > 0$  and the pair  $(A, B)$  is stabilizable, i.e., there exists a matrix  $K_0 \in \mathbb{R}^{d \times n}$  such that  $\rho(A - BK_0) < 1$ . Further, we assume the pair  $(A, Q)$  is detectable, i.e.,  $(A^\top, Q)$  is stabilizable, to guarantee stability of the closed-loop. Note that detectability of  $(A, Q)$  is more general than  $Q > 0$ , i.e.,  $Q > 0$  implies  $(A, Q)$  is detectable. For  $w_t$ , in the stochastic case, we assume  $\{w_t\}_{t=0,1,\dots}$  are sampled from a joint distribution with bounded cross-correlation, i.e.,  $\mathbb{E}[w_t^\top w_{t'}] \leq m$  for any  $t, t'$ ; in the adversarial case, we assume  $w_t$  is picked from a bounded set  $\Omega$  and  $\|w_t\| \leq r$ .*

**Generality.** These are standard assumptions in the literature, e.g., Dean et al. (2020), D. J. Foster and Simchowicz (2020), and Simchowicz and D. Foster (2020)



and it is worth noting that our notion of stochasticity is much more general than typically considered (Dean et al., 2020; Cohen et al., 2019; Cassel et al., 2020). We also note that many important problems can be straightforwardly converted to our model. For example, input-disturbed systems and the Linear Quadratic (LQ) tracking problem (B. D. O. Anderson and Moore, 2007).

**Example 11.1** (Linear quadratic tracking). *The standard quadratic tracking problem is defined with dynamics  $x_{t+1} = Ax_t + Bu_t + \tilde{w}_t$  and cost function  $J = \sum_{t=0}^{T-1} (x_{t+1} - d_{t+1})^\top Q(x_{t+1} - d_{t+1}) + u_t^\top Ru_t$ , where  $\{d_t\}_{t=1}^T$  is the desired trajectory to track. To map this to our model, let  $\tilde{x}_t = x_t - d_t$ . Then, we get  $J = \sum_{t=0}^{T-1} \tilde{x}_{t+1}^\top Q\tilde{x}_{t+1} + u_t^\top Ru_t$  and  $\tilde{x}_{t+1} = A\tilde{x}_t + Bu_t + w_t$ , which is an LQR control problem with disturbance  $w_t = \tilde{w}_t + Ad_t - d_{t+1}$  in the dynamics.*

### Predictions

In the classic model, at each step  $t$ , the controller decides  $u_t$  after observing  $w_{t-1}$  and  $x_t$ . In other words,  $u_t$  is a function of all the previous information:  $x_0, x_1, \dots, x_{t-1}$  and  $w_0, w_1, \dots, w_{t-1}$ , or equivalently, of  $x_0, w_0, w_1, \dots, w_{t-1}$ . We describe this scenario via the following *event sequence*:

$$x_0 \quad u_0 \quad w_0 \quad u_1 \quad w_1 \quad \cdots \quad u_{T-1} \quad w_{T-1},$$

where each  $u_t$  denotes the decision of a control policy, each  $w_t$  denote the observation of a disturbance, and each decision may depend on previous events.

However, in many real-world applications the controller may have some knowledge about future. In particular, at time step  $t$ , the controller may have *predictions* of immediate  $k$  future disturbances and make decision  $u_t$  based on  $x_0, w_0, \dots, w_{t+k-1}$ . In this case, the event sequence is given by:

$$x_0 \quad w_0 \quad \cdots \quad w_{k-1} \quad u_0 \quad w_k \quad u_1 \quad w_{k+1} \quad \cdots \quad u_{T-k-1} \quad w_{T-1} \quad u_{T-k} \quad \cdots \quad u_{T-1}.$$

The existence of predictions is common in many applications such as disturbance estimation in robotics and model predictive control (MPC) (Camacho and Alba, 2013), which is a common approach for the LQ tracking problem. When given  $k$  predictions of  $d_t$ , the LQ tracking problem can be formulated as a LQR problem with  $k - 1$  predictions of future disturbances. Throughout Sections 11.2 to 11.4 we assume all the predictions are *exact*, and discuss inexact predictions in Section 11.5.

## Disturbances

The characteristics of the disturbances have a fundamental impact on the optimal control policy and cost. We consider two types of disturbance: stochastic disturbances, which are drawn from a joint distribution (not necessarily i.i.d.), and adversarial disturbances, which are chosen by an adversary to maximize the overall control cost of the policy.

In the stochastic setting, we model the disturbance sequence  $\{w_t\}_{t=0}^{T-1}$  as a discrete-time stochastic process with joint distribution  $\mathcal{W}$  which is known to the controller. Let  $W_t = W_t(w_0, \dots, w_{t-1})$  be the conditional distribution of  $w_t$  given  $w_0, \dots, w_{t-1}$ . Then the cost of the optimal online policy with  $k$  predictions is given by:

$$STO_k^T = \mathbb{E}_{w_0 \sim W_0, \dots, w_{k-1} \sim W_{k-1}} \left( \min_{u_0} \left( \mathbb{E}_{w_k \sim W_k} \left( \dots \min_{u_{T-k-1}} \left( \mathbb{E}_{w_{T-1} \sim W_{T-1}} \left( \min_{u_{T-k}, \dots, u_{T-1}} J \right) \right) \right) \right) \right).$$

Note that the cost  $J = J(x_0, u_0, \dots, u_{T-1}, w_0, \dots, w_{T-1})$ . Two extreme cases are noteworthy:  $k = 0$  reduces to the classic case without prediction (in Chapter 8) and  $k = T$  reduces to the offline optimal.

In the adversarial setting, each disturbance  $w_t$  is selected by an adversary from a bounded set  $\Omega \subseteq \mathbb{R}^n$  in order to maximize the cost. The controller has no information about the disturbance except that it is in  $\Omega$ . Similar to the stochastic setting, we define:

$$ADV_k^T = \sup_{w_0, \dots, w_{k-1} \in \Omega} \left( \min_{u_0} \left( \sup_{w_k \in \Omega} \left( \dots \min_{u_{T-k-1}} \left( \sup_{w_{T-1} \in \Omega} \left( \min_{u_{T-k}, \dots, u_{T-1}} J \right) \right) \right) \right) \right).$$

This can be viewed as online  $\mathcal{H}_\infty$  control (Zhou and Doyle, 1998) with predictions.

The average cost in an infinite horizon is particularly important in both control and learning communities to understand asymptotic behaviors. We use separate notation for it:

$$STO_k = \lim_{T \rightarrow \infty} \frac{1}{T} STO_k^T, \quad ADV_k = \lim_{T \rightarrow \infty} \frac{1}{T} ADV_k^T.$$

We emphasize that we do not have any constraints (like linearity) on the policy space, and both  $STO_k^T$  and  $ADV_k^T$  are *globally optimal* with the corresponding type of disturbance. This point is important in light of recent results that show that linear policies cannot make use of predictions at all (see Chapter 10 and Goel and Hassibi (2020)), i.e., the cost of the best linear policy with infinite predictions ( $k = \infty$ ) is asymptotically equal to that with no predictions ( $k = 0$ ) in the setting with i.i.d. zero-mean stochastic disturbances. In this chapter, we explicitly derive the optimal policy for every  $k > 0$ , which is *nonlinear* in general.

## Model Predictive Control

---

**Algorithm 11.1:** Model predictive control with  $k$  predictions

---

**Parameter:**  $\{A, B, Q, R\}$  and  $\tilde{Q}_f \in \mathbb{R}^{n \times n}$

**Input:**  $x_0, w_0, \dots, w_{k-1}$

1 **for**  $t = 0$  **to**  $T - 1$  **do**

2     **Input:**  $x_t, w_{t+k-1}$      // The controller now knows  $x_0, \dots, x_t, w_0, \dots, w_{t+k-1}$   
     $(u_t, \dots, u_{t+k-1}) = \arg \min_u \sum_{i=t}^{t+k-1} (x_i^\top Q x_i + u_i^\top R u_i) + x_{t+k}^\top \tilde{Q}_f x_{t+k}$  subject to  
     $x_{i+1} = A x_i + B u_i + w_i$  for  $i = t, \dots, t + k - 1$   
    **Output:**  $u_t$

---

As introduced in Chapter 8, Model predictive control (MPC) is perhaps the most common control policy for situations where predictions are available. MPC is a greedy algorithm with a receding horizon based on all available current predictions. Algorithm 11.1 provides a formal definition, and we additionally refer the reader to the book by Camacho and Alba (2013) for a literature review on MPC. We adopt a conventional definition of MPC as an online optimal control problem with a finite-time horizon with dynamics constraints. Note that other prior work on MPC sometimes considers other input and state constraints (Camacho and Alba, 2013).

MPC is a practical algorithm in many scenarios like robotics (Baca et al., 2018), energy system (Vazquez et al., 2016) and data center cooling (Lazic et al., 2018). The existing theoretical studies of MPC focus on asymptotic stability and performance (Angeli, Casavola, et al., 2016; Angeli, Amrit, et al., 2011; Grüne and Pirkelmann, 2018; Grüne and Stieler, 2014; Rosolia and Borrelli, 2017). To our knowledge, we provide the first general, dynamic regret guarantee for MPC in this thesis.

In Section 11.3, we study the performance of MPC in three different cases, where disturbances are i.i.d. zero-mean stochastic, generally stochastic, and adversarial. We define the performance of MPC in the stochastic and adversarial settings as follows:

$$\begin{aligned} \text{MPCS}_k^T &= \mathbb{E}_{w_0, \dots, w_{T-1}} J^{\text{MPC}_k}, & \text{MPCS}_k &= \lim_{T \rightarrow \infty} \frac{1}{T} \text{MPCS}_k^T, \\ \text{MPCA}_k^T &= \sup_{w_0, \dots, w_{T-1}} J^{\text{MPC}_k}, & \text{MPCA}_k &= \lim_{T \rightarrow \infty} \frac{1}{T} \text{MPCA}_k^T, \end{aligned}$$

where  $J^{\text{MPC}_k}$  is the cost of MPC given a specific disturbance sequence, i.e.,  $J^{\text{MPC}_k}(w) = J(u, w)$  where for each  $t$ ,  $u_t = \phi(x_t, w_t, \dots, w_{t+k-1})$  and  $\phi(\cdot)$  is the function that maps  $x_t, w_t, \dots, w_{t+k-1}$  to the policy  $u_t$ , as defined in Algorithm 11.1. By definition,  $\text{MPCS}_k \geq \text{STO}_k$  and  $\text{MPCA}_k \geq \text{ADV}_k$  for every  $k \geq 1$  since they use the same information but the latter ones are defined to be optimal.

## Dynamic Regret

Throughout Sections 11.2 to 11.4, we use *dynamic regret* as the performance metric. In Section 11.5, we will also discuss another metric, competitive ratio.

Regret is a standard metric in online learning and provides a bound on the cost difference between an online algorithm and the optimal static policy in a specific policy class given complete information. The predominant policy class is the linear policy class ( $u = Kx$ ). We focus on the *dynamic* regret, which compares to the globally optimal offline policy without any constraint, rather than the optimal static offline policy in a specific class. Note that the globally optimal offline policy may be nonlinear or non-stationary. See more discussions about regret versus dynamic regret in Chapters 8 and 10.

More specifically, we compare the cost of an online algorithm with  $k$  predictions to that of the offline optimal (nonlinear) algorithm, i.e., one that has predictions of all disturbances. For MPC with  $k$  predictions, we define its dynamic regret in the stochastic and adversarial settings, respectively, as:

$$\begin{aligned} \text{Reg}^S(\text{MPC}_k) &= \mathbb{E}_{(w_0, \dots, w_{T-1}) \sim \mathcal{W}} \left( J^{\text{MPC}_k}(w) - \min_{u'_0, \dots, u'_{T-1}} J(u', w) \right), \\ \text{Reg}^A(\text{MPC}_k) &= \sup_{w_0, \dots, w_{T-1} \in \Omega} \left( J^{\text{MPC}_k}(w) - \min_{u'_0, \dots, u'_{T-1}} J(u', w) \right). \end{aligned}$$

Again, as compared to (static) regret, dynamic regret does not have any restriction on the policies  $u'_0, \dots, u'_{T-1}$  used for comparison and thus differs from other notions of regret where  $u'_0, \dots, u'_{T-1}$  are limited in special cases. For example, in the classic form of regret,  $u'_0 = \dots = u'_{T-1}$ ; and in the regret compared to the best offline linear controller (Agarwal, Bullins, et al., 2019; Cohen et al., 2019),  $u'_t = -K^*x_t$ .

