Trajectory Generation for Nonlinear Control Systems

Thesis by
Michiel J. van Nieuwstadt

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

California Institute of Technology
Pasadena, California

1997

(Defended July 18, 1996)

Acknowledgements

My first encounter with Caltech was in 1991, when I was an intern in the section of Robotics and Automation at the Jet Propulsion Laboratory in Pasadena. My advisor at the University of Twente, Prof. Arun Bagchi, and Prof. Joel Burdick at Caltech, were indispensable in arranging this internship. My group supervisor at JPL, Dr. Guillermo Rodriguez, made the internship a rewarding experience.

At Caltech, my advisor Prof. Richard Murray provided me with support and opportunities to conduct the research reported in this dissertation. He also provided countless intellectual challenges and pointed out new and interesting directions of research. It goes without saying that witout his help and encouragement this thesis would not have been.

My wife Line was not my wife when I started this endeavor. It required quite a great deal of confidence on her side to pursue our relationship to the point where it is now. I thank her for her support and love.

My parents never stopped supporting me and encouraging me to find my own way. My second set of parents here in Los Angeles did a great deal to make me feel at home. I owe them my deepest gratitude.

I thank my fellow students, Sudipto Sur, Bob Behnken, Francesco Bullo, Simon Yeung, and many others for their company, help, time, and stimulating discussions. Dr. Willem Sluis introduced me to exterior differential systems, and we had many enlightening discussions about exterior differential systems and life in general. The initial work on flatness in a differential geometric setting was performed in collaboration with Muruhan Rathinam. Mark Milam answered my many questions about aerodynamics, and did a terrific job designing and building the second generation ducted fan used for the experiments in Chapter ??. Dr. John Morris initiated me to the model helicopter, and Xiaoyun Zhu helped me build the second generation of the helicopter hardware. Dr. Andrew Lewis let me use his LaTeX style file citstyle.cls and provided much other LaTeX help.

Trajectory Generation for Nonlinear Control Systems

by
Michiel J. van Nieuwstadt

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

Abstract

This thesis explores the paradigm of two degree of freedom design for nonlinear control systems. In two degree of freedom design one generates an explicit trajectory for state and input around which the system is linearized. Linear techniques are then used to stabilize the system around the nominal trajectory and to deal with uncertainty. This approach allows the use of the wealth of tools in linear control theory to stabilize a system in the face of uncertainty, while exploiting the nonlinearities to increase performance. Indeed, this thesis shows through simulations and experiments that the generation of a nominal trajectory allows more aggressive tracking in mechanical systems.

The generation of trajectories for general systems involves the solution of two point boundary value problems which are hard to solve numerically. For the special class of differentially flat systems there exists a unique correspondence between trajectories in the output space and the full state and input space. This allows us to generate trajectories in the lower dimensional output space where we don't have differential constraints, and subsequently map these to the full state and input space through an algebraic procedure. No differential equations have to be solved in this process. This thesis gives a definition of differential flatness in terms of differential geometry, and proves some properties of flat systems. In particular, it is shown that differential flatness is equivalent to dynamic feedback linearizability in an open and dense set.

This dissertation focuses on differentially flat systems. We describe some interesting trajectory generation problems for these systems, and present software to solve them. We also present algorithms and software for real time trajectory generation, that allow a computational tradeoff between stability and performance. We prove convergence for a rather general class of desired trajectories. If a system is not differentially flat we can approximate it with a differentially flat system, and extend the techniques for flat systems. The various extensions for approximately flat systems are validated in simulation and experiments on a thrust vectored aircraft. A system may exhibit a two layer structure where the outer layer is a flat system, and the inner system is not. We call this structure *outer flatness*. We investigate trajectory generation for these systems and present theorems on the type of tracking we can achieve. We validate the outer flatness approach on a model helicopter in simulations and experiment.

Contents

List of Figures

List of Tables

List of Symbols

Symbol	**: Description and page when applicable**
$\mathcal{I}$	: algebraic ideal generated by a Pfaffian system, 15
α	: angle of attack [rad] or [deg], 67
U	: body x velocity [m/s], 67
V	: body y velocity [m/s], 67
W	: body z velocity [m/s], 67
$\bar{c}$	: mean aerodynamic chord [m], 67
δ_a	: (incremental) aileron, 99
δ_e	: (incremental) elevator, 99
δ_r	: (incremental) rudder, 99
ρ	: density, [kg/m^3], 67
D	: drag [N], 68
C_d	: drag coefficient, 68
fpara	: force parallel to fan shroud, 57
fperp	: force perpendicular to fan shroud, 57
g	: gravitational acceleration, 96
L	: lift [N], 68
C_L	: lift coefficient, 68
T_b	: main rotor thrust, 95
m_g	: gravitational mass [kg], 67
M	: moment [Nm], 68
C_M	: moment coefficient, 68
J	: moment of inertia [kg m^2], 67
m_x	: inertial mass in the x direction [kg], 67
m_y	: inertial mass in the y direction [kg], 67
m_z	: inertial mass in the z direction [kg], 67
p	: roll rate, 99
I	: Pfaffian system, 15
ϕ	: roll angle, 95
Q	: pitch rate [rad/s], 67
ψ	: yaw angle, 95

q : pitch rate, 99
r : yaw rate, 99
S : eventually constant signals, 51
τ_b : tail rotor torque, 95
θ_d : desired θ position, 57
θ : pitch angle, 95
u : system input, 112
v_x : horizontal spatial velocity, 74
v_{xd} : desired horizontal spatial velocity, 74
S : wing surface [m^2], 67
x : system state, 112
x_d : desired x position, 57
x : horizontal position, 30
y : system output, 112
y_d : desired y position, 57
y : vertical position, positive up, 30
z : flat outputs, 36
z_d : desired z position, 57
z : vertical position, positive down, 30

Chapter 1

Introduction

Over the past four decades the field of control theory has witnessed an incredible growth in theory and tools. Much of the success of the theory can be attributed to the development of software that made this theory accessible to the practicing control engineer. Without these tools, the theory would have been just that: theory. The majority of these tools apply to linear control theory. Even though the nonlinear theory has witnessed substantial development, it has not been accompanied by computational tools that make the theory accessible. It is the author's belief that software tools are an essential part of the development of a new theory. New paradigms need to be continuously validated in simulation and experiment, therefore experimental validation takes a prominent place in this work.

This thesis is a first step in the development of software tools for certain classes of nonlinear systems. The paradigm we advocate is the so called two degree of freedom design. This paradigm entails explicit generation of a nominal state space and input trajectory using the full nonlinear system description, and the use of linear theory to deal with uncertainty and to stabilize around this trajectory. It is shown through experiments and simulation that stabilizing around a nominal trajectory allows a more aggressive response for nonlinear systems.

1.1 An Overview of Trajectory Tracking Methods

This section will use some technical terms from control theory that we loosely introduce here. We summarize the technical details and precise definitions of nonlinear geometric control theory in Appendix **??**. The reader unfamiliar with the concepts presented here is referred to that appendix.

Trajectory tracking is an important problem in nonlinear and linear systems theory alike. It is most prominent in the control of mechanical systems, where we want the outputs of the system to follow a prescribed path. Important examples of mechanical systems where trajectory tracking is important are vehicles and robotic manipulators. Trajectory tracking is less common in the control of distributed parameter systems, like compressors, combustors and acoustic systems. Trajectory tracking methods can roughly be divided in two classes: methods that compute explicitly a nominal trajectory for the state space, and those that don't.