In Section 11.3, we obtain both upper bounds and lower bounds on dynamic regret. For lower bounds, we define the minimum possible regret that an algorithm with  $k$  predictions can achieve (i.e., the regret of the algorithm that minimizes the regret):

$$\begin{aligned} \text{Reg}_k^{S*} &= \mathbb{E}_{w_0, \dots, w_{k-1}} \min_{u_0} \mathbb{E}_{w_k} \dots \min_{u_{T-k-1}} \mathbb{E}_{w_{T-1}} \min_{u_{T-k}, \dots, u_{T-1}} \left( J(u, w) - \min_{u'_0, \dots, u'_{T-1}} J(u', w) \right), \\ \text{Reg}_k^{A*} &= \sup_{w_0, \dots, w_{k-1}} \min_{u_0} \sup_{w_k} \dots \min_{u_{T-k-1}} \sup_{w_{T-1}} \min_{u_{T-k}, \dots, u_{T-1}} \left( J(u, w) - \min_{u'_0, \dots, u'_{T-1}} J(u', w) \right). \end{aligned}$$

Finally, we end our discussion of dynamic regret with a note highlighting an impor-

tant contrast between stochastic and adversarial settings. In the stochastic setting,

$$\begin{aligned}
Reg_k^{S*} &= \mathbb{E}_{w_0, \dots, w_{k-1}} \min_{u_0} \mathbb{E}_{w_k} \dots \min_{u_{T-k-1}} \mathbb{E}_{w_{T-1}} \left( \min_{u_{T-k}, \dots, u_{T-1}} J(u, w) - \min_{u'_0, \dots, u'_{T-1}} J(u', w) \right) \\
&= \mathbb{E}_{w_0, \dots, w_{k-1}} \min_{u_0} \mathbb{E}_{w_k} \dots \min_{u_{T-k-1}} \mathbb{E}_{w_{T-1}} \min_{u_{T-k}, \dots, u_{T-1}} J(u, w) - \mathbb{E}_{w_0, \dots, w_{T-1}} \min_{u'_0, \dots, u'_{T-1}} J(u', w) \\
&= STO_k^T - STO_T^T.
\end{aligned}$$

This equality still holds if we take  $\arg \min$  instead of  $\min$  and thus the *regret-optimal policy* is the same as the *cost-optimal policy*. However, in the adversarial case, a similar reasoning gives an inequality:  $Reg_k^{A*} \geq ADV_k^T - ADV_T^T$ , and correspondingly, the regret-optimal and cost-optimal policies can be different. Similarly, for MPC, we have  $Reg^S(\text{MPC}_k) = \text{MPCS}_k^T - STO_T^T$  while  $Reg^A(\text{MPC}_k) \geq \text{MPCA}_k^T - ADV_T^T$ .

### 11.3 Dynamic Regret of MPC in LQ Systems

#### Zero-Mean I.I.D. Disturbances

We begin our analysis with the simplest of the three settings we consider: the disturbances  $w_t$  are independent and identically distributed with zero mean. Though i.i.d. zero-mean is a limited setting, it is still complex enough to study predictions and the first results characterizing the optimal policy with predictions appeared only recently (Goel and Hassibi, 2020; D. J. Foster and Simchowitz, 2020), focusing only on the optimal policy when  $k \rightarrow \infty$ .

Before delving into our results, we first recap the classic *Infinite Horizon Linear Quadratic Stochastic Regulator* (see B. D. Anderson and Moore (2012) and Chapter 8), i.e., the case when  $k = 0$ :

**Lemma 11.1** (B. D. Anderson and Moore (2012)). *Let  $w_t$  be i.i.d. with zero mean and covariance matrix  $W$ . Then, the optimal control policy corresponding to  $STO_0$  is given by:*

$$u_t = -(R + B^\top PB)^{-1} B^\top PAx_t =: -Kx_t,$$

where  $P$  is the solution of discrete-time algebraic Riccati equation (DARE)

$$P = Q + A^\top PA - A^\top PB(R + B^\top PB)^{-1} B^\top PA. \quad (11.1)$$

The corresponding closed-loop dynamics  $A - BK$  is exponentially stable, i.e.,  $\rho(A - BK) < 1$ . Further, the optimal cost is given by  $STO_0 = \text{Tr}\{PW\}$ .

This result has been extensively studied in optimal control theory (Kirk, 2004; B. D. O. Anderson and Moore, 2007) as well as in reinforcement learning (Fazel

et al., 2018; Dean et al., 2020; Simchowitz and D. Foster, 2020). We want to emphasize two important properties of the optimal policy  $u_t = -Kx_t$ . First, the policy is *linear* in the state  $x_t$ . In contrast, we show later that the optimal policy when  $k \neq 0$  is, in general, *nonlinear*. Second, under the assumptions of our model, this policy is *exponentially stable*, i.e.,  $\rho(A - BK) < 1$ . We leverage this to show the power of predictions later.

**Optimal policy with  $k$  predictions.** For notational simplicity, we define  $F = A - BK$  and  $\lambda = \frac{1 + \rho(F)}{2} < 1$ . From Gelfand's formula, there exists a constant  $c(n)$  such that  $\|F^k\| \leq c(n)\lambda^k$  for all  $k \geq 1$ . We explicitly characterize the optimal policy with  $k$  predictions in the following theorem.

**Theorem 11.1.** *Let  $w_t$  be i.i.d. with zero mean and covariance matrix  $W$ . Suppose the controller has  $k \geq 1$  predictions. Then, the optimal control policy at each step  $t$  is given by:*

$$u_t = -(R + B^\top PB)^{-1} B^\top \left( PAx_t + \sum_{i=0}^{k-1} \underbrace{(A^\top - A^\top PH)^i P}_{=F^\top} w_{t+i} \right), \quad (11.2)$$

where  $P$  is the solution of DARE in Eq. (11.1). The cost under this policy is:

$$\text{STO}_k = \text{Tr} \left\{ \left( P - \sum_{i=0}^{k-1} P(A - HPA)^i H(A^\top - A^\top PH)^i P \right) W \right\}, \quad (11.3)$$

where  $H = B(R + B^\top PB)^{-1} B^\top$ .

*Proof.* Our proof technique closely follows that in Section 4.1 of Goel and Hassibi (2020). To begin, note that the definition of  $\text{STO}_k^T$  has a structure of repeating min's and  $\mathbb{E}$ 's. Similarly to the analysis of Eq. (8.5) in Chapter 8, we use dynamic programming to compute the value iteratively. In particular, we apply backward induction to solve the optimal cost-to-go functions, from time step  $T$  to the initial state. Given state  $x_t$  and predictions  $w_t, \dots, w_{t+k-1}$ , we define the cost-to-go function:

$$\begin{aligned} V_t(x_t; w_{t:t+k-1}) &:= \min_{u_t} \mathbb{E} \min_{w_{t+k}} \cdots \mathbb{E} \min_{w_{T-1}} \min_{u_{T-k}, \dots, u_{T-1}} \sum_{i=t}^{T-1} (x_i^\top Q x_i + u_i^\top R u_i) + x_T^\top Q_f x_T \\ &= x_t^\top Q x_t + \min_{u_t} \left( u_t^\top R u_t + \mathbb{E}_{w_{t+k}} [V_{t+1}(Ax_t + Bu_t + w_t; w_{t+1:t+k})] \right) \end{aligned} \quad (11.4)$$

with  $V_T(x_T; \dots) = x_T^\top Q_f x_T$ . Note that  $\mathbb{E}_{w_{t+k}}$  has no effect for  $t \geq T - k$ . This function measures the expected overall control cost from a given state to the end, assuming the controller makes the optimal decision at each time.

We will show by backward induction that for every  $t = 0, \dots, T$ ,  $V_t(x_t; w_{t:t+k-1}) = x_t^\top P_t x_t + v_t^\top x_t + q_t$ , where  $P_t, v_t, q_t$  are coefficients that may depend on  $w_{t:t+k-1}$ . This is clearly true for  $t = T$ . Suppose this is true at  $t + 1$ . Then,

$$\begin{aligned} V_t(x; w_{t:t+k-1}) &= x^\top Qx + \min_u \left( u^\top Ru + (Ax + Bu + w_t)^\top P_{t+1} (Ax + Bu + w_t) \right. \\ &\quad \left. + \mathbb{E}_{w_{t+k}} [v_{t+1}]^\top (Ax + Bu + w_t) + \mathbb{E}_{w_{t+k}} [q_{t+1}] \right) \\ &= x^\top Qx + (Ax + w_t)^\top P_{t+1} (Ax + w_t) + \mathbb{E}_{w_{t+k}} [v_{t+1}]^\top (Ax + w_t) + \mathbb{E}_{w_{t+k}} [q_{t+1}] \\ &\quad + \min_u \left( u^\top (R + B^\top P_{t+1} B) u + u^\top B^\top \left( 2P_{t+1} Ax + 2P_{t+1} w_t + \mathbb{E}_{w_{t+k}} [v_{t+1}] \right) \right). \end{aligned}$$

The optimal  $u$  is obtained by setting the derivative to be zero:

$$u^* = -(R + B^\top P_{t+1} B)^{-1} B^\top \left( P_{t+1} Ax + P_{t+1} w_t + \frac{1}{2} \mathbb{E}_{w_{t+k}} [v_{t+1}] \right). \quad (11.5)$$

Let  $H_t = B(R + B^\top P_{t+1} B)^{-1} B^\top$ . Plugging  $u^*$  back into  $V_t$ , we have

$$\begin{aligned} V_t(x; w_{t:t+k-1}) &= x^\top Qx + (Ax + w_t)^\top P_{t+1} (Ax + w_t) + \mathbb{E}_{w_{t+k}} [v_{t+1}]^\top (Ax + w_t) + \mathbb{E}_{w_{t+k}} [q_{t+1}] \\ &\quad - \left( P_{t+1} Ax + P_{t+1} w_t + \frac{1}{2} \mathbb{E}_{w_{t+k}} [v_{t+1}] \right)^\top H_t \left( P_{t+1} Ax + P_{t+1} w_t + \frac{1}{2} \mathbb{E}_{w_{t+k}} [v_{t+1}] \right) \\ &= x^\top (Q + A^\top P_{t+1} A - A^\top P_{t+1} H_t P_{t+1} A) x \\ &\quad + x^\top \left( (A^\top - A^\top P_{t+1} H_t) \mathbb{E}_{w_{t+k}} [v_{t+1}] + 2(A^\top - A^\top P_{t+1} H_t) P_{t+1} w_t \right) \\ &\quad + w_t^\top (P_{t+1} - P_{t+1} H_t P_{t+1}) w_t + w_t^\top (I - P_{t+1} H_t) \mathbb{E}_{w_{t+k}} [v_{t+1}] \\ &\quad - \frac{1}{4} \mathbb{E}_{w_{t+k}} [v_{t+1}]^\top H_t \mathbb{E}_{w_{t+k}} [v_{t+1}] + \mathbb{E}_{w_{t+k}} [q_{t+1}]. \end{aligned}$$

Thus, the recursive formulae are given by:

$$P_t = Q + A^\top P_{t+1} A - A^\top P_{t+1} H_t P_{t+1} A, \quad (11.6a)$$

$$v_t = (A^\top - A^\top P_{t+1} H_t) \mathbb{E}_{w_{t+k}} [v_{t+1}] + 2(A^\top - A^\top P_{t+1} H_t) P_{t+1} w_t, \quad (11.6b)$$

$$\begin{aligned} q_t &= w_t^\top (P_{t+1} - P_{t+1} H_t P_{t+1}) w_t + w_t^\top (I - P_{t+1} H_t) \mathbb{E}_{w_{t+k}} [v_{t+1}] \\ &\quad - \frac{1}{4} \mathbb{E}_{w_{t+k}} [v_{t+1}]^\top H_t \mathbb{E}_{w_{t+k}} [v_{t+1}] + \mathbb{E}_{w_{t+k}} [q_{t+1}]. \end{aligned} \quad (11.6c)$$

As  $T - t \rightarrow \infty$ ,  $P_t$  and  $H_t$  converge to  $P$  and  $H$ , respectively, where  $P$  is the solution of discrete-time algebraic Riccati equation (DARE)  $P = Q + A^\top P A - A^\top P H P A$ , and  $H = B(R + B^\top P B)^{-1} B^\top$ . Note that  $v_T = 0$  and  $q_T = 0$ . Then,

$$v_t = 2 \sum_{i=0}^{k-1} (A^\top - A^\top P H)^{i+1} P w_{t+i}, \quad (11.7)$$

$$q_t = w_t^\top (P - P H P) w_t + w_t^\top (I - P H) \mathbb{E}_{w_{t+k}} [v_{t+1}] - \frac{1}{4} \mathbb{E}_{w_{t+k}} [v_{t+1}]^\top H \mathbb{E}_{w_{t+k}} [v_{t+1}] + \mathbb{E}_{w_{t+k}} [q_{t+1}], \quad (11.8)$$

$$\mathbb{E}_{w_{t+k}} [v_{t+1}] = 2 \sum_{i=1}^{k-1} (A^\top - A^\top P H)^i P w_{t+i}. \quad (11.9)$$

Taking the expectation of  $q_t$  over all randomness, namely  $w_0, w_1, w_2, \dots$ , we have

$$\begin{aligned} \mathbb{E}[q_t] &= \text{Tr}\{(P - P H P)W\} - \sum_{i=1}^{k-1} \text{Tr}\{P(A - H P A)^i H (A^\top - A^\top P H)^i P W\} + \mathbb{E}[q_{t+1}] \\ &= \text{Tr}\left\{\left(P - \sum_{i=0}^{k-1} P(A - H P A)^i H (A^\top - A^\top P H)^i P\right)W\right\} + \mathbb{E}[q_{t+1}], \end{aligned} \quad (11.10)$$

where in the first equality we use  $\mathbb{E}[w_t] = 0$  and the independence of the disturbances. Thus, as  $T \rightarrow \infty$ , in each time step, a constant cost is incurred and the average cost  $\text{STO}_k$  is exactly this value.

$$\begin{aligned} \text{STO}_k &= \lim_{T \rightarrow \infty} \frac{1}{T} \text{STO}_k^T = \lim_{T \rightarrow \infty} \frac{1}{T} \mathbb{E}[V_0(x_0; w_{0:k-1})] = \lim_{T \rightarrow \infty} \frac{1}{T} \mathbb{E}[q_0] \\ &= \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}[q_t] - \mathbb{E}[q_{t+1}] = \text{Tr}\left\{\left(P - \sum_{i=0}^{k-1} P(A - H P A)^i H (A^\top - A^\top P H)^i P\right)W\right\}. \end{aligned}$$

The explicit form of the optimal control policy is obtained by combining Eqs. (11.5) and (11.9).  $\square$

Roughly speaking, the proof is based on an analysis of quadratic cost-to-go functions in the form  $V_t(x_t) = x_t^\top P_t x_t + v_t^\top x_t + q_t$  (Chapter 8's analysis does not have the linear term  $v_t^\top x_t$ ). Note that  $A - H P A = A - B(R + B^\top P B)^{-1} B^\top P A = A - B K = F$ . Thus, the online optimal cost  $\text{STO}_k$  with  $k$  predictions approaches the offline optimal cost  $\text{STO}_\infty$  by an exponential rate. In other words,  $\text{STO}_k / \text{STO}_\infty = 1 + O(\|F^k\|^2) = 1 + O(\lambda^{2k})$ . Two extreme cases of our result are noteworthy. When  $k = 0$ , it reduces to the classic Lemma 11.1. When  $k \rightarrow \infty$ , it reduces to the offline optimal case derived by Goel and Hassibi (2020).



**MPC's performance.** As might be expected, since the disturbances are i.i.d., future disturbances have no dependence on the current. As a result, MPC in Algorithm 11.1 with  $Q_f = P$  gives the *optimal* policy, as shown below.

**Theorem 11.2.** *In Algorithm 11.1, let  $\tilde{Q}_f = P$ . Then, the MPC policy with  $k$  predictions is always given by Eq. (11.2) (no matter  $w_t$  is i.i.d. stochastic, generally stochastic, or adversarial). Moreover, assuming i.i.d. disturbance with zero mean, the MPC policy is optimal (both cost-optimal and dynamic-regret-optimal).*

*Proof.* Due to the greedy nature, MPC policy is given by the solution of a length- $k$  optimal control problem, given deterministic and known  $w_t, \dots, w_{t+k-1}$ . In other words, we want to derive the optimal policy  $(u_t, \dots, u_{t+k-1})$  that minimizes

$$\sum_{i=t}^{t+k-1} (x_i^\top Q x_i + u_i^\top R u_i) + x_{t+k}^\top P x_{t+k},$$

where  $x_{i+1} = Ax_i + Bu_i + w_i$ , given  $x_t, w_t, \dots, w_{t+k-1}$ . Define the cost-to-go function at time  $i$  given  $x_i, w_i, \dots, w_{t+k-1}$ :

$$\begin{aligned} V_i(x_i; w_{i:t+k-1}) &= \min_{u_{i:t+k-1}} \sum_{j=i}^{t+k-1} (x_j^\top Q x_j + u_j^\top R u_j) + x_{t+k}^\top P x_{t+k} \\ &= x_i^\top Q x_i + \min_{u_i} (u_i^\top R u_i + V_{i+1}(Ax_i + Bu_i + w_i; w_{i+1:t+k-1})). \end{aligned}$$

Note that  $V_{t+k}(x_{t+k}) = x_{t+k}^\top P x_{t+k}$ . Similar to the proof of Theorem 11.1, we can inductively show that  $V_i(x_i; w_{i:t+k-1}) = x_i^\top P x_i + v_i^\top x_i + q_i$  for some  $v_i$  and  $q_i$ . Note that the second-degree coefficient no longer depends on the index  $i$  as in the previous proof because we start from  $P$ , the solution of DARE. We then have the followings equations that parallel with Equations (11.5) and (11.7):

$$\begin{aligned} v_i &= 2 \sum_{j=0}^{t+k-i-1} F^{\top j+1} P w_{i+j}, \\ u_i^* &= -(R + B^\top P B)^{-1} B^\top \left( P A x_i + P w_i + \frac{1}{2} v_{i+1} \right) \\ &= -(R + B^\top P B)^{-1} B^\top \left( P A x_i + \sum_{j=0}^{t+k-i-1} F^{\top j} P w_{i+j} \right). \end{aligned}$$

The case  $i = t$  gives:

$$u_t^* = -(R + B^\top P B)^{-1} B^\top \left( P A x_t + \sum_{j=0}^{k-1} F^{\top j} P w_{t+j} \right),$$

which is the MPC policy at time step  $t$ , and is same as Equation (11.2).  $\square$

### General Stochastic Disturbance

In this section, we consider a general form of stochastic disturbance, more general than typically considered in this context (Dean et al., 2020; Cohen et al., 2019; Cassel et al., 2020). Suppose the disturbance sequence  $\{w_t\}_{t=0,1,2,\dots}$  is sampled from a joint distribution  $\mathcal{W}$  such that the trace of the cross-correlation of each pair is uniformly bounded, i.e., there exist  $m > 0$  such that for all  $t, t' \geq 1$ ,  $\mathbb{E}[w_t^\top w_{t'}] \leq m$ .

**Optimal policy with  $k$  predictions.** In the case of general stochastic disturbances, we cannot obtain as clean a form for  $\text{STO}_k$  as in the i.i.d. case in Theorem 11.1. However, the marginal benefit of having an extra prediction decays with the same (exponential) rate and the optimal policy is similar to that in Theorem 11.1, but with some additional terms that characterize the expected future disturbances given the current information, as shown in the following theorem.

**Theorem 11.3.** *The optimal control policy with general stochastic disturbance is given by:*

$$u_t = -(R + B^\top P B)^{-1} B^\top \left( P A x_t + \sum_{i=0}^{k-1} F^{\top i} P w_{t+i} + \sum_{i=k}^{\infty} F^{\top i} P \mu_{t+i|t+k-1} \right), \quad (11.11)$$

where  $\mu_{t'|t} = \mathbb{E}[w_{t'} | w_0, \dots, w_t]$ . Under this policy, the marginal benefit of obtaining an extra prediction decays exponentially fast in the existing number  $k$  of predictions. Formally, for  $k \geq 1$ ,

$$\text{STO}_k - \text{STO}_{k+1} = O(\|F^k\|^2) = O(\lambda^{2k}).^2$$

*Proof.* Similar to the proof of Theorem 11.1, we assume

$$V_t(x_t; w_{0:t+k-1}) = x_t^\top P_t x_t + x_t^\top v_t + q_t,$$

where  $V_t$  has a similar definition as in Eq. (11.4) but may further depend on  $w_0, \dots, w_{t-1}$  because the disturbance sequence is no longer Markovian. In this case,  $P_t$ ,  $v_t$  and  $q_t$  still satisfy the recursive forms in Eq. (11.6). However, the expected values of  $w_t$  and  $v_t$  are different since we have a more general distribution now. Let  $T - t \rightarrow \infty$ ,  $\mu_{t'|t} = \mathbb{E}[w_{t'} | w_0, \dots, w_t]$  and  $F = A - HPA$ . Then,

$$v_t^k = 2 \sum_{i=0}^{k-1} F^{\top i+1} P w_{t+i} + 2 \sum_{i=k}^{\infty} F^{\top i+1} P \mu_{t+i|t+k-1}, \quad (11.12)$$

$$q_t^k = \overbrace{w_t^\top (P - PHP) w_t + w_t^\top (I - PH)}_{w_{t+k}} \mathbb{E}_{w_{t+k}} [v_{t+1}^k] - \frac{1}{4} \mathbb{E}_{w_{t+k}} [v_{t+1}^k]^\top H \mathbb{E}_{w_{t+k}} [v_{t+1}^k] + \mathbb{E}_{w_{t+k}} [q_{t+1}^k],$$

<sup>2</sup>We say that  $f(k) = O(g(k))$  if  $\exists C > 0, \forall k \geq 1, |f(k)| \leq C g(k)$ ;  $\Omega()$  is similar except that the last “ $\leq$ ” is replaced by “ $\geq$ ”;  $\Theta()$  means both  $O()$  and  $\Omega()$ . This is stronger than the standard definition where  $f(k) = O(g(k))$  if  $\exists C > 0, k^* > 0, \forall k \geq k^*, |f(k)| \leq C g(k)$ .

where the superscript  $k$  denotes the number of predictions.

The optimal policy in this case has the same form as Eq. (11.5). Plugging Eq. (11.12) into it, we obtain the optimal policy in the theorem.

Further,

$$\mathbb{E}[q_t^k - q_t^{k+1}] = \mathbb{E}\left[w_t^\top (I - PH) \left( \mathbb{E}_{w_{t+k}} [v_{t+1}^k] - \mathbb{E}_{w_{t+k+1}} [v_{t+1}^{k+1}] \right)\right] \quad (11.13a)$$

$$+ \frac{1}{4} \mathbb{E}\left[ \mathbb{E}_{w_{t+k+1}} [v_{t+1}^{k+1}]^\top H \mathbb{E}_{w_{t+k+1}} [v_{t+1}^{k+1}] - \mathbb{E}_{w_{t+k}} [v_{t+1}^k]^\top H \mathbb{E}_{w_{t+k}} [v_{t+1}^k] \right] \quad (11.13b)$$

$$+ \mathbb{E}[q_{t+1}^k - q_{t+1}^{k+1}], \quad (11.13c)$$

where the expectation  $\mathbb{E}$  is taken over all randomness. Part (11.13a) is zero because

$$\mathbb{E}_{w_{t+k}} [v_{t+1}^k] = \mathbb{E}_{w_{t+k}, w_{t+k+1}} [v_{t+1}^{k+1}].$$

$$\begin{aligned} \text{Part (11.13b)} &= \frac{1}{4} \mathbb{E}_{w_{t+k}} \left[ \left( \mathbb{E}_{w_{t+k+1}} [v_{t+1}^{k+1}] - \mathbb{E}_{w_{t+k}} [v_{t+1}^k] \right)^\top H \left( \mathbb{E}_{w_{t+k+1}} [v_{t+1}^{k+1}] - \mathbb{E}_{w_{t+k}} [v_{t+1}^k] \right) \right] \\ &= \mathbb{E}_{w_{t+k}} [z_{k,t}^\top H z_{k,t}], \end{aligned}$$

where

$$z_{k,t} = F^{\top k} P (w_{t+k} - \mu_{t+k|t+k-1}) + \sum_{i=k+1}^{\infty} F^{\top i} P (\mu_{t+i|t+k} - \mu_{t+i|t+k-1}).$$

Note that  $z_{k,t} = F^\top z_{k-1,t+1} = F^{\top k} z_{0,t+k}$ . Thus,

$$\begin{aligned} \text{STO}_k - \text{STO}_{k+1} &= \lim_{T \rightarrow \infty} \frac{1}{T} \mathbb{E}[q_0^k - q_0^{k+1}] \\ &= \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}[z_{k,t}^\top H z_{k,t}] \\ &= \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}[z_{0,t+k}^\top F^k H F^{\top k} z_{0,t+k}] \\ &= \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=0}^{T-1} \text{Tr} \left\{ F^k H F^{\top k} \mathbb{E}[z_{0,t+k} z_{0,t+k}^\top] \right\} \\ &\leq \|F^k\|^2 \|H\| \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=0}^{T-1} \text{Tr} \{ \mathbb{E}[z_{0,t+k} z_{0,t+k}^\top] \} \end{aligned}$$

where in the last line we use the fact that if  $A$  is symmetric, then  $\text{Tr}\{AB\} \leq \lambda_{\max}(A)\text{Tr}\{B\}$ . Finally we just need to show the last item  $\text{Tr}\left\{\mathbb{E}\left[z_{0,t+k}z_{0,t+k}^\top\right]\right\}$  is uniformly bounded for all  $t$ . This is straightforward because the cross-correlation of each disturbance pair is uniformly bounded, i.e., there exists  $m > 0$  such that for all  $t, t' \geq 1$ ,  $\mathbb{E}[w_t^\top w_{t'}] \leq m$ .

$$\begin{aligned} \text{Tr}\left\{\mathbb{E}\left[z_{0,t}z_{0,t}^\top\right]\right\} &= \sum_{i,j=0}^{\infty} \text{Tr}\left\{\mathbb{E}\left[PF^iF^{\top j}P(\mu_{t+j|t} - \mu_{t+j|t-1})(\mu_{t+i|t} - \mu_{t+i|t-1})^\top\right]\right\} \\ &= \sum_{i,j=0}^{\infty} \text{Tr}\left\{PF^iF^{\top j}P\mathbb{E}\left[\mu_{t+j|t}\mu_{t+i|t}^\top - \mu_{t+j|t-1}\mu_{t+i|t-1}^\top\right]\right\} \\ &\leq \sum_{i,j=0}^{\infty} \|F^i\| \|F^j\| \|P\|^2 \mathbb{E}\left[w_{t+j}^\top w_{t+i} - w_{t+j}^\top w_{t+i}\right] \\ &\leq \sum_{i,j=0}^{\infty} c\lambda^i c\lambda^j \|P\|^2 2m = 2\frac{c^2}{(1-\lambda)^2} \|P\|^2 m \end{aligned}$$

for some constant  $c$  from Gelfand's formula. Thus  $\text{Tr}\left\{\mathbb{E}\left[z_{0,t}z_{0,t}^\top\right]\right\}$  is bounded by a constant independent of  $t$ . Thus,

$$\text{STO}_k - \text{STO}_{k+1} = O(\|F^k\|^2).$$

□

Roughly speaking, this proof leverages a novel difference analysis of cost-to-go functions. Note that for some distributions,  $\text{STO}_k$  may approach  $\text{STO}_\infty$  much faster than exponential rate. It is even possible that  $\text{STO}_k = \text{STO}_\infty$  for finite  $k$ , as we show in Example 11.2 below. On the other hand, there are scenarios where  $\text{STO}_k$  approaches  $\text{STO}_\infty$  in an exactly exponential manner, as we show in Example 11.3 below.