In this dissertation we are not interested in trajectories generated as output of another system. One particular instance of this latter problem is the *model matching* problem, where we are interested in following all trajectories generated by a reference system subjected to the same input as our plant. For linear systems this problem is widely studied [?]. For nonlinear systems some initial work has been done in [?]. Some researchers study the problem of tracking a trajectory generated by an *exosystem* subjected to one particular input [?]. It is our opinion that this problem is merely of academic interest. In practice the desired trajectories are not generated by exosystems, but rather given to us as independent entities. Hence we will devote our attention to the case where the desired trajectory is generated by arbitrary means.

The most straightforward approach to the problem of trajectory tracking is the one advocated in this thesis: compute a nominal path for the state of the system that has the desired output, and try to regulate the system around this path. This approach contains two distinct parts: the computation of the nominal trajectory, and the design of a controller that tries to keep the system on the trajectory. For obvious reasons this approach is called *two degree of freedom design.* This is to be contrasted with the *one degree of freedom design* , where one only steers to a nominal output trajectory, while not caring what the entire state does, as long as the desired output is followed. The desired output trajectory does in general not determine the full state, and the two degree of freedom design uses more knowledge of the system than the one degree of freedom design. It can therefore be expected that better performance will result if we control the system around a nominal state, rather than a nominal output. Indeed, we will show in this thesis that the two degree of freedom design yields superior performance for trajectory tracking. For two degree of freedom design we use explicit *trajectory generation* to achieve *trajectory tracking.* This dissertation is concerned with the problem of trajectory generation.

For linear systems without *right half plane zeros*, trajectory tracking can be accomplished quite simply without resort to optimal control theory. This can be done by writing a differential equation for the error between the output and the desired output and selecting a feedback that make this differential equation asymptotically stable around the origin. The error converging to zero is equivalent to the output tracking the desired output. It might be that the full system has *internal dynamics* that are not visible from the output. The requirement that the linear system have no right half plane zeros guarantees that these internal dynamics are stable. If it so happens that there are no internal dynamics, this method generates a full state space trajectory from the desired output and its derivatives. This happens when the outputs and their derivatives determine the trajectories for all states. If there are internal dynamics, the state space trajectory is not fully determined.

With the advent of nonlinear geometric control theory, the problem of trajectory tracking for nonlinear systems made great progress. It was realized that some nonlinear systems could be transformed into linear systems by a coordinate transformation on the states and a special control law [?, ?]. This process is called *feedback linearization.* The same procedure as for linear systems would then ensure trajectory tracking. One would simply transform the desired output trajectory to

linear coordinates, and the resulting stabilizing control law for the error system back to nonlinear coordinates. As in the linear case, the full system could have internal dynamics that were not visible from the output. For the linearizing scheme to work, these internal dynamics would have to be stable. In the linear case this is guaranteed by the requirement that all zeros be in the left half plane. For nonlinear systems, we call the equivalent property *minimum phase* zero dynamics. Again, a full state space trajectory is generated in the linearized coordinates if it so happens that there are no internal dynamics.

If there are internal dynamics, one can try to extend the trajectory for the outputs and their derivatives to a full state space trajectory. One such method is reported by Chen et al. in the papers [?, ?], and by Devasia in [?]. The method is called *noncausal inversion* for trajectory generation for systems with unstable zero dynamics. This method requires the system to have well defined relative degree and hyperbolic zero dynamics, i.e. no eigenvalues on the imaginary axis. In the absence of imaginary eigenvalues, the zero dynamics manifold can be split into a stable and an unstable manifold. The method of noncausal inversion tries to find a stable solution for the full state space trajectory by steering from the unstable zero dynamics manifold to the stable zero dynamics manifold. The noncausality results from the fact that we first have to get from the origin to the right position on the unstable zero dynamics manifold. The solution is found by repeatedly solving a two point boundary value problem for the linearized zero dynamics driven by the desired trajectory. At each step a system of differential equations has to be solved, and computational requirements are heavy.

This iteration can also be performed in the frequency domain, as shown by Meyer et al. in [?]. This is because the solution of differential equations in the time domain can be done through integrating the desired output with a convolution kernel. This convolution corresponds to multiplication in the Fourier domain. In [?], the method is applied to flight path generation between via points for commercial aircraft. The update rate of via points is in the order of several minutes, which is long enough to allow steering on the unstable zero dynamics manifold. If the input is provided by a pilot in real time, the computational requirements and acausality might be prohibitive.

Finally, an approach that does not generate a feasible state space trajectory, but improves on the output-only trajectory has been explored by Getz et al. in [?]. The method generates an approximate trajectory for the internal dynamics by following an instantaneous equilibrium for the internal dynamics. The first and higher order derivatives of the internal states are set to zero. Therefore the total state trajectory is not feasible. Further refinements of this technique can be found in [?].

In this thesis we investigate fast computational methods of generating full state space trajectories from output trajectories for differentially flat systems, or derivatives thereof. Differentially flat systems are systems that exhibit a one-to-one correspondence between output trajectories and full state space and input trajectories. Trajectories can be planned in output space and then lifted to the state and input space, through an algebraic mapping.

1.2 Limitations of Feedback Linearization

Although feedback linearization is a popular approach in nonlinear control theory, it is good to point out some limitations. One of the main problems of feedback linearization is the coordinate transformation, which makes the design of a controller hard. Oftentimes tuning of a controller is achieved by comparing step responses, in simulation or experiment. In the author's experience it is particularly hard to design a controller for a system in which the states do not correspond to physical quantities. The coordinate transformation hides the meaning of the true dynamics.

Example 1.1 To illustrate the potential problems with feedback linearization, consider the following simple system:

$$\dot{x} = -x + (1 + ax^2)u \tag{1.1}$$

where the constant a has a nominal value $\bar{a}$, but is only known within some degree of accuracy: $a = \bar{a} + \delta a$. A proportional feedback linearizing controller is

$$u = \frac{x + k_1 x}{1 + \bar{a}x^2} \tag{1.2}$$

whereas a controller based on the Jacobian linearization with the same gain is

$$u = k_1 x. \tag{1.3}$$

Figure 1.1 shows the response from an initial error to zero for both controllers, where $k_1 = 5$, $\bar{a} = 10$. The controller based on the Jacobian linearization regulated the state to zero faster. The reason is clear: the nonlinearity in the system (1.1) is actually helping us to drive the system to zero, and the feedback linearizing controller cancels this beneficial term.

In some cases the Jacobi-linearized controller is optimal with respect to some cost criterion, and the feedback linearized controller is suboptimal. This is illustrated in the following example.