**Example 11.2.** Define the joint distribution  $\mathcal{W}$  such that with probability  $\frac{1}{2}$ , all  $w_t = w$ , and otherwise all  $w_t = -w$ . In this case, one prediction is equivalent to infinite predictions since it is enough to distinguish these two scenarios with only  $w_0$ . As a result,  $\text{STO}_1 = \text{STO}_\infty$ .

**Example 11.3.** Suppose the system is 1-d ( $n = d = 1$ ) and the disturbance is i.i.d. with zero mean, i.e., the setting of Theorem 11.1. Then, according to Eq. (11.3), as

long as  $F, P, H, W$  are non-zero,

$$\text{STO}_k - \text{STO}_\infty = \sum_{i=k}^{\infty} F^{2i} P^2 H W = \Theta(F^{2k}).$$

**MPC's performance.** The comparison between the MPC policy in Eq. (11.2) and the optimal policy in Eq. (11.11) reveals that MPC is a *truncation* of the optimal policy and is no longer optimal because MPC is a greedy policy without considering future dependence on current information. Nevertheless, it is still a near-optimal policy, as characterized by the following results.

**Theorem 11.4.**  $\text{MPCS}_k - \text{MPCS}_{k+1} = O(\|F^k\|^2) = O(\lambda^{2k})$ . Moreover, in Example 11.3,  $\text{MPCS}_k - \text{MPCS}_{k+1} = \Theta(\|F^k\|^2)$ .

*Proof.* To recursively calculate the value of  $J^{\text{MPC}_k}$ , we define:

$$\begin{aligned} V_t^{\text{MPC}_k}(x_t; w_{0:t+k-1}) &= \sum_{i=t}^{T-1} (x_i^\top Q x_i + u_i^\top R u_i) + x_T^\top Q_f x_T \\ &= x_t^\top Q x_t + u_t^\top R u_t + V_{t+1}(A x_t + B u_t + w_t; w_{0:t+k}) \end{aligned}$$

as the cost-to-go function with MPC as the policy, i.e.,  $u_t$  is the control at time step  $t$  from the MPC policy with  $k$  predictions. Similar to the previous proofs, we assume  $V_t^{\text{MPC}_k}(x) = x^\top P_t x + x^\top v_t + q_t$  (which turns out to be correct by induction) and  $T - t \rightarrow \infty$  so that  $P_t = P$ . Then,

$$\begin{aligned} V_t^{\text{MPC}_k}(x_t; w_{0:t+k-1}) &= x_t^\top Q x_t + u_t^\top R u_t + (A x_t + B u_t + w_t)^\top P (A x_t + B u_t + w_t) \\ &\quad + (A x_t + B u_t + w_t)^\top v_{t+1} + q_{t+1} \\ &= u_t^\top (R + B^\top P B) u_t + 2 u_t^\top B^\top (P A x_t + P w_t + v_{t+1}/2) \\ &\quad + x_t^\top Q x_t + (A x_t + w_t)^\top P (A x_t + w_t) + (A x_t + w_t)^\top v_{t+1} + q_{t+1}. \end{aligned} \tag{11.14}$$

Let  $F = A - HPA$ . Plugging in the formula of  $u_t$  in Theorem 11.2, we have

$$\begin{aligned}
V_t^{\text{MPC}^k}(x_t; w_{0:t+k-1}) &= \left( \frac{1}{2}v_{t+1} - \sum_{i=1}^{k-1} F^{\top i} P w_{t+i} \right)^{\top} H \left( \frac{1}{2}v_{t+1} - \sum_{i=1}^{k-1} F^{\top i} P w_{t+i} \right) \\
&\quad - \left( P A x_t + P w_t + \frac{1}{2}v_{t+1} \right)^{\top} H \left( P A x_t + P w_t + \frac{1}{2}v_{t+1} \right) \\
&\quad + x_t^{\top} Q x_t + (A x_t + w_t)^{\top} P (A x_t + w_t) + (A x_t + w_t)^{\top} v_{t+1} + q_{t+1} \\
&= x_t^{\top} (Q + A^{\top} P A - A^{\top} P H P A) x_t + x_t^{\top} (F^{\top} v_{t+1} + 2F^{\top} P w_t) \\
&\quad + \left( \frac{1}{2}v_{t+1} - \sum_{i=1}^{k-1} F^{\top i} P w_{t+i} \right)^{\top} H \left( \frac{1}{2}v_{t+1} - \sum_{i=1}^{k-1} F^{\top i} P w_{t+i} \right) \\
&\quad - \left( P w_t + \frac{1}{2}v_{t+1} \right)^{\top} H \left( P w_t + \frac{1}{2}v_{t+1} \right) + w_t^{\top} P w_t + w_t^{\top} v_{t+1} + q_{t+1} \\
&= x_t^{\top} P x_t + x_t^{\top} v_t + q_t.
\end{aligned}$$

Thus,

$$v_t = F^{\top} v_{t+1} + 2F^{\top} P w_t = 2 \sum_{i=0}^{\infty} F^{\top i+1} P w_{t+i}.$$

Then, we can plug  $v_{t+1}$  into  $q_t$ :

$$\begin{aligned}
q_t &= q_{t+1} + \left( \sum_{i=k}^{\infty} F^{\top i} P w_{t+i} \right)^{\top} H \left( \sum_{i=k}^{\infty} F^{\top i} P w_{t+i} \right) \\
&\quad - \left( \sum_{i=0}^{\infty} F^{\top i} P w_{t+i} \right)^{\top} H \left( \sum_{i=0}^{\infty} F^{\top i} P w_{t+i} \right) + w_t^{\top} P w_t + 2w_t^{\top} \left( \sum_{i=1}^{\infty} F^{\top i} P w_{t+i} \right).
\end{aligned} \tag{11.15}$$

Note that Eq. (11.15) is for MPC with  $k$  predictions. With the disturbance sequence  $\{w_t\}$  fixed, we can compare the per-step cost of MPC with  $k$  predictions and that with  $k+1$  predictions:

$$\begin{aligned}
q_t^k - q_t^{k+1} &= q_{t+1}^k - q_{t+1}^{k+1} + \left( \sum_{i=k}^{\infty} F^{\top i} P w_{t+i} \right)^{\top} H \left( \sum_{i=k}^{\infty} F^{\top i} P w_{t+i} \right) \\
&\quad - \left( \sum_{i=k+1}^{\infty} F^{\top i} P w_{t+i} \right)^{\top} H \left( \sum_{i=k+1}^{\infty} F^{\top i} P w_{t+i} \right) \\
&= q_{t+1}^k - q_{t+1}^{k+1} + w_{t+k}^{\top} P F^k H F^{\top k} \left( P w_{t+k} + 2 \sum_{i=1}^{\infty} F^{\top i} P w_{t+i+k} \right).
\end{aligned} \tag{11.16}$$

Thus,

$$\begin{aligned}
\mathbb{E}[q_t^k - q_t^{k+1} - (q_{t+1}^k - q_{t+1}^{k+1})] &= \mathbb{E}\left[w_{t+k}^\top P F^k H F^{\top k} \left( P w_{t+k} + 2 \sum_{i=1}^{\infty} F^{\top i} P w_{t+i+k} \right)\right] \\
&= \text{Tr}\left\{ P F^k H F^{\top k} \left( P \mathbb{E}[w_{t+k} w_{t+k}^\top] + 2 \sum_{i=1}^{\infty} F^{\top i} P \mathbb{E}[w_{t+i+k} w_{t+i+k}^\top] \right)\right\} \\
&= \text{Tr}\{P F^k H F^{\top k} Z_{k,t}\},
\end{aligned}$$

where  $Z_{k,t} = P \mathbb{E}[w_{t+k} w_{t+k}^\top] + 2 \sum_{i=1}^{\infty} F^{\top i} P \mathbb{E}[w_{t+i+k} w_{t+i+k}^\top]$ . Note that  $Z_{k,t} = Z_{k-1,t+1}$ .

$$\begin{aligned}
\text{MPCS}_k - \text{MPCS}_{k+1} &= \lim_{T \rightarrow \infty} \frac{1}{T} \mathbb{E}[q_0^k - q_0^{k+1}] \\
&= \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=0}^{T-1} \text{Tr}\{P F^k H F^{\top k} Z_{k,t}\} \\
&\leq \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=0}^{T-1} \|P\| \|H\| \|F^k\|^2 \text{Tr}\{Z_{k,t}\},
\end{aligned}$$

where in the last line we use the fact that if  $A$  is symmetric, then  $\text{Tr}\{AB\} \leq \|A\| \text{Tr}\{B\}$ . Similarly to the last part in the proof of Theorem 11.3, now we just need to show the last term  $\text{Tr}\{Z_{k,t}\}$  is uniformly bounded for all  $t$ . Again, this is because the cross-correlation of each disturbance pair is uniformly bounded.

$$\begin{aligned}
\text{Tr}\{Z_{k,t}\} &\leq \|P\| \text{Tr}\{\mathbb{E}[w_{t+k} w_{t+k}^\top]\} + 2 \sum_{i=1}^{\infty} \|P\| \|F^i\| \mathbb{E}\left[\sum_j \sigma_j(w_{t+i+k} w_{t+k}^\top)\right] \\
&\leq \|P\| m + 2 \sum_{i=1}^{\infty} c \lambda^i \|P\| m = \|P\| m + 2c \frac{\lambda}{1-\lambda} \|P\| m
\end{aligned}$$

where  $c$  is some constant, and in the first line, we use the fact that  $\text{Tr}\{AB\} \leq \|A\| \sum_j \sigma_j(B)$  with  $\sigma_j(\cdot)$  denoting the  $j$ -th singular value. Thus,  $\text{Tr}\{Z_{k,t}\}$  is uniformly bounded. Therefore,  $\text{MPCS}_k - \text{MPCS}_{k+1} = O(\|F^k\|^2)$ .  $\square$

In other words, the marginal benefit for the MPC algorithm of an extra prediction decays exponentially fast, paralleling the result for optimal policy in Eq. (11.11).

Besides, the dynamic regret of MPC (nearly) matches the order of the optimal dynamic regret, as shown in Theorem 11.5 and Theorem 11.6.

**Theorem 11.5** (Main result).  $\text{Reg}^S(\text{MPC}_k) = \text{MPCS}_k^T - \text{STO}_T^T = O(\|F^k\|^2 T + 1) = O(\lambda^{2k} T + 1)$ , where the second term results from the difference between finite/infinite horizons.

*Proof.* To calculate the dynamic regret, we cannot simply let  $T - t \rightarrow \infty$  as we did before Eq. (11.14) in the proof of Theorem 11.4 and instead need to handle the expressions in a more delicate manner. In particular, we need to rigorously analyze the impact of finite horizon. Let  $\Delta_t = P_t - P$ .

$$\begin{aligned}
& V_t^{\text{MPC}^k}(x_t; w_{0:t+k-1}) \\
&= u_t^\top (R + B^\top P_{t+1} B) u_t + 2u_t^\top B^\top (P_{t+1} A x_t + P_{t+1} w_t + v_{t+1}/2) \\
&\quad + x_t^\top Q x_t + (A x_t + w_t)^\top P_{t+1} (A x_t + w_t) + (A x_t + w_t)^\top v_{t+1} + q_{t+1} \\
&= u_t^\top (R + B^\top P B) u_t + 2u_t^\top B^\top (P A x_t + P w_t + v_{t+1}/2) \\
&\quad + x_t^\top Q x_t + (A x_t + w_t)^\top P (A x_t + w_t) + (A x_t + w_t)^\top v_{t+1} + q_{t+1} \\
&\quad + u_t^\top B^\top \Delta_{t+1} B u_t + 2u_t^\top B^\top \Delta_{t+1} (A x_t + w_t) + (A x_t + w_t)^\top \Delta_{t+1} (A x_t + w_t).
\end{aligned}$$

Plugging in the MPC policy as in Theorem 11.2, we have:

$$\begin{aligned}
& V_t^{\text{MPC}^k}(x_t; w_{0:t+k-1}) \\
&= x_t^\top (Q + A^\top P A - A^\top P H P A) x_t + x_t^\top (F^\top v_{t+1} + 2F^\top P w_t) \\
&\quad + \left( \frac{1}{2} v_{t+1} - \sum_{i=1}^{k-1} F^{\top i} P w_{t+i} \right)^\top H \left( \frac{1}{2} v_{t+1} - \sum_{i=1}^{k-1} F^{\top i} P w_{t+i} \right) \\
&\quad - \left( P w_t + \frac{1}{2} v_{t+1} \right)^\top H \left( P w_t + \frac{1}{2} v_{t+1} \right) + w_t^\top P w_t + w_t^\top v_{t+1} + q_{t+1} \\
&\quad + \left( F x_t + w_t - \sum_{i=0}^{k-1} F^{\top i} P w_{t+i} \right)^\top \Delta_{t+1} \left( F x_t + w_t - \sum_{i=0}^{k-1} F^{\top i} P w_{t+i} \right) \\
&= x_t^\top (Q + A^\top P A - A^\top P H P A + F^\top \Delta_{t+1} F) x_t \\
&\quad + x_t^\top \left( F^\top v_{t+1} + 2F^\top P w_t + 2F^\top \Delta_{t+1} \left( w_t - \sum_{i=0}^{k-1} F^{\top i} P w_{t+i} \right) \right) \\
&\quad + \left( \frac{1}{2} v_{t+1} - \sum_{i=1}^{k-1} F^{\top i} P w_{t+i} \right)^\top H \left( \frac{1}{2} v_{t+1} - \sum_{i=1}^{k-1} F^{\top i} P w_{t+i} \right) \\
&\quad - \left( P w_t + \frac{1}{2} v_{t+1} \right)^\top H \left( P w_t + \frac{1}{2} v_{t+1} \right) + w_t^\top P w_t + w_t^\top v_{t+1} + q_{t+1} \\
&\quad + \left( w_t - \sum_{i=0}^{k-1} F^{\top i} P w_{t+i} \right)^\top \Delta_{t+1} \left( w_t - \sum_{i=0}^{k-1} F^{\top i} P w_{t+i} \right).
\end{aligned}$$

Comparing this with the induction hypothesis  $V_t^{\text{MPC}^k} = x_t^\top (P + \Delta_t) x_t + x_t^\top v_t + q_t$ , we obtain the recursive formulae for  $\Delta_t, v_t, q_t$ .

$$\Delta_t = F^\top \Delta_{t+1} F = F^{\top T-t} \Delta_T F^{T-t} = F^{\top T-t} (Q_f - P) F^{T-t}.$$



This implies that  $P_t$  converges to  $P$  exponentially fast, i.e.,  $\|\Delta_t\| = O(\|F^{T-t}\|^2) = O(\lambda^{2(T-t)})$ .

$$\begin{aligned}
v_t &= F^\top v_{t+1} + 2F^\top P w_t + 2F^\top \Delta_{t+1} \left( w_t - \sum_{i=0}^{k-1} F^{\top i} P w_{t+i} \right) \\
&= 2 \sum_{j=0}^{T-t-1} \left( F^{\top j+1} P w_{t+j} + F^{\top j+1} \Delta_{t+j+1} \left( w_{t+j} - \sum_{i=0}^{k-1} F^{\top i} P w_{t+j+i} \right) \right) \\
&= 2 \sum_{i=0}^{T-t-1} F^{\top i+1} P w_{t+i} + 2 \sum_{j=0}^{T-t-1} F^{\top j+1} \Delta_{t+j+1} \left( w_{t+j} - \sum_{i=0}^{k-1} F^{\top i} P w_{t+j+i} \right).
\end{aligned}$$

Denote the second term by  $2d_t$ . We have

$$\begin{aligned}
d_t &= \sum_{j=0}^{T-t-1} F^{\top j+1} \Delta_{t+j+1} \left( w_{t+j} - \sum_{i=0}^{k-1} F^{\top i} P w_{t+j+i} \right) \\
&= \sum_{j=0}^{T-t-1} O(\lambda^j \lambda^{2(T-t-j)}) = O(\lambda^{T-t}).
\end{aligned}$$

$$\begin{aligned}
d_t^k - d_t^{k+1} &= \sum_{j=0}^{T-t-k-1} F^{\top j+1} \Delta_{t+j+1} F^{\top k} P w_{t+j+k} \\
&= \sum_{j=0}^{T-t-k-1} O(\lambda^j \lambda^{2(T-t-j)} \|F^k\|) = O(\lambda^{T-t+k} \|F^k\|).
\end{aligned} \tag{11.17}$$

Finally, we have a formula for  $q_t$  that parallels Eq. (11.15):

$$\begin{aligned}
q_t &= q_{t+1} + \left( d_{t+1} + \sum_{i=k}^{T-t-1} F^{\top i} P w_{t+i} \right)^\top H \left( d_{t+1} + \sum_{i=k}^{T-t-1} F^{\top i} P w_{t+i} \right) \\
&\quad - \left( d_{t+1} + \sum_{i=0}^{T-t-1} F^{\top i} P w_{t+i} \right)^\top H \left( d_{t+1} + \sum_{i=0}^{T-t-1} F^{\top i} P w_{t+i} \right) \\
&\quad + w_t^\top P w_t + 2w_t^\top \left( d_{t+1} + \sum_{i=1}^{T-t-1} F^{\top i} P w_{t+i} \right).
\end{aligned}$$

Taking the difference between  $k$  and  $k + 1$  predictions, we have

$$\begin{aligned}
& q_t^k - q_t^{k+1} - (q_{t+1}^k - q_{t+1}^{k+1}) \\
&= (w_{t+k}^\top P F^k + (d_{t+1}^k - d_{t+1}^{k+1})^\top) H \left( d_{t+1}^k + d_{t+1}^{k+1} + F^\top P w_{t+k} + 2 \sum_{i=1}^{T-t-k-1} F^{\top i+k} P w_{t+i+k} \right) \\
&= (w_{t+k}^\top P F^k + O(\lambda^{T-t} \|F^k\|)) H \left( O(\lambda^{T-t}) + F^\top P w_{t+k} + 2 \sum_{i=1}^{T-t-k-1} F^{\top i+k} P w_{t+i+k} \right),
\end{aligned} \tag{11.18}$$

and thus

$$\mathbb{E}[q_t^k - q_t^{k+1} - (q_{t+1}^k - q_{t+1}^{k+1})] = O(\|F^k\|(\lambda^{T-t} + \|F^k\|)).$$

$$\begin{aligned}
\mathbb{E}[q_0^k - q_0^T] &= \sum_{t=0}^{T-1} \mathbb{E}[q_t^k - q_t^{k+1} - (q_{t+1}^k - q_{t+1}^{k+1})] \\
&= \sum_{t=0}^{T-1} O(\|F^k\|(\lambda^{T-t} + \|F^k\|)) \\
&= O(\|F^k\|^2 T + \|F^k\|).
\end{aligned}$$

$$\mathbb{E}[v_0^k - v_0^T] = 2(d_0^k - d_0^T) = O(\lambda^{T+k} \|F^k\|).$$

$$\begin{aligned}
\mathbb{E} J^{\text{MPC}_k} - \mathbb{E} J^{\text{MPC}_T} &= \mathbb{E}[V_0^k(x_0) - V_0^T(x_0)] \\
&= \mathbb{E}[x_0^\top (v_0^k - v_0^T) + (q_0^k + q_0^T)] \\
&= O(\|F^k\|^2 T + \|F^k\|).
\end{aligned} \tag{11.19}$$

By definition,  $J^{\text{MPC}_T}$  is the cost of MPC policy given all future disturbances before making any decisions. It almost equals to  $\min_u J$ , the optimal policy given all future disturbances, except that during optimization, MPC assumes the final-step cost to be  $x_T^\top P x_T$  instead of  $x_T^\top Q_f x_T$ . This will incur at most constant extra cost, i.e.,

$$J^{\text{MPC}_T} - \min_u J = O(P - Q_f) = O(1). \tag{11.20}$$

By Eqs. (11.19) and (11.20),

$$\text{Reg}^S(\text{MPC}_k) = \mathbb{E} J^{\text{MPC}_k} - \mathbb{E} \min_u J = O(\|F^k\|^2 T + \|F^k\| + 1) = O(\|F^k\|^2 T + 1).$$

□

**Theorem 11.6.** *The optimal dynamic regret  $Reg_k^{S^*} = STO_k^T - STO_T^T = O(\|F^k\|^2 T + 1) = O(\lambda^{2k} T + 1)$  and there exist  $A, B, Q, R, Q_f, x_0$ , and  $W$  such that  $Reg_k^{S^*} = \Theta(\|F^k\|^2 (T - k))$ .*

*Proof.* The first part follows from Theorem 11.5 and that fact that  $Reg_k^{S^*} \leq Reg^S(\text{MPC}_k)$ . The second part is shown by Example 11.3, i.e., suppose  $n = d = 1$  and the disturbance are i.i.d. and zero-mean. Additionally, let  $Q_f = P$  and  $x_0 = 0$ . In this case, MPC has not only the same policy but also the same cost as the optimal control policy. Also,  $P_t = P$  for all  $t$ . To calculate the total cost, we follow the approach used in the proof of Theorem 11.1. Since  $T$  is finite now, we have a similar (to Eq. (11.7)) but different form of  $v_t$ :

$$v_t = 2 \sum_{i=0}^{\min\{k-1, T-t-1\}} F^{\top i+1} P W_{t+i}.$$

Thus,

$$\begin{aligned} \mathbb{E}[q_t] &= \text{Tr} \left\{ \left( P - \sum_{i=0}^{\min\{k-1, T-t-1\}} P F^i H F^{\top i} P \right) W \right\} + \mathbb{E}[q_{t+1}]. \\ \mathbb{E}[q_0] &= \text{Tr} \left\{ \sum_{t=0}^{T-1} \left( P - \sum_{i=0}^{\min\{k-1, T-t-1\}} P F^i H F^{\top i} P \right) W \right\}. \end{aligned}$$

Let  $q_t^k$  denote  $q_t$  in the scenario of  $k$  predictions.

$$\begin{aligned} Reg^{S^*} &= \mathbb{E}[q_0^k - q_0^T] = \text{Tr} \left\{ \sum_{t=0}^{T-k-1} \sum_{i=k}^{T-t-1} P F^i H F^{\top i} P W \right\} \\ &\geq (T - k) \text{Tr} \left\{ P F^k H F^{\top k} P W \right\} = \Omega(\|F^k\|^2 (T - k)). \end{aligned}$$

On the other hand,

$$Reg^{S^*} = \mathbb{E}[q_0^k - q_0^T] \leq (T - k) \text{Tr} \left\{ \sum_{i=k}^{\infty} P F^i H F^{\top i} P W \right\} = O(\|F^k\|^2 (T - k)).$$

Therefore,  $Reg^{S^*} = \Theta(\|F^k\|^2 (T - k))$ . □

Note that, in the stochastic case, the regret-optimal policy is the same as the cost-optimal policy, i.e., the policy for  $STO_k^T$  is the same as  $Reg_k^{S^*}$ .

### Adversarial Disturbance

We now move from stochastic to adversarial disturbances. In this case, the disturbances are chosen from a bounded set  $\Omega \subseteq \mathbb{R}^n$  by an adversary in order to maximize the controller's cost. Maintaining small regret is more challenging in adversarial models than in stochastic ones, so one may expect weaker bounds. Perhaps surprisingly, we obtain bounds with the same order.

**Optimal policy with  $k$  predictions.** In the adversarial setting, the cost of the optimal policy, defined with a sequence of min's and sup's, is the equilibrium value of a two-player zero-sum game. In general, it is impossible to give an analytical expression of either  $\text{ADV}_k$  or the corresponding optimal policy. However, we prove the following result that is structurally similar to the results from the stochastic setting, highlighting the exponential improvement from predictions.