Example 1.2 (provided by J.C. Doyle) Consider the system

$$\dot{x} = e^x u \tag{1.4}$$

with cost criterion

$$J = \int_0^\infty x^2 + u^2 dt. \tag{1.5}$$

The linear controller with unity gain is

$$u_{lin} = -x, \tag{1.6}$$

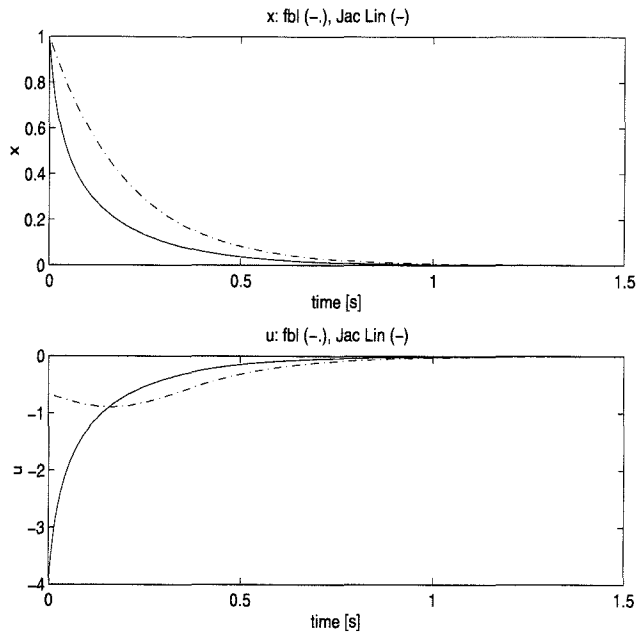


Figure 1.1 Feedback linearizing and Jacobian linearization controllers.

and the feedback linearizing controller with the same linearization at the origin is

$$u_{fb} = -xe^{-x}. \tag{1.7}$$

The optimal controller for this simple example can be found by solving the Bellman equation [?],

$$V_t + \min_u (V_x e^x u + x^2 + u^2) = 0, \tag{1.8}$$

giving

$$u_{opt} = -\frac{1}{2} V_x e^x \tag{1.9}$$

and upon substitution in Equation (1.8):

$$\begin{aligned} -\frac{1}{4} V_x^2 e^{2x} + x^2 &= 0 \\ V_x &= 2xe^{-x} \\ u_{opt} &= -x = u_{lin}. \end{aligned} \tag{1.10}$$

Hence the linear controller is optimal with respect to cost criterion J, whereas the feedback linearizing controller is not. The cost for the linear and feedback linearizing

controller as a function of the initial condition x is

$$\begin{aligned} J_{lin}(x) &= 2 - 2(1+x)e^{-x} \\ J_{fl}(x) &= \frac{1}{4}(1 - e^{-2x}(1+2x) + 2x^2) \end{aligned} \tag{1.11}$$

respectively. For $x \to +\infty$, J_{fl}/J_{lin} grows quadratically. For $x \to -\infty$, J_{fl}/J_{lin} grows exponentially. This shows that the feedback linearizing controller is arbitrarily worse than the optimal linearizing controller.

In most cases we cannot solve the Bellman equation analytically, and this example was constructed to make the equation solvable. Other examples show that the optimality of the linear controller is not restricted to systems with strong nonlinearities in the factor multiplying the input.

In [?] it is shown that in certain initial configurations a linear controller outperforms the feedback linearizing controller for trajectory tracking for the kinematic car. This is related to the fact that feedback linearization tries to decouple a multi-input system into separate chains of integrators. The coupling between inputs and outputs can be beneficial in certain cases. This in turn is related to the fact that it is easier to design controllers in physical coordinates.

1.3 Optimal Control

A solution to trajectory tracking that does not compute an explicit state space trajectory is given by optimal control [?, ?, ?]. Optimal control allows the minimization of an integral cost criterion subject to constraints on the initial and final states. In particular, optimal control encompasses the problem of steering from an initial to a final state while minimizing the error between the output and a desired trajectory for the output. It also encompasses the problem of minimizing an arbitrary cost function of the states and inputs subject to initial and final constraints, and the formulation of the problem of steering from an initial state to a final state in minimum time. Although the formulation of the solution to optimal control problems is quite straightforward and extremely elegant, the practical solution is complicated. It involves the numerical solution of two point boundary value problems, which is typically done by iteration and cannot guarantee convergence. For one time missions like spacecraft trajectories, the generality of optimal control outweighs the computation penalty. Indeed, optimal control has been used for many years to compute orbits for spacecraft. Computation time was not a big concern, since the mission was one of its kind and planned a long time ahead. In this thesis we are concerned with situations where computation time is at a premium, and convergence guarantees are indispensable. In these situations we have to perform the trajectory calculation many times in one single mission, and the desired trajectory changes continuously. See Section 1.5 for examples of applications where these conditions are important.

1.4 Two Degree of Freedom Design

As mentioned, two degree of freedom design is a paradigm where we generate a nominal trajectory for the state and input space around which we try to stabilize the system by means of linear controller. This is depicted in Figure 1.2. The source of the output trajectory is dependent on the application. It can be a human pilot or a machine based scheduler in a highly automated aircraft, a driver in a vehicle, or a high level scheduler for a robotic manufacturing plant. The module that generated the desired output trajectory is a level in itself, so in some sense the situation depicted in Figure 1.2 is really a three degree of freedom controller. The topmost level generates the output trajectory, which might not be feasible, or just consist of a series of way points. The intermediate level generates a feasible full state space and input trajectory, and the low level stabilizes the system around the nominal trajectory. Since the scheme is traditionally known as two degree of freedom design, we will follow that convention.

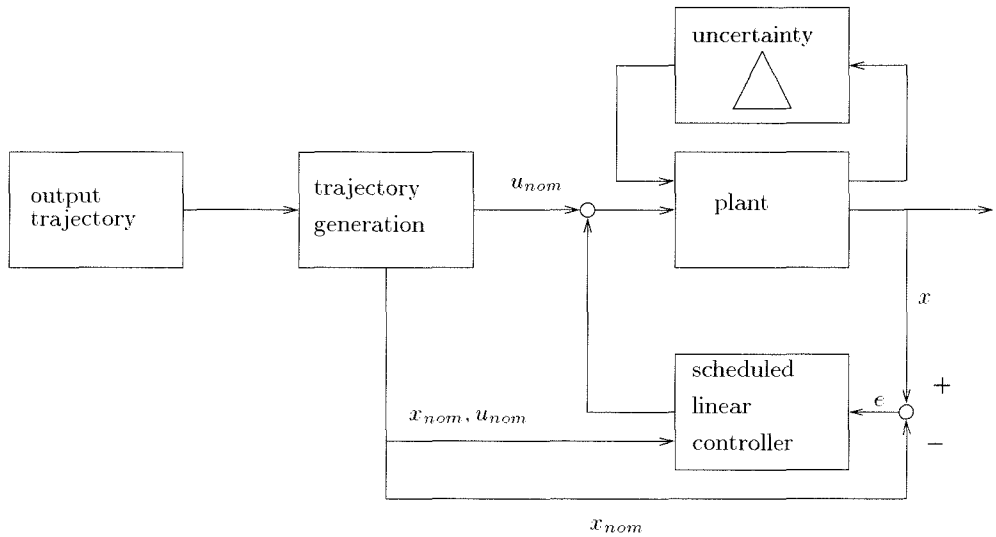


Figure 1.2 Two degree of freedom control.

The trajectory generation module generates a nominal state space trajectory and a nominal control input. This part of the controller can be run at a rate lower than the sampling rate, since the dynamics of the operator are typically much slower than those of the plant. The plant is linearized around the nominal trajectory, and a linear controller is used to stabilize the plant around this trajectory and deal with uncertainty. The advantage of linearizing the plant around a trajectory as opposed to using a coordinate transformation is that in the latter case it is often impossible to get a good uncertainty description that makes physical sense. The linear controller runs at a higher rate, since it needs to stabilize the plant dynamics. Note that the linear controller needs to have information about the nominal state to compute the appropriate linearization.

Two degree of freedom design allows an explicit robustness analysis using successive linearizations. Some promising work on the robustness analysis of nonlinear systems with uncertainty along a trajectory is reported in [?, ?, ?].

Typically the linear controller is gain scheduled, i.e., in different operating regimes we use different controllers. Various scheduling schemes can be devised. Suppose the set of controllers is $\{K_i\}$, and the scheduling variables are θ.

1. Hard switching: use controller K_i if $\theta \in \Theta_i$. This might result in chattering around the switching boundary.

2. Linear interpolation: design controller K_i for scheduling variables θ_i and use $K = \lambda_i K_i$ if $\theta = \lambda_i \theta_i$. This has the problem that the controller is never the controller designed for an operating point, but always some linear combination with neighboring controllers. This might hinder analysis.

3. Switching with hysteresis: use controller K_i if $d(\theta, \Theta_i) < \epsilon_i$, keep using this controller until $d(\theta, \Theta_i) > \delta_i$ where $\delta_i > \epsilon_i$. This avoids the chattering around the switching boundary associated with hard switching.

Even though gain scheduled controllers are widely and successfully used in practice, the resulting closed loop system is nonlinear and typically time varying, and there is no guarantee that the resulting system is stable. In particular, stability of the scheduled system over its entire operating range does not follow from stability in each separate regime. Only if the system is slowly time varying, i.e. the open loop system dynamics are much slower than the controller dynamics, we can establish stability. Even then the conditions are hard to compute [?, ?, ?].

Linear parameter varying (LPV) [?, ?] controllers provide global stability for plants with gain scheduled controllers. Usually, LPV control is used for measurable parameters that vary in some bounded interval. To apply LPV to two DOF design, we would have to schedule with respect to the nominal state and input, which would then have to be bounded for all time. This requirement is counter intuitive especially for the state. We would also need to be able to express the error system as a linear fractional transformation on the nominal state. This can in general not be done. Adding a nominal feed forward trajectory to an LPV controller would break the stability guarantee since we no longer have a linear time invariant (LTI) system, with a linear fractional parameter dependence.

An approach to deal with set point tracking for non-minimum phase systems is presented in [?]. This approach inverts the non-minimum phase zeros of the linearization to the left hand plane, and shows that this achieves better set-point tracking. The paper restricts attention to single input systems that are completely maximum phase, i.e. have all the poles in the right half plane. The authors suggest that their approach is more general though, and we certainly look forward to seeing the extensions hinted at in the paper. In any case, this approach does not apply to trajectory tracking per se.

1.5 Motivating Applications

Aircraft control has long been a driving application for the theory of nonlinear control systems. Aircraft have been flying successfully for many years without the help of sophisticated control systems, and the question arises if aircraft control really benefits from such sophisticated schemes as two degree of freedom control. The applications we have in mind exhibit a much higher level of vehicle autonomy than is currently present in aircraft control systems. The system might be a remotely piloted vehicle, with a low bandwidth communication link. The remote pilot can only give terse information at low update rates, and the on board computer has to generate feasible trajectories from the pilot information. Another application is an evasion-pursuit scenario, where the desired trajectory is the path of a vehicle to be intercepted. The desired output trajectory is obtained by on-board sensors and fed on-line to the trajectory generation module. Yet another application is station keeping, where a vehicle is supposed to monitor a target that might move, or maybe a target that is stationary, whereas the vehicle is subject to disturbances. More concretely, the California PATH project features a high degree of autonomy in cars, that are expected to follow other cars, or avoid them. Here too, the desired output trajectory comes from on-board sensors, and might not be feasible. The trajectory generation module functions as an intermediate layer that takes this coarse output trajectory and feeds a feasible nominal path to the lower level controllers. Finally, automated inspection by aerial vehicles is another application of highly autonomous systems. The desired inspection path may be stored as way points or be updated during the mission, depending on the type of information obtained. The trajectory generation module has to convert this sparse destination data into a feasible trajectory.

In all these cases, the variety of maneuvers is too rich to be stored on board in reasonably sized memory banks. The update rate of the desired output trajectory is slow enough to allow some on board computation, but not low enough to allow off line computation by heavy computational machinery. It is this class of applications for which trajectory generation has great potential.

Current aircraft control systems still leave a great authority to a human pilot. It will take time for more advanced control schemes, as suggested in this dissertation, to find acceptance in current flight control architectures. The most promising short term applications are in remotely piloted and fully autonomous vehicles.

1.6 Summary of Main Contributions

This dissertation is concerned with the problem of trajectory generation for nonlinear systems. A central theme in the thesis is the class of *differentially flat systems*. Differentially flat systems exhibit a one-to-one correspondence between trajectories in output space and trajectories in the full state and input space. From the desired trajectory in output space we can then algebraically generate the state space trajectory, making the trajectory generation problem trivial.

Differential flatness was originally formulated in the language of differential al-

gebra. Most of the theory and tools in nonlinear control theory use the language of differential geometry. It is therefore useful to give a definition of differential flatness in terms of differential geometry, to allow connections with the important results in nonlinear control theory. The main theoretical contribution is a foundation for differential flatness in the language of exterior differential systems. Using the tools of exterior differential systems the thesis proves some results connecting flatness to feedback linearizability. In particular, it is shown that differential flatness is equivalent to feedback linearizability in an open and dense set. The thesis gives a complete characterization of differential flatness in the single input case, allowing time varying flat outputs.

To follow up on the promise to provide computational tools, a number of important trajectory generation problems are formulated, and a software library is presented that solves these problems. All simulations presented in the thesis used this library and the real time experiments used the same library compiled on a PC. The software is analyzed on computation time.

Another theoretical contribution is the formulation of the real time trajectory generation problem and two algorithms that solve it. Again, software implementing these algorithms is provided and analyzed.

A number of methods are given that extend the trajectory generation methods for flat systems to systems that are not flat. These methods start with a flat approximation to the system and look at the remaining terms as perturbations. These methods are analyzed on their theoretical properties and validated in simulation and experiment.

Some systems exhibit a natural division in two subsystems: an outer system that is flat with respect to some pseudo inputs, and an inner system that is not flat. If we can control the inner system tight enough, we can treat the pseudo inputs to the outer system as inputs. Flatness of the outer system allows full state trajectory generation. We present two theorems on the conditions required to achieve exponential and bounded tracking for the total system based on exponential and bounded tracking of the inner and outer system.

Experimental validation takes an important place in this thesis. Both algorithms and software are evaluated on real time experiments available at Caltech. One is the ducted fan, which is a model of the pitch dynamics of a thrust vectored aircraft. The other is an electric model helicopter.

1.7 Overview of the Dissertation

The theoretical foundation for differential flatness in terms of exterior differential systems is given in Chapter 2. This chapter also proves some properties of differentially flat systems in the geometric framework and gives examples of flat systems. Chapter 3 presents some important trajectory generation problems and algorithms to solve them for differentially flat systems. It also presents the software library that implements these algorithms. It illustrates these through simulations and experiments. Chapter 4 presents the real time trajectory generation problem, and two algorithms to solve it for differentially flat systems. Again, simulations and

experiments validate the algorithms. Chapter 5 presents some extensions to deal with perturbations to flatness, and validates these in experiment and simulation. In Chapter 6 we define outer flatness, discuss some theoretical properties and present the helicopter experiment as a test case for outer flatness. In Chapter 7 we summarize the main points and point out directions for future research.