**Theorem 11.7.** *For  $k \geq 1$ ,  $\text{ADV}_k - \text{ADV}_{k+1} = O(\|F^k\|^2) = O(\lambda^{2k})$ .*

*Proof.* This proof is based on Theorem 11.8. It turns out that the behavior of the MPC policy and its cost is easier to analyze than the optimal one, especially in the adversarial setting.

$$\text{ADV}_k - \text{ADV}_{k+1} \leq \text{ADV}_k - \text{ADV}_\infty \leq \text{MPCA}_k - \text{ADV}_\infty = \sum_{i=k}^{\infty} \text{MPCA}_i - \text{MPCA}_{i+1}.$$

By Theorem 11.8,

$$\text{MPCA}_i - \text{MPCA}_{i+1} \leq O(\|F^i\|^2) \leq O(\|F^k\|^2 \|F^{i-k}\|^2) \leq O(\|F^k\|^2 \lambda^{2(i-k)}).$$

Thus,

$$\text{ADV}_k - \text{ADV}_{k+1} \leq O\left(\|F^k\|^2 \sum_{i=k}^{\infty} \lambda^{2(i-k)}\right) = O(\|F^k\|^2).$$

□

Similarly to Example 11.2 for the stochastic case, in the adversarial setting, the optimal cost with  $k$  predictions may approach the offline optimal cost (under infinite predictions) much faster than exponential rate, and it is possible that  $\text{ADV}_k = \text{ADV}_\infty$  for finite  $k$ , as shown in Example 11.4.

**Example 11.4.** *Let  $A = B = Q = R = 1$  and  $\Omega = [-1, 1]$ . In this case, one prediction is enough to leverage the full power of prediction. Formally, we have*

$ADV_1 = ADV_\infty = 1$ . In other words, for all  $k \geq 1$ ,  $ADV_k = 1$ . The optimal control policy (as  $T \rightarrow \infty$ ) is a piecewise function:

$$u^*(x, w) = \begin{cases} -(x + w) & , -1 \leq x + w \leq 1 \\ -(x + w) + \frac{3-\sqrt{5}}{2}(x + w - 1) & , x + w > 1 \\ -(x + w) + \frac{3-\sqrt{5}}{2}(x + w + 1) & , x + w < -1 \end{cases} .$$

The proof leverages two different cost-to-go functions for the min player and the sup player (see Yu et al., 2020, Appendix C.2).

Note that with adversarial disturbances, the optimal policy could be much more complex than stochastic cases. Unlike Example 11.4, where the optimal policy is piecewise linear with only 3 pieces, for other values of  $A, B, Q, R$ , this function may have many more pieces.

**MPC's performance.** Under adversarial disturbances, MPC is suboptimal, e.g., in Example 11.4. However, its dynamic regret bound turn out to be the same as those in the stochastic setting, as shown in following theorems.

**Theorem 11.8.**  $MPCA_k - MPCA_{k+1} = O(\|F^k\|^2) = O(\lambda^{2k})$ .

*Proof.* Note that Eq. (11.16) in the proof of Theorem 11.4 does not rely on the type of disturbance, i.e., Eq. (11.16) holds for adversarial disturbance as well. Let  $r = \sup_{w \in \Omega} \|w\|_2$ .

$$\begin{aligned} q_t^k - q_t^{k+1} - (q_{t+1}^k - q_{t+1}^{k+1}) &= w_{t+k}^\top P F^k H F^{\top k} \left( P w_{t+k} + 2 \sum_{i=1}^{\infty} F^{\top i} P w_{t+i+k} \right) \\ &\leq \|w_{t+k}\| \|P\| \|H\| \|F^k\|^2 \left( \|P\| \|w_{t+k}\| + 2 \sum_{i=1}^{\infty} \|F^i\| \|P\| \|w_{t+i+k}\| \right) \\ &\leq \|F^k\|^2 \left( 1 + 2 \sum_{i=1}^{\infty} \|F^i\| \right) \|H\| \|P\|^2 r^2 \\ &\leq \|F^k\|^2 \left( 1 + 2 \frac{c\lambda}{1-\lambda} \right) \|H\| \|P\|^2 r^2 \end{aligned}$$

for some constant  $c$ .

$$\begin{aligned}
\text{MPCA}_k - \text{MPCA}_{k+1} &= \lim_{T \rightarrow \infty} \frac{1}{T} (\max_w q_0^k - \max_w q_0^{k+1}) \\
&\leq \lim_{T \rightarrow \infty} \frac{1}{T} \max_w (q_0^k - q_0^{k+1}) \\
&\leq \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=0}^{T-1} \max_w (q_t^k - q_t^{k+1} - (q_{t+1}^k - q_{t+1}^{k+1})) \\
&\leq \|F^k\|^2 \left(1 + 2 \frac{c\lambda}{1-\lambda}\right) \|H\| \|P\|^2 r^2 = O(\|F^k\|^2).
\end{aligned}$$

□

**Theorem 11.9** (Main result).  $\text{Reg}^A(\text{MPC}_k) = O(\|F^k\|^2 T + 1) = O(\lambda^{2k} T + 1)$ .

*Proof.* We follow the notations in the proof of Theorem 11.5. Eq. (11.18) does not rely on the type of disturbance, so it holds for adversarial disturbance as well. By Eq. (11.18) and the fact that  $w_t$  is bounded, we have

$$q_t^k - q_t^{k+1} - (q_{t+1}^k - q_{t+1}^{k+1}) = O(\|F^k\|(\lambda^{T-t} + \|F^k\|)),$$

where the constant in the Big-Oh notation does not depend on the disturbance sequence  $w$ . Thus,

$$\max_w (q_0^k - q_0^T) \leq \sum_{t=0}^{T-1} \max_w (q_t^k - q_t^{k+1} - (q_{t+1}^k - q_{t+1}^{k+1})) = O(\|F^k\|^2 T + \|F^k\|).$$

By Eq. (11.17) and the boundedness of  $w_t$ ,

$$\max_w (v_0^k - v_0^T) = 2 \max_w (d_0^k - d_0^T) = O(\lambda^{T+k} \|F^k\|).$$

$$\begin{aligned}
\max_w (J^{\text{MPC}_k} - J^{\text{MPC}_T}) &= \max_w (V_0^k(x_0) - V_0^T(x_0)) \leq \max_w (x_0^\top (v_0^k - v_0^T)) + \max_w (q_0^k - q_0^T) \\
&= O(\|F^k\|^2 T + \|F^k\|).
\end{aligned}$$

As Eq. (11.20),  $J^{\text{MPC}_T} - \min_u J = O(1)$ . Thus,

$$\begin{aligned}
\text{Reg}^A(\text{MPC}_k) &= \max_w (J^{\text{MPC}_k} - \min_u J) \leq \max_w (J^{\text{MPC}_k} - J^{\text{MPC}_T}) + \max_w (J^{\text{MPC}_T} - \min_u J) \\
&= O(\|F^k\|^2 T + \|F^k\| + 1) = O(\|F^k\|^2 T + 1).
\end{aligned}$$

□

This dynamic regret is linear in the horizon  $T$  if we fix the number of predictions. However, if  $k$  is a super-constant function of  $T$ , i.e., an increasing function of  $T$  that is not upper-bounded by a constant, then the regret is sub-linear. Furthermore, if we let  $k = \frac{\log T}{2 \log(1/\lambda)}$ , then  $Reg^A(\text{MPC}_k) = O(1)$ . In other words, we can get constant regret with  $O(\log T)$  predictions, even with adversarial disturbances. Finally, as implied by the following result, the  $O(\log T)$  horizon cannot be improved since even the regret minimizing algorithm needs the same order of predictions to reach constant regret, depicted by the following theorem.

**Theorem 11.10.**  $Reg_k^{A^*} = O(\|F^k\|^2 T + 1) = O(\lambda^{2k} T + 1)$ . Moreover, there exist  $A, B, Q, R, Q_f, x_0$ , and  $\Omega$  such that  $Reg_k^{A^*} = \Omega(\|F^k\|^2 (T - k))$ .

*Proof.* The first part of the theorem follows from Theorem 11.9 and the fact that  $Reg_k^{A^*} \leq Reg^A(\text{MPC}_k)$ . We reduce the second part of this theorem to the second part of Theorem 11.6. Since the proof of Theorem 11.6 works for any fixed distribution of  $w_t$  (with finite second moment), we can restrict that distribution to have bounded support. Denote this bounded support by  $\Omega$ . Then, we have

$$\begin{aligned} Reg_k^{A^*} &= \sup_{w_0, \dots, w_{k-1}} \min_{u_0} \sup_{w_k} \dots \min_{u_{T-k-1}} \sup_{w_{T-1}} \min_{u_{T-k}, \dots, u_{T-1}} \left( J(u, w) - \min_{u'_0, \dots, u'_{T-1}} J(u', w) \right) \\ &\geq \mathbb{E}_{w_0, \dots, w_{k-1}} \min_{u_0} \mathbb{E}_{w_k} \dots \min_{u_{T-k-1}} \mathbb{E}_{w_{T-1}} \min_{u_{T-k}, \dots, u_{T-1}} \left( J(u, w) - \min_{u'_0, \dots, u'_{T-1}} J(u', w) \right) \\ &= Reg_k^{S^*} = \Theta(\|F^k\|^2 (T - k)). \end{aligned}$$

□

### 11.4 Simulations

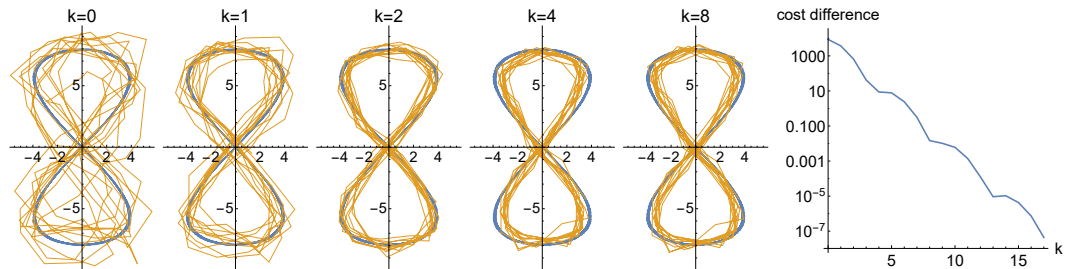


Figure 11.1: The power of predictions in online tracking. The left five figures show the desired trajectory (blue) and the actual trajectories (orange). The rightmost figure shows the cost difference (regret) between MPC using  $k$  predictions and the offline optimal policy. Note that the y-axis of the rightmost figure is in log-scale.

To illustrate our theoretical results, we test MPC with different numbers of predictions in a Linear Quadratic (LQ) tracking problem, where the desired trajectory is

given by:

$$d_t = \begin{bmatrix} 8 \sin(t/3) \cos(t/3) \\ 8 \sin(t/3) \end{bmatrix}.$$

We consider following double integrator dynamics:

$$p_{t+1} = p_t + v_t + h_t, \quad v_{t+1} = v_t + u_t + \eta_t,$$

where  $p_t \in \mathbb{R}^2$  is the position,  $v_t$  is the velocity,  $u_t$  is the control, and  $h_t, \eta_t \sim U[-1, 1]^2$  are i.i.d. noises. The objective is to minimize

$$\sum_{t=0}^{T-1} \|p_t - d_t\|^2 + \|u_t\|^2,$$

where we let  $T = 200$ . This problem can be converted to the standard LQR with disturbance  $w_t$  by letting  $x_t = \begin{bmatrix} p_t \\ v_t \end{bmatrix}$  and  $\tilde{w}_t = \begin{bmatrix} h_t \\ \eta_t \end{bmatrix}$  and then using the reduction in the LQ tracking example (Example 11.1). Note that after the reduction, the disturbances are the combination of a deterministic trajectory and i.i.d. noises, which corresponds to the general stochastic case discussed in Section 11.3.

Figure 11.1 shows the tracking results with MPC using different numbers of predictions. We see that the regret exponentially decreases as the number of predictions increases, which is consistent with our theoretical results.

### 11.5 Extension I: Delayed Inexact Predictions

Throughout Sections 11.2 to 11.4, we focus on LQR systems with *exact predictions* of disturbance  $w_t$ . In this section, we will discuss the influence from *delayed inexact predictions*.

#### Problem Statement

In this section, we use the following model to study the setting with delayed and inexact predictions. Formally, at each step  $t$ , the revealed information is:

$$x_0, u_0, \dots, u_{t-1}, w_0, \dots, w_{t-d-1}, \hat{w}_{t-d|t}, \dots, \hat{w}_{T-1|t},$$

or equivalently,

$$x_0, u_0, \dots, u_{t-1}, x_1, \dots, x_{t-d}, \hat{w}_{t-d|t}, \dots, \hat{w}_{T-1|t},$$

where  $d \geq 0$  is the length of feedback delay, and  $\hat{w}_{s|t}$  is the prediction of  $w_s$  at time  $t$ . Namely, at time step  $t$ , the controller must make the decision  $u_t$  without knowing  $x_{t-d+1}, \dots, x_t$ , but with *noisy* predictions  $\hat{w}_{t-d|t}, \dots, \hat{w}_{T-1|t}$ .