Chapter 2

Differential Flatness

2.1 Introduction

This chapter will give the mathematical definition of the class of *differentially flat* systems in terms of exterior differential systems. Much of this material is very technical, and the hurried reader can get a good understanding of flatness by looking at Equation (2.2), which captures the essence of flatness. In Section 2.7 we present examples of flat systems. The results in this chapter were joined work with Muruhan Rathinam, and appeared earlier in an abbreviated form as [?].

2.2 Historical Context

The problem of equivalence of nonlinear systems (in particular to linear systems, that is, feedback linearization) is traditionally approached in the context of differential geometry [?, ?]. A complete characterization of static feedback linearizability in the multi-input case is available, and for single input systems it has been shown that static and dynamic feedback linearizability are equivalent [?]. Some special results have been obtained for dynamic feedback linearizability of multi-input systems, but the general problem remains unsolved. Typically, the conditions for feedback linearizability are expressed in terms of the involutivity of distributions on a manifold.

More recently it has been shown that the conditions on distributions have a natural interpretation in terms of exterior differential systems [?, ?]. In exterior differential systems, a control system is viewed as a Pfaffian module. Some of the advantages of this approach are the wealth of tools available and the fact that implicit equations and non-affine systems can be treated in a unified framework. For an extensive treatment of exterior differential systems we refer to [?].

Fliess and coworkers [?, ?, ?] studied the feedback linearization problem in the context of differential algebra and introduced the concept of *differential flatness*. In differential algebra, a system is viewed as a differential field generated by a set of variables (states and inputs). The system is said to be differentially flat if one can find a set of variables, called the flat outputs, such that the system is (non-differentially) algebraic over the differential field generated by the set of flat outputs. Roughly speaking, a system is flat if we can find a set of outputs (equal in number

to the number of inputs) such that all states and inputs can be determined from these outputs without integration. More precisely, if the system has states $x \in \mathbb{R}^n$, and inputs $u \in \mathbb{R}^m$ then the system is flat if we can find outputs $y \in \mathbb{R}^m$ of the form

$$y = y(x, u, \dot{u}, \ldots, u^{(l)}) \tag{2.1}$$

such that

$$\begin{aligned} x &= x(y, \dot{y}, \ldots, y^{(q)}) \\ u &= u(y, \dot{y}, \ldots, y^{(q)}). \end{aligned} \tag{2.2}$$

Differentially flat systems are useful in situations where explicit trajectory generation is required. Since the behaviour of flat systems is determined by the flat outputs, we can plan trajectories in output space, and then map these to appropriate inputs. A common example is the kinematic car with trailers, where the xy position of the last trailer provides flat outputs [?]. This implies that all feasible trajectories of the system can be determined by specifying only the trajectory of the last trailer. Unlike other approaches in the literature (such as converting the kinematics into a normal form), this technique works globally.

A limitation of the differential algebraic setting is that it does not provide tools for regularity analysis. The results are given in terms of meromorphic functions in the variables and their derivatives, without characterizing the solutions. In particular, solutions to the differential polynomials may not exist. For example, the system:

$$\begin{aligned} \dot{x}_1 &= u \\ \dot{x}_2 &= x_1^2 \end{aligned} \tag{2.3}$$

is flat in the differentially algebraic sense with flat output $y = x_2$. However, it is clear that the derivative of x_2 always has to be positive, and therefore we cannot follow an arbitrary trajectory in y space.

To treat time as a special variable in the relations (2.2), one can to resort to Lie-Bäcklund transformations on infinite dimensional spaces [?, ?]. The latter paper distinguishes between "orbital (or topological) flatness" where time scalings are allowed, and "differential flatness" where they are not.

In the beginning of this century, the French geometer E. Cartan developed a set of powerful tools for the study of equivalence of systems of differential equations [?, ?, ?]. Equivalence need not be restricted to systems of equal dimensions. In particular a system can be *prolonged* to a bigger system on a bigger manifold, and equivalence between these prolongations can be studied. This is the concept of *absolute equivalence* of systems. Prolonging a system corresponds to dynamic feedback, and it is clear that we can benefit from the tools developed by Cartan to study the feedback linearization problem. The connections between Cartan prolongations and feedback linearizability for single input systems were studied in [?].

In this chapter we reinterpret flatness in a differential geometric setting. We

make extensive use of the tools offered by exterior differential systems, and the ideas of Cartan. This approach allows us to study some of the regularity issues, and also to give an explicit treatment of time dependence. Moreover, we can easily make connections to the extensive body of theory that exists in differential geometry. We show how to recover the differentially algebraic definition, and give an exterior differential systems proof for a result proven by Martin [?, ?] using differential algebra: a flat system can be put into Brunovsky normal form by dynamic feedback in an open and dense set (this set need not contain an equilibrium point).

We also give a complete characterization of flatness for systems with a single input. In this case, flatness in the neighborhood of an equilibrium point is equivalent to linearizability by static state feedback around that point. This result is stronger than linearizability by endogenous feedback as indicated by Martin et al. [?, ?], since the latter only holds in an open and dense set. We also treat the case of time varying versus time invariant flat outputs, and show that in the case of a single input, time invariant system the flat output can always be chosen time independent. In exterior differential systems, the special role of the time coordinate is expressed as an independence condition, i.e., a one-form that is not allowed to vanish on any of the solution curves. A fundamental problem with exterior differential systems is that most results only hold on open dense sets [?]. It requires extra effort to obtain results in the neighborhood of a point, see for example [?]. In this chapter too, we can only get local results by introducing regularity assumptions, typically in the form of rank conditions.

The organization of the chapter is as follows. In Section 2.3 we introduce the definitions pertaining to absolute equivalence and their interpretation in control theory. In Section 2.4 we introduce our definition of differential flatness and show how to recover the differential algebraic results. In Section 2.5 we study the connections between flatness and feedback linearizability. In Section 2.6 we present our main theorems characterizing flatness for single input systems, and in Section 2.7 we give examples of differentially flat systems. In Section ?? we summarize our results and point out some open questions.

2.3 Prolongations and Control Theory

This section introduces the concept of prolongations, and states some basic theorems. It relates these concepts to control theory. Proofs of most of these results can be found in [?]. We assume that all manifolds and mappings are smooth (C^∞) unless explicitly stated otherwise.

Definition 2.1 (Pfaffian system) A *Pfaffian system* I on a manifold M is a submodule of the module of differential one-forms $\Omega^1(M)$ over the commutative ring of smooth functions $C^\infty(M)$. A set of one-forms $\omega^1, \ldots, \omega^n$, generates a Pfaffian system $I = \{\omega^1, \ldots, \omega^n\} = \{\Sigma f_k \omega^k | f_k \in C^\infty(M)\}$.

In this work, we restrict attention to finitely generated Pfaffian systems on finite dimensional manifolds.

It is important to distinguish between a Pfaffian system and its set of generators or the algebraic ideal $\mathcal{I}$ in $\Lambda(M)$ generated by I. Since we are only dealing with Pfaffian systems the term *system* will henceforth mean a Pfaffian system.

For a Pfaffian system I we can define its *derived system* $I^{(1)}$ as $I^{(1)} = \{\omega \in I | d\omega \equiv 0 \bmod \mathcal{I}\}$, where $\mathcal{I}$ is the algebraic ideal generated by I. The derived system is itself a Pfaffian system, so we can define the sequence $I, I^{(1)}, I^{(2)}, \ldots$ which is called the *derived flag* of I. The algebraic ideal will contain forms of degree $1 \ldots \dim(M)$. The degree k part of I is the set of all k-forms in $\mathcal{I}$.

Assumption 2.2 (Regularity of Pfaffian systems) Unless explicitly otherwise stated, we will assume throughout this work that the system is *regular*, i.e.

1. The system and all its derived systems have constant rank.
2. For each k, the exterior differential system generated by $I^{(k)}$ has a degree 2 part with constant rank.

If the system is regular the derived flag is decreasing, so there will be an N such that $I^{(N)} = I^{(N+1)}$. This $I^{(N)}$ is called the *bottom derived system.*

When one studies the system of one-forms corresponding to a system of differential equations, the independent variable time becomes just another coordinate on the manifold along with the dependent variables. Hence the notion of an independent variable is lost. If x denotes the dependent variables, a solution to such a system $c : s \to (t(s), x(s))$ is a curve on the manifold. But we are only interested in solution curves which correspond to graphs of functions $x(t)$. Hence we need to reject solutions for which $\frac{dt}{ds}$ vanishes at some point. This is done by introducing dt as an *independence condition*, i.e., a one-form that is not allowed to vanish on any of the solution curves. An independence condition is well defined only up to a nonvanishing multiple and modulo I. We will write a system with independence condition τ as (I, τ). The form τ is usually exact, but it does not have to be. In this work we shall always take τ exact, in agreement with its physical interpretation as time.

Definition 2.3 (Control System) A Pfaffian system with independence condition (I, dt) is called a *control system* if $\{I, dt\}$ is integrable.

In local coordinates, control systems can be written in the form:

$$I = \{dx_1 - f_1(x, u, t)dt, \ldots, dx_n - f_n(x, u, t)dt\} \tag{2.4}$$

with states $\{x_1, \ldots, x_n\}$ and inputs $\{u_1, \ldots, u_p\}$. Note that a control system is always assumed to have independence condition dt. If the functions f are independent of time then we speak of a *time invariant* control system.

Definition 2.4 (Cartan Prolongation) Let (I, dt) be a Pfaffian system on a manifold M. Let B be a manifold such that $\pi : B \to M$ is a fiber bundle. A Pfaffian system $(J, \pi^* dt)$ on B is a *Cartan prolongation* of the system (I, dt) if the following conditions hold:

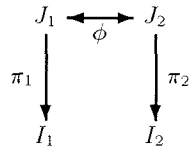
1. $\pi^*(I) \subset J$
2. For every integral curve of I, $c : (-\epsilon, \epsilon) \to M$, there is a unique lifted integral curve of J, $\tilde{c} : (-\epsilon, \epsilon) \to B$ with $\pi \circ \tilde{c} = c$.

Assumption 2.5 (Regularity of Cartan prolongations) In this work we only look at Cartan prolongations that preserve codimension.

Note that all prolongations are required to preserve the independence condition of the original system. The above definition implies that there is a smooth 1-1 correspondence between the integral curves of a system and of its Cartan prolongation. Cartan prolongations are useful to study equivalence between systems of differential equations that are defined on manifolds of different dimensions. This occurs in dynamic feedback extensions of control systems. We increase the dimension of the state by adding dynamic feedback, but the extended system is still in some sense equivalent to the original system.

This allows us to define the concept of absolute equivalence introduced by Elie Cartan [?]:

Definition 2.6 (Absolute Equivalence) Two systems I_1, I_2 are called *absolutely equivalent* if they have Cartan prolongations J_1, J_2 respectively that are equivalent in the usual sense, i.e., there exists a diffeomorphism ϕ such that $\phi^*(J_2) = J_1$. This is illustrated in the following diagram:



An interesting subclass of Cartan prolongations is formed by *prolongations by differentiation*: If (I, dt) is a system with independence condition on M, and du an exact one-form on M that is independent of $\{I, dt\}$, and if y is a fiber coordinate of B. Then $\{I, du - ydt\}$ is called a *prolongation by differentiation* of I. Note that we have omitted writing $\pi^*(du - ydt)$ where $\pi : B \to M$ is the surjective submersion. We will make this abuse in the rest of the work for notational convenience. Prolongations by differentiation correspond to adding integrators to a system. In the context of control systems, the coordinate u is the input that is differentiated.

If we add integrators to all controls, we obtain a *total prolongation*: Let (I, dt) be a system with independence condition, where $\dim I = n$. Let $\dim M = n + p + 1$. Let $u_1, \ldots, u_p$ be coordinates such that $du_1, \ldots, du_p$ are independent of $\{I, dt\}$, and let $y_1, \ldots, y_p$ be fiber coordinates of B, then $\{I, du_1 - y_1 dt, \ldots, du_p - y_p dt\}$ is called a *total prolongation* of I. Total prolongations can be defined independent of coordinates, and are therefore intrinsic geometric objects. It can be shown that in codimension 2 (i.e., a system with n generators on an $n+2$ dimensional manifold), all Cartan prolongations are locally equivalent to total prolongations [?].

We will call *dynamic feedback* a feedback of the form

$$\begin{aligned}\dot{z} &= a(x, z, v, t)\\ u &= b(x, z, v, t).\end{aligned}$$

If t does not appear in (a, b) we call (a, b) a *time invariant* dynamic feedback. The dynamic feedback is called *regular dynamic feedback* if for each fixed x and t the map $b(x, ., ., t) : (z, v) \mapsto u$ is a submersion. An important question is what type of dynamic feedback corresponds to what type of prolongation. Clearly, prolongations by differentiation correspond to dynamic extension [?] (adding integrators to the inputs) .

Cartan prolongations provide an intrinsic geometric way to study dynamic feedbacks. We shall show that Cartan prolongations that extend a control system to another control system can be expressed as dynamic feedbacks in local coordinates. The following example shows that not every dynamic feedback corresponds to a Cartan prolongation:

Example 2.7 (Dynamic Feedback vs. Cartan prolongation) Consider the control system

$$\dot{x}_1 = u,$$

with feedback

$$\begin{aligned}\dot{z}_1 &= z_2\\ \dot{z}_2 &= -z_1\\ u &= g(z)v.\end{aligned}$$

This dynamic feedback introduces harmonic components which can be used to asymptotically stabilize nonholonomic systems (see [?] for a description of how this might be done). It is not a Cartan prolongation since (z, v) cannot be uniquely determined from (x, u).

It must be said that the feedback in Example 2.7 is somewhat unusual, in that most theorems concerning dynamic feedback are restricted to adding some type of integrator to the inputs of the system.

Definition 2.8 (Endogenous Feedback) Let $\dot{x} = f(x, u, t)$ be a control system. A dynamic feedback

$$\begin{aligned}\dot{z} &= a(x, z, v, t)\\ u &= b(x, z, v, t)\end{aligned} \tag{2.5}$$

is said to be *endogenous* if z and v satisfying (2.5) can be expressed as functions of x, u, t and a finite number of their derivatives:

$$\begin{aligned} z &= \alpha(x, u, \ldots, u^{(l)}, t) \\ v &= \beta(x, u, \ldots, u^{(l)}, t). \end{aligned} \tag{2.6}$$

An endogenous feedback is called *regular* if for each fixed x and t the map $b(x, ., ., t) : (z, v) \mapsto u$ is a submersion.

Note that this differs slightly from the definition given in [?, ?] due to the explicit time dependence used here. The relationship between Cartan prolongations and endogenous dynamic feedback is given by the following two theorems. The first says that a regular endogenous feedback corresponds to a Cartan prolongation.