We define  $e_{s|t} = w_s - \hat{w}_{s|t}$  as the estimation error, and we assume that the predictor satisfies  $\|e_{t-d+i|t}\| \leq \epsilon_i \|w_{t-d+i}\|$  for all  $i \geq 0$  and  $t \geq 0$ , where  $\epsilon_i$  is a given parameter that measures the prediction quality at  $i$  steps into the unknown. In this section, we focus on the most challenging adversarial case where both  $w_t$  and  $e_{s|t}$  are adversarial.

Although predictions are available for every time step, those for far future may have bad quality, i.e.,  $\epsilon_i$  is typically large for large  $i$ . Therefore, a good control policy may not use all predictions in the same way: using only the predictions with smaller estimation error may yield better performance. For example, if  $\epsilon_i > 1$ , we can simply let  $\hat{w}_{t-d+i|t} = 0$ , which is a better prediction with  $\epsilon_i = 1$ .

**Competitive ratio.** In this section, we measure the performance using a even stronger metric *competitive ratio*, which bounds the worst-case ratio of the cost of an online policy (Alg, i.e., the MPC policy) to the cost of the optimal offline policy (Opt) with perfect knowledge of  $\{w_t\}_{t=0}^{T-1}$ . Formally, we study the so-called *weak competitive ratio*, which allows for an additive horizon-independent constant factor. We say that a policy is (weakly)  $c$ -competitive if, given  $A, B, Q, R, Q_f$  and  $r$ , for any adversarially and adaptively chosen disturbances  $w_t$  and predictions errors  $e_{s|t}$ , we have  $\text{Alg} \leq c \text{Opt} + \kappa$ , where  $\kappa$  is a constant, independent of  $T$ . We say that the algorithm is *constant competitive* when  $c$  is a constant independent of  $T$ . We use competitive ratio in this section to avoid a path-length term in the bound (a dynamic regret bound typically includes a path-length term, as discussed in Chapter 8).

Under the competitive ratio metric, in general, the error bound assumption ( $\|e_{t-d+i|t}\| \leq \epsilon_i \|w_{t-d+i}\|$ ) has to be multiplicative rather than additive. Consider the case where all disturbances  $w_t$  are zero, but there are nonzero prediction errors. The optimal offline policy incurs zero cost, while any online policy that uses the (wrong) predictions incurs nonzero cost, hence leading to an infinite ratio.

**A myopic variant of MPC.** In this section we consider a straightforward myopic extension of the classic MPC algorithm to deal with delayed inexact predictions. In the case without delay ( $d = 0$ ), suppose the controller uses  $k$  predictions. At each time  $t$ , the controller optimizes based on  $x_t, \hat{w}_{t|t}, \dots, \hat{w}_{t+k-1|t}$ :

$$(u_t, \dots, u_{t+k-1}) = \arg \min_u \left( \sum_{i=t}^{t+k-1} (x_i^\top Q x_i + u_i^\top R u_i) + x_{t+k}^\top \tilde{Q}_f x_{t+k} \right),$$

$$\text{s.t. } x_{i+1} = A x_i + B u_i + \hat{w}_{i|t}, \quad \forall i = t, \dots, t+k-1.$$

This optimization is myopic in the sense that it assumes that the length of the problem is  $k$  instead of  $T$  and treats predicted future disturbances as true disturbances. Again,

the terminal cost matrix  $\tilde{Q}_f$  may or may not be the same as the terminal cost matrix  $Q_f$  of the original problem, and can be viewed as a hyperparameter. Similarly,  $k$  is also a hyperparameter. Larger  $k$  is not necessarily better because the predictions in the far future may have very large errors. In this section, similar to Algorithm 11.1, we let  $\tilde{Q}_f = P$ , where  $P$  is the solution of the discrete algebraic Riccati equation (DARE).

Now let us discuss the extension of this myopic MPC policy to the case with delay ( $d > 0$ ). When  $k \geq d$ , the extension is perhaps straightforward. Here, although the controller does not know the current state  $x_t$ , it knows  $x_{t-d}$  and  $\hat{w}_{t-d|t}, \dots, \hat{w}_{t-1|t}$ . Thus, it can estimate the current state. This means that it is possible to simply use this estimation,  $\hat{x}_{t|t}$ , as a replacement for  $x_t$  in the algorithm, which yields the following:

$$u_t = -(R + B^\top PB)^{-1} B^\top \left( PA\hat{x}_{t|t} + \sum_{i=0}^{k-d-1} F^\top{}^i P\hat{w}_{t+i|t} \right). \quad (11.21)$$

When  $k < d$ , the extension is not as obvious. In this setting, the quality of the predictions is poor enough that it is better not to use the predictions to estimate the current state. Thus, one cannot simply estimate the current state and run classic MPC. In this case, the key is to view (classic) MPC from a different perspective: MPC locally solves an optimal control problem by treating known disturbances (predictions) as exact, and treating unknown disturbances as zero. Following this philosophy, in the case when predictions are not enough to be used to estimate the current state, we can instead assume that unknown disturbances are exactly zero. The following theorem derives the optimal policy under this ‘‘optimistic’’ assumption.

**Theorem 11.11.** *Suppose there are  $d$  delays and  $k$  exact predictions with  $k < d$ . The myopic MPC policy assumes all used predictions are exact and other disturbances (with unused predictions) are zero. Its policy at time  $t$  is:*

$$u_t = -(R + B^\top PB)^{-1} B^\top PA \left( A^{d-k} \hat{x}_{t-d+k|t} + \sum_{i=0}^{d-k-1} A^i B u_{t-1-i} \right). \quad (11.22)$$

*Proof.* See Yu et al. (2022). □

In other words, the policy in (11.22) first obtains the greedy estimation  $\hat{x}_{t-d+k|t}$  using predictions  $\hat{w}_{t-d|t}, \dots, \hat{w}_{t-d+k-1|t}$ , and then estimates the current state by treating  $w_{t-d+k} = \dots = w_{t-1} = 0$ . In fact, instead of treating them as zero, we can

impose other values or distributions on those disturbances. This would generalize Theorem 11.11 to a broader class of policies.

To summarize the two cases above ( $k \geq d$  and  $k < d$ ), the myopic generalization of MPC we study in this section is described as follows. Suppose we want to use  $k$  predictions. If  $k \geq d$ , then we estimate the current state  $x_t$  and apply (11.21). If  $k < d$ , then we estimate the state at time  $t - d + k$  and apply (11.22). In fact, the two cases coincide when  $k = d$ .

### Main Results

Our main result provides bounds on the competitive ratio for the myopic MPC-like policy in the case of inexact delayed predictions. We present our general result below and then discuss the special cases of (i) exact predictions and no delay, (ii) inexact predictions and no delay, and (iii) delay but no access to predictions. The special cases illustrate the contrast between inexact and exact predictions as well as the impact of delay. All proofs and more detailed discussions can be found in Yu et al. (2022).

**Theorem 11.12** (General result). *Suppose there are  $d$  steps of delays and the controller uses  $k$  predictions. Define  $c = \|P\| \|P^{-1}\| (1 + \|F\|)$ . When  $k \geq d$ ,*

$$\text{Alg} \leq \left[ \frac{\left( c \sum_{i=0}^{d-1} \epsilon_i \|A^{d-i}\| + c \sum_{i=d}^{k-1} \epsilon_i \|F^{i-d}\| + \|F^{k-d}\| \right)^2}{\|H\|^{-1} \lambda_{\min}(P^{-1} - FP^{-1}F^\top - H)} + 1 \right] \text{Opt} + O(1).$$

When  $k \leq d$ ,

$$\text{Alg} \leq \left[ \frac{\left( c \sum_{i=0}^{k-1} \epsilon_i \|A^{d-i}\| + c \sum_{i=k}^{d-1} \|A^{d-i}\| + 1 \right)^2}{\|H\|^{-1} \lambda_{\min}(P^{-1} - FP^{-1}F^\top - H)} + 1 \right] \text{Opt} + O(1).$$

The  $O(1)$  is with respect to  $T$ . It may depend on the system parameters  $A, B, Q, R, Q_f$  and the range of disturbances  $r$ , but not on  $T$ . When  $Q_f = P$  the  $O(1)$  is zero.

The two cases in Theorem 11.12 correspond to the two cases in the algorithm: when predictions are of high enough quality to allow estimation of the current state and when they are not. In the first case, we see that the quality of predictions in the near future has more impact, especially when  $\rho(A) > 1$ . In the second case, we see that the amount of delay  $d$  exponentially increases the bound if  $\epsilon_i > 0$  and  $\rho(A) > 1$ .

**Theorem 11.13** (Exact predictions without delay). *Suppose there are  $k$  exact predictions and no feedback delay. Then:*

$$\text{Alg} \leq \left[ 1 + \frac{\|F^k\|^2 \|H\|}{\lambda_{\min}(P^{-1} - FP^{-1}F^\top - H)} \right] \text{Opt} + O(1).$$

In other words, Theorem 11.13 gives a competitive ratio result for the standard setting discussed in Sections 11.2 and 11.3. In particular, the competitive ratio exponentially decreases to 1 as  $k$  goes up.

**Theorem 11.14** (Inexact predictions without delay). *Suppose there are  $k$  inexact predictions and no feedback delay. Then,*

$$\text{Alg} \leq \left[ \frac{\|H\| \left( c \sum_{i=0}^{k-1} \epsilon_i \|F^i\| + \|F^k\| \right)^2}{\lambda_{\min}(P^{-1} - FP^{-1}F^\top - H)} + 1 \right] \text{Opt} + O(1).$$

Theorem 11.14 differs from the previous one in that the controller can minimize the bound with respect to  $k$  according to the prediction quality. We characterize this optimization in the following result in 1-d systems.

**Corollary 11.1.** *Suppose there are  $k$  inexact predictions and no feedback delay. Assume  $n = m = 1$ . Given non-decreasing  $\{\epsilon_i\}$ , to minimize the competitive ratio bound in Theorem 11.14, the optimal number  $k$  of predictions to use is such that:*

$$\epsilon_{k-1} < \frac{1 - |F|}{1 + |F|} < \epsilon_k.$$

Finally, we discuss the case  $d > 0$  and  $k = 0$  (i.e., delay without predictions).

**Theorem 11.15** (Delay without predictions). *Suppose there are  $d$  delays and no predictions are available. Then the competitive ratio is bounded by*

$$\text{Alg} \leq \left[ \frac{\|H\| \left( c \sum_{i=1}^d \|A^i\| + 1 \right)^2}{\lambda_{\min}(P^{-1} - FP^{-1}F^\top - H)} + 1 \right] \text{Opt} + O(1).$$

### Beyond Myopic MPC: A Robustness-Consistency Perspective

So far, in this section we have been focusing on a *myopic* MPC policy. Namely, the policy will assume that all predictions  $\hat{w}_{t+i|t}$  are true disturbances.

Intuitively, when designing an online control or learning algorithm with predictions, we want to balance “*consistency*,” which measures the competitive ratio when

predictions are accurate, and “*robustness*,” which bounds the competitive ratio when predictions are inaccurate. In online learning and algorithm community, such a balance is called the *robustness-consistency trade-off* (commonly discussed in online caching, ski-rental, online set cover and online matching problems).

Obviously, the classic (myopic) MPC approach is a consistent algorithm, i.e., its competitive ratio is small when predictions are accurate. However, it might not be a robust algorithm, if the predictions are inaccurate. Therefore, we propose a  $\beta$ -confident algorithm (see more details in T. Li et al. (2022)) such that we can optimally balance between robustness and consistency:

$$u_t = -(R + B^\top P B)^{-1} B^\top \left( P A x_t + \beta \sum_{\tau=t}^{T-1} (F^\top)^{\tau-t} P \hat{w}_\tau \right) \quad (11.23)$$

where the main difference from the classic myopic MPC is that there is a tunable parameter  $\beta \in [0, 1]$ . In T. Li et al. (2022), we proposed a self-tuning algorithm to optimally tune this  $\beta$  parameter in an online manner, based on the prediction quality of all previous predictions.

## 11.6 Extension II: Time-Variant Systems and General Costs

In all previous sections, we analyzed the non-asymptotic performance (using dynamic regret or competitive ratio as the metric) of Model Predictive Control (MPC) in time-invariant LQ systems. Namely, the dynamics is  $x_{t+1} = A x_t + B u_t + w_t$  and the cost is  $x_t^\top Q x_t + u_t^\top R u_t$ , where  $A, B, Q, R$  are time-invariant matrices.

Naturally, the reader might wonder how much our strong guarantees for MPC rely on the structure of the simple LQ system. In particular, does the exponential decaying property (i.e., the dynamic regret or competitive ratio bound exponentially improves as the number of predictions increases) still hold in more complicated systems?

In this section, we give an affirmative answer to the question above, by briefly introducing the results in Lin, Hu, et al. (2021) for linear time-variant (LTV) systems with general time-variant well-conditioned costs.