Theorem 2.9 (Endogenous feedbacks are Cartan prolongations) *Let I be a control system on an open set $T \times X \times U$ which in coordinates (t, x, u) is given by $\dot{x} = f(x, u, t)$. Let J denote the control system on the open set $T \times X \times Z \times V$ which is obtained from the above system by adding a regular endogenous dynamic feedback. Then J is a Cartan prolongation of I.*

Proof: Define the mapping $F : T \times X \times Z \times V \to T \times X \times U$ by $F(t, x, z, v) = (t, x, b(x, z, v, t))$. Since b is regular, F is a submersion. Furthermore b is surjective since the feedback is endogenous. Therefore F is surjective too. Since F is a surjective submersion, $T \times X \times Z \times V$ is fibered over $T \times X \times U$. Hence we have that solutions $(t, x(t), z(t), v(t))$ of J project down to solutions $(t, x(t), b(x(t), z(t), v(t), t))$ of I. Therefore the first requirement of being a Cartan prolongation is satisfied. The second requirement of unique lifting is trivially satisfied by the fact that z and v are obtained uniquely by equation (2.6). ■

Conversely, a Cartan prolongation can be realized by endogenous dynamic feedback in an open and dense set, if the resulting prolongation is a control system:

Theorem 2.10 (Cartan prolongations are locally endogenous feedbacks) *Let I be a control system on a manifold M with p inputs, $\{u_1, \ldots, u_p\}$. Every Cartan prolongation $J = \{I, \omega_1, \ldots, \omega_r\}$ on B with independence condition dt such that J is again a control system is realizable by endogenous regular feedback on an open and dense set of B.*

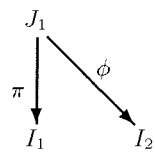
Proof: Let r denote the fiber dimension of B over M, and let $\{w_1, \ldots, w_r\}$ denote the fiber coordinates. Since I is a control system, $\{I, dt\}$ is integrable, and we can find n first integrals $x_1, \ldots, x_n$. Preservation of the codimension and integrability of $\{J, dt\}$ means that we can find r extra functions $a_1, \ldots, a_r$ such that $J = \{I, dz_1 - a_1 dt, \ldots, dz_r - a_r dt\}$. Here the z_i are first integrals of $\{J, dt\}$ that are not first integrals of $\{I, dt\}$. Pick p coordinates $v(u, w)$ such that $\{t, x, z, v\}$ form a set of coordinates of B. The v coordinates are the new control inputs. Clearly $a_i = a_i(x, z, v, t)$ since we have no other coordinates. Also since $\{t, x, z, v\}$ form

coordinates for B, and u is defined on B, there has to be a function b such that $u = b(x, z, v, t)$. Since both (t, x, u, w) and (t, x, z, v) form coordinates on B, there has to be a diffeomorphism ϕ between the 2. From the form of the matrix $\frac{\partial \phi}{\partial(t,x,z,v)}$ it can be seen that $\frac{\partial b}{\partial(z,v)}$ is full rank, and hence b is regular. This recovers the form of equation (2.5). Since J is a Cartan prolongation, every (x, u, t) lifts to a unique (x, z, v, t). From Lemma 2.15 , to be presented in the next section, it then follows that we can express (z, v) as functions of x and u and its derivatives in an open and dense set. We thus obtain the form of equation (2.6). ■

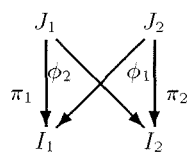
2.4 Differentially Flat Systems

In this section we present a definition of flatness in terms of prolongations. Our goal is to establish a definition of flatness in terms of differential geometry, while capturing the essential features of flatness in differential algebra [?, ?]. We build our definition on the minimal requirements needed to recover these features, namely the one to one correspondence between solution curves of the original system and an unconstrained system, while maintaining regularity of the various mappings. Our definition makes use of the concept of an absolute morphism [?].

Definition 2.11 (Absolute morphism) An *absolute morphism* from a system (I_1, dt) on M_1 to a system (I_2, dt) on M_2 consists of a Cartan prolongation (J_1, dt) on $\pi : B_1 \to M_1$ together with surjective submersion $\phi : B_1 \to M_2$ such that $\phi^*(I_2) \subset J_1$. This is illustrated below:



Definition 2.12 (Invertibly absolutely morphic systems) Two systems (I_1, dt) and (I_2, dt) are said to be *absolutely morphic* if there exist absolute morphisms from (I_1, dt) to (I_2, dt) and from (I_2, dt) to (I_1, dt). This is illustrated below:



Two systems (I_1, dt) and (I_2, dt) are said to be *invertibly absolutely morphic* if they are absolutely morphic and the following inversion property holds: let $c_1(t)$ be an integral curve of I_1 with $\tilde{c}_1$ the (unique) integral curve of J_1 such that $c_1 = \pi \circ \tilde{c}_1$, and let $\gamma(t) = \phi_2 \circ \tilde{c}_1(t)$ be the projection of $\tilde{c}_1$. Then we require that $c_1(t) = \phi_1 \circ \tilde{\gamma}(t)$, where $\tilde{\gamma}(t)$ is the lift of γ from I_2 to J_2. The same property must hold for solution curves of I_2.

If two systems are invertibly absolutely morphic, then the integral curves of one system map to the integral curves of the other and this process is invertible in the sense described above. If two systems are absolutely equivalent then they are also absolutely morphic, since they can both be prolonged to systems of the same dimension which are diffeomorphic to each other. However, for two systems to be absolutely morphic we do not require that any of the systems have the same dimension.

A differentially flat system is one in which the "flat outputs" completely specify the integral curves of the system. More precisely:

Definition 2.13 (Differential Flatness) A system (I, dt) is *differentially flat* if it is invertibly absolutely morphic to the trivial system $I_t = (\{0\}, dt)$.

Notice that we require that the independence condition be preserved by the absolute morphisms, and hence our notion of time is the same for both systems. Since an independence condition is only well defined up to nonvanishing multiples and modulo the system, we do allow time scalings between the systems We also allow time to enter into the absolute morphisms which map one system onto the other.

If the system (I, dt) is defined on a manifold M, then we can restrict the system to a neighborhood around a point in M, which is again itself a manifold. We will call a system flat in that neighborhood if the restricted system is flat.

The following discussion leans heavily on a theorem due to Sluis and Shadwick,[?, ?], which we recall here for completeness:

Theorem 2.14 *Let I be a system on a manifold M and J a Cartan prolongation of I on $\pi : B \to M$. On an open and dense subset of B, there exists a prolongation by differentiation of J that is also a prolongation by differentiation of I.*

Proof: See [?], Theorem 24. ■

In order to establish the relationship between our definition and the differential algebraic notion of flatness, we need the following straightforward corollary to Theorem 2.14. This lemma expresses the dependence of the fiber coordinates of a Cartan prolongation on the coordinates of the base space:

Lemma 2.15 *Let (I, dt) be a system on a manifold M with local coordinates $(t, x) \in \mathbb{R}^1 \times \mathbb{R}^n$ and let (J, dt) be a Cartan prolongation on the manifold B with fiber coordinates $y \in \mathbb{R}^r$. Assume the regularity assumptions 2.2, 2.5 hold. Then on an open dense set, each y_i can be uniquely determined from t, x and a finite number of derivatives of x.*

Proof: By Theorem 2.14 there is a prolongation by differentiation, on an open and dense set, say I_2, of J, with fiber coordinates z_i, that is also a prolongation by differentiation of the original system I, say with fiber coordinates w_i. This means that the (x, y, z, t) are diffeomorphic to (x, w, t): $y = y(x, w, t)$. The w are derivatives of x, and therefore the claim is proven. ■

This lemma allows us to explicitly characterize differentially flat systems in a local coordinate chart. Let a system in local coordinates (t, x) be differentially flat and let the corresponding trivial system have local coordinates (t, y). Then on an open and dense set there are surjective submersions h and g with the following property: Given any curve $y(t)$, then

$$x(t) = g(t, y(t), \ldots, y^{(q)}(t))$$

is a solution of the original system and furthermore the curve $y(t)$ can be obtained from $x(t)$ by

$$y(t) = h(t, x(t), \ldots, x^{(l)}(t)).$$

This follows from using definitions of absolute morphisms, the invertibility property, and Lemma 2.15, stating that fiber coordinates are functions of base coordinates and their derivatives and the independent coordinate.