### Problem Statement and Assumptions

We consider a finite-horizon discrete-time online control problem with LTV dynamics, time-varying costs, and disturbances, namely

$$\begin{aligned} \min_{x_0:T, u_0:T-1} \quad & \sum_{t=1}^T (f_t(x_t) + c_t(u_{t-1})) \\ \text{s.t.} \quad & x_t = A_{t-1}x_{t-1} + B_{t-1}u_{t-1} + w_{t-1}, t = 1, \dots, T, \\ & x_0 = x(0), \end{aligned} \quad (11.24)$$

where  $x_t \in \mathbb{R}^n$ ,  $u_t \in \mathbb{R}^m$ , and  $w_t \in \mathbb{R}^n$  denote the state, the control action, and the disturbance of the system at time steps  $t = 1, \dots, T$ , and  $x(0) \in \mathbb{R}^n$  is a given initial state. By convention, the hitting cost function  $f_t : \mathbb{R}^n \rightarrow \mathbb{R}_+$  and control cost function  $c_t : \mathbb{R}^m \rightarrow \mathbb{R}_+$  are assumed to be time-varying and well-conditioned. Define the tuple  $\vartheta_t := (A_t, B_t, w_t, f_{t+1}, c_{t+1})$ .

We assume that the algorithm has access to the exact predictions of disturbances, cost functions and dynamical matrices in the future  $k$  time steps (which are time-varying), i.e., the event sequence is

$$x_0, \vartheta_0, \vartheta_1, \dots, \vartheta_{k-1}, u_0, \vartheta_k, u_1, \vartheta_{k+1}, \dots, u_{T-k-1}, \vartheta_{T-1}, u_{T-k}, u_{T-k+1}, \dots, u_{T-1}.$$

Here we assume all predictions are *exact*.

**Assumption 11.2.** *We assume all cost functions  $f_t(\cdot), c_t(\cdot)$  are twice continuously differentiable and well-conditioned (i.e., both strongly convex and strongly smooth, see definitions in Chapter 8). Moreover, we assume they are non-negative and  $f_t(0) = 0, c_t(0) = 0$ .*

*We assume  $A_t, B_t$  and  $B_t^\dagger$  are uniformly bounded, and the LTV system  $\{A_t, B_t\}$  are uniformly  $(d, \delta)$ -controllable (the controllability index is  $d$  and the controllability matrix's minimum singular value is uniformly lower bounded by  $\delta$ ).*

### Main Results

Suppose all above assumptions hold, we have the following informal theorem for the performance of MPC:

**Theorem 11.16** (Informal, see the formal version in Lin, Hu, et al. (2021)). *Suppose the terminal cost  $V(\cdot)$  in MPC is either the indicator function of the origin, or a non-negative convex function that is twice continuously differentiable and satisfies*

$V(0) = 0$ . If the prediction window  $k \geq d$  is sufficiently large (larger than a constant depending on the system parameter, but not on  $T$ ), we have:

$$\text{Alg} - \text{Opt} = O(\bar{\lambda}^k T + 1)$$

where  $0 \leq \bar{\lambda} < 1$ . Moreover, if the terminal cost is the indicator function of the origin, and the prediction window is sufficiently large, we have:

$$\text{Alg} \leq \left(1 + O(\bar{\lambda}^k)\right) \cdot \text{Opt}.$$

Generally speaking, Theorem 11.16 shows that the exponentially decaying property still holds even in highly time-variant systems, as long as the prediction window is large enough.

**Proof sketch.** In all previous sections, we analyzed the dynamic regret or competitive ratio of MPC via *direct* methods. Namely, we directly characterize the MPC policy's structure and the offline optimal policy's structure, and then compare their difference. However, it is very hard (if not impossible) to apply the direct method in time-variant systems because such characterizations are in general intractable. Therefore, to prove Theorem 11.16, Lin, Hu, et al. (2021) uses a novel proof framework based on a perturbation bound that characterizes how a small change to the system parameters impacts the optimal trajectory. However, even under the perturbation analysis framework, analyzing LTV systems directly is still hard, so we instead develop a novel reduction from LTV systems to fully-actuated systems, i.e., systems where the controller can steer the system to any state in the whole space  $\mathbb{R}^n$  freely at every time step. This special case is a form of online optimization called *smoothed online convex optimization* (SOCO). Such a reduction is inspired by Chapter 10.

## References

- Abbasi-Yadkori, Yasin and Csaba Szepesvári (2011). *Regret bounds for the adaptive control of linear quadratic systems*. In: *Proceedings of the 24th Annual Conference on Learning Theory*. JMLR Workshop and Conference Proceedings, pp. 1–26.
- Agarwal, Naman, Brian Bullins, Elad Hazan, Sham M. Kakade, and Karan Singh (2019). *Online control with adversarial disturbances*. In: *International Conference on Machine Learning (ICML)*.
- Agarwal, Naman, Elad Hazan, and Karan Singh (2019). *Logarithmic regret for online control*. In: *Neural Information Processing Systems (NeurIPS)*.

- Anderson, Brian D. O. and John B. Moore (2007). *Optimal control: Linear quadratic methods*. Courier Corporation.
- (2012). *Optimal filtering*. Courier Corporation.
- Angeli, David, Rishi Amrit, and James B. Rawlings (2011). *On average performance and stability of economic model predictive control*. In: *IEEE Transactions on Automatic Control* 57.7, pp. 1615–1626.
- Angeli, David, Alessandro Casavola, and Francesco Tedesco (2016). *Theoretical advances on economic model predictive control with time-varying costs*. In: *Annual Reviews in Control* 41, pp. 218–224.
- Baca, Tomas, Daniel Hert, Giuseppe Loianno, Martin Saska, and Vijay Kumar (2018). *Model predictive trajectory tracking and collision avoidance for reliable outdoor deployment of unmanned aerial vehicles*. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 6753–6760.
- Camacho, Eduardo F. and Carlos Bordons Alba (2013). *Model predictive control*. Springer Science & Business Media.
- Cassel, Asaf, Alon Cohen, and Tomer Koren (2020). *Logarithmic regret for learning linear quadratic regulators efficiently*. In: *arXiv preprint arXiv:2002.08095*.
- Chen, Niangjun, Anish Agarwal, Adam Wierman, Siddharth Barman, and Lachlan Andrew (2015). *Online convex optimization using predictions*. In: *Proceedings of ACM SIGMETRICS*, pp. 191–204.
- Cohen, Alon, Tomer Koren, and Yishay Mansour (2019). *Learning linear-quadratic regulators efficiently with only  $\sqrt{T}$  regret*. In: *International Conference on Machine Learning (ICML)*.
- Dean, Sarah, Horia Mania, Nikolai Matni, Benjamin Recht, and Stephen Tu (2020). *On the sample complexity of the linear quadratic regulator*. In: *Foundations of Computational Mathematics* 20.4, pp. 633–679.
- Fazel, Maryam, Rong Ge, Sham M. Kakade, and Mehran Mesbahi (2018). *Global convergence of policy gradient methods for the linear quadratic regulator*. In: *arXiv preprint arXiv:1801.05039*.
- Foster, Dylan J. and Max Simchowitz (2020). *Logarithmic regret for adversarial online control*. In: *arXiv preprint arXiv:2003.00189*.
- Goel, Gautam and Babak Hassibi (2020). *The power of linear controllers in LQR control*. In: *arXiv preprint arXiv:2002.02574*.
- Goel, Gautam and Adam Wierman (2019). *An online algorithm for smoothed regression and LQR control*. In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*.



- Grüne, Lars and Simon Pirkelmann (2018). *Economic model predictive control for time-varying system: Performance and stability results*. In: *Optimal Control Applications and Methods*.
- Grüne, Lars and Marleen Stieler (2014). *Asymptotic stability and transient optimality of economic MPC without terminal conditions*. In: *Journal of Process Control* 24.8, pp. 1187–1196.
- Hazan, Elad, Sham M. Kakade, and Karan Singh (2020). *The nonstochastic control problem*. In: *Conference on Algorithmic Learning Theory (ALT)*.
- Kirk, Donald E. (2004). *Optimal control theory: An introduction*. Courier Corporation.
- Lazic, Nevena, Craig Boutilier, Tyler Lu, Eehern Wong, Binz Roy, Moonkyung Ryu, and Greg Imwalle (2018). *Data center cooling using model-predictive control*. In: *Advances in Neural Information Processing Systems*, pp. 3814–3823.
- Li, Tongxin, Ruixiao Yang, Guannan Qu, Guanya Shi, Chenkai Yu, Adam Wierman, and Steven Low (2022). *Robustness and consistency in linear quadratic control with untrusted predictions*. In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 6.1, pp. 1–35. DOI: 10.1145/3508038.
- Li, Yingying, Xin Chen, and Na Li (2019). *Online optimal control with linear dynamics and predictions: Algorithms and regret analysis*. In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 14858–14870.
- Lin, Yiheng, Gautam Goel, and Adam Wierman (2020). *Online optimization with predictions and non-convex losses*. In: *Proceedings of ACM SIGMETRICS*.
- Lin, Yiheng, Yang Hu, Guanya Shi, Haoyuan Sun, Guannan Qu, and Adam Wierman (2021). *Perturbation-based regret analysis of predictive control in linear time varying systems*. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 34. Curran Associates, Inc., pp. 5174–5185. URL: <https://arxiv.org/abs/2106.10497>.
- Rosolia, Ugo and Francesco Borrelli (2017). *Learning model predictive control for iterative tasks. A data-driven control framework*. In: *IEEE Transactions on Automatic Control* 63.7, pp. 1883–1896.
- (2019). *Sample-based learning model predictive control for linear uncertain systems*. In: *arXiv preprint arXiv:1904.06432*.
- Simchowitz, Max and Dylan Foster (2020). *Naive exploration is optimal for online LQR*. In: *International Conference on Machine Learning*. Proceedings of Machine Learning Research, pp. 8937–8948.
- Vazquez, Sergio, Jose Rodriguez, Marco Rivera, Leopoldo G. Franquelo, and Margarita Norambuena (2016). *Model predictive control for power converters and drives: Advances and trends*. In: *IEEE Transactions on Industrial Electronics* 64.2, pp. 935–947.

- Yu, Chenkai, Guanya Shi, Soon-Jo Chung, Yisong Yue, and Adam Wierman (2020). *The power of predictions in online control*. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 33. Curran Associates, Inc., pp. 1994–2004. URL: <https://proceedings.neurips.cc/paper/2020/file/155fa09596c7e18e50b58eb7e0c6ccb4-Paper.pdf>.
- (2022). *Competitive control with delayed imperfect information*. In: *American Control Conference (ACC)*. URL: <https://arxiv.org/abs/2010.11637>.
- Zhou, Kemin and John Comstock Doyle (1998). *Essentials of robust control*. Vol. 104. Prentice Hall Upper Saddle River, NJ.

## Chapter 12

### DISCUSSION AND FUTURE WORK

In Part II, through Chapters 9 to 11, we have introduced three unifying interfaces between learning and control theory. See Fig. 8.1 in Chapter 8 for a summary. In this chapter, we will conclude Part II by discussing several important future research directions.

#### **Provably Safe Lifelong Learning in Real-World Autonomous Systems**

As one of the most challenging open problems in AI, lifelong learning considers systems that can continually learn many tasks over a lifetime. Lifelong learning already poses several fundamental problems (e.g., catastrophic forgetting), let alone considering the safety-critical real-world setting (e.g., aircraft control with self-improvement over a lifetime).

Note that Chapters 5 and 9 can be viewed as a “one-shot” and supervised approximation of lifelong learning, where a representation shared by several tasks is learned first (with supervision) and then adapted by adaptive control. Towards unsupervised and continual lifelong learning with safety guarantees, the goal is to systematically address the following questions: How to distill knowledge from previous tasks without supervision? How to design an “event-triggered” mechanism to selectively and safely transfer the distilled knowledge to new tasks?

#### **Encoding Control-Theoretic Knowledge to Reinforcement Learning**

Most popular RL algorithms (e.g., TRPO, SAC) are *universal* for all tasks. In contrast, drastically different control methods are developed for different systems/tasks, and their successes highly rely on structures inside these systems/tasks. For example, Chapter 11 shows that the strong non-asymptotic guarantees of MPC are not only from the receding horizon nature, but also from the structure of the underlying dynamical systems and the exponential closed-loop stability property.

One interesting and important research direction is to encode these structures and algorithmic principles into black-box RL algorithms, which will potentially make RL algorithms much more data-efficient, robust, interpretable, and safe.

### **Sample Complexity Analysis for Nonlinear Systems**

Existing sample complexity results for dynamical systems focus on linear dynamics, but most real-world systems are highly nonlinear. Significantly more challenges appear from nonlinearity. For example, it is often intractable to characterize the closed-loop behavior of the optimal policy. Chapter 10 serves as an initial step by considering LTV systems and some particular classes of nonlinear systems. One important yet challenging future research topic is to have end-to-end guarantees for uncertain nonlinear systems.

### **Hierarchical and Layered Learning and Control**

Having different levels of abstractions is crucial for autonomous systems, but how to design each level in a data-driven manner is unclear. As an initial step, Chapters 5 and 9 build on a 2-layer structure. Namely, an outer loop optimizes a shared representation and an inner loop fine-tunes the representation in a low dimensional space. The future direction is to study general algorithmic principles and convergence properties in general multi-layer learning and control settings.