This local characterization of differential flatness corresponds to the differential algebraic definition except that h and g need not be algebraic or meromorphic. Also, we do not require the system equations to be algebraic or meromorphic. The explicit time dependence corresponds to the differential algebraic setting where the differential ground field is a field of functions and not merely a field of constants. The functions g and h now being surjective submersions enables us to link the concept of flatness to geometric nonlinear control theory where we usually impose regularity. We emphasize that we only required a one to one correspondence of solution curves *a priori* for our definition of flatness, and not that this dependence was in the form of derivatives. The particular form of this dependence followed from our analysis.

Finally, the following theorem allows us to characterize the notion of flatness in terms of absolute equivalence.

Theorem 2.16 *Two systems are invertibly absolutely morphic if and only if they are absolutely equivalent.*

Proof: Sufficiency is trivial. We shall prove necessity. For convenience we shall not mention independence conditions, but they are assumed to be present and do not affect the proof. Let I_1 on M_1 and I_2 on M_2 be invertibly absolutely morphic. Let J_1 on B_1 be the prolongation of I_1 with $\pi_1 : B_1 \to M_1$ and similarly J_2 on B_2 be the prolongation of I_2 with $\pi_2 : B_2 \to M_2$. Let the absolute morphisms be $\phi_1 : B_2 \to M_1$ and $\phi_2 : B_1 \to M_2$.

We now argue that J_2 is a Cartan prolongation of I_1 (and hence I_1 and I_2 are absolutely equivalent). By assumption ϕ_1 is a surjective submersion and every solution $\tilde{c}_2$ of J_2 projects down to a solution c_1 of I_1 on M_1. The only extra requirement for J_2 on $\phi_1 : B_2 \to M_1$ to be a (Cartan) prolongation is that every solution c_1 of I_1 has a unique lift $\tilde{c}_2$ (on B_2) which is a solution of J_2.

To show existence of a lift, observe that for any given c_1 which is a solution of I_1, we can obtain its unique lift $\tilde{c}_1$ on B_1 (which solves J_1), and get its projection c_2 on M_2 (which solves I_2) and then consider its unique lift $\tilde{c}_2$ on B_2. Now it follows

from the invertibility property that $\phi_1 \circ \tilde{c}_2 = c_1$. In other words, $\tilde{c}_2$ projects down to c_1.

To see the uniqueness of this lift, suppose $\tilde{c}_2$ and $\tilde{c}_3$ which are solutions of J_2 on B_2, both project down to c_1 on M_1. Consider their projections c_2 and c_3 (respectively) on M_2. When we lift c_2 or c_3 to B_2 and project down to M_1 we get c_1. Which when lifted to B_1 gives, say $\tilde{c}_1$. By the requirement of the absolute morphisms being invertible $\tilde{c}_1$ should project down to (via ϕ_2) c_2 as well as c_3. Then uniqueness of projection implies that c_2 and c_3 are the same. Which implies $\tilde{c}_2$ and $\tilde{c}_3$ are the same.

Hence J_2 is a Cartan prolongation of I_1 as well. Hence I_1 and I_2 are absolutely equivalent. ■

Using this theorem we can completely characterize differential flatness in terms of absolute equivalence:

Corollary 2.17 *A system (I, dt) is differentially flat if and only if it is absolutely equivalent to the trivial system $I_t = (\{0\}, dt)$.*

Note that we require the feedback equivalence to preserve time, since both systems have the same independence condition. In the classical feedback equivalence we only consider diffeomorphisms of the form $(t, x, u) \mapsto (t, \phi(x), \psi(x, u))$. For flatness we allow diffeomorphisms of the form $(t, x, u) \mapsto (t, \phi(t, x, u), \psi(t, x, u))$. We could allow time scalings of the form $t \mapsto s(t)$ but this does not change the independence condition and does therefore not gain any generality. In Cartan's notion of equivalence all diffeomorphisms are completely general. This is akin to the notion of *orbital* flatness presented in [?], where one allows time scalings dependent on all states and inputs.

Often we will be interested in a more restricted form of flatness that eliminates the explicit appearance of time that appears in the general definition.

Definition 2.18 An absolute morphism from a time invariant control system (I_1, dt) to a time invariant control system (I_2, dt) is a *time-independent absolute morphism* if locally the maps $\pi : B_1 \to M_1$ and $\phi : B_1 \to M_2$ in Definition 2.11 have the form $(t, x, u) \mapsto (t, \eta(x, u), \psi(x, u))$, i.e. the mappings between states and inputs do not depend on time. A system (I, dt) is *time-independent differentially flat* if it is differentially flat using time-independent absolute morphisms.

Note that the example given above is time-independent differentially flat. One might be tempted to think that if the control system I is time invariant and knowing that the trivial system is time invariant, we can assume that the absolute morphism $x = \phi(t, y, y^{(1)}, \ldots, y^{(q)})$ has to be time independent as well. That this is not true is illustrated by the following example.

Example 2.19 Consider the system $\dot{y} = ay$, and the coordinate transformation $y = x^2 e^{t+x}$. Then $\dot{x} = \frac{(a-1)x}{2+x}$. Both systems are time invariant, but the coordinate transformation depends on time.

2.5 Linear Systems and Linearizability

The differential algebra approach to control has given rise to new interpretations of linearity [?, ?]. Rather than overloading the concept of linearity we feel it increases clarity if we stick with the conventional notion of linearity (see for example [?]) and introduce a new term for the broader concept of linearity as exposed in [?, ?]. We will try to clarify the different notions and indicate what the underlying approaches and assumptions are. This will enable us to elucidate the connections with flatness and prolongations. The following definitions are widely accepted and taken from [?].

Definition 2.20 A *dynamical system* is a 5-tuple (X, U, Y, T, ρ). Here X is the set of states, U is a set of allowable input functions and $U(T)$ denotes the possible values of the inputs at a fixed time. Y is the set of outputs functions and T is the set of times over which the system evolves. The map $\rho : (X, U, T, T) \to Y$, $\rho(x_0, u_{[t_0,t_1]}, t_0, t_1) = y_1$ is the response function that maps an initial state x_0 at an initial time t_0 given an input on an interval $[t_0, t_1]$, to an output y_1 at a final time t_1.

Definition 2.21 (Linear system) A dynamical system is said to be *linear* if

1. The sets X, U and Y are linear vector spaces over the same field.
2. For each fixed initial and final time (t_0, t_1) respectively, the response function $\rho(.,.,t_0,t_1)$ is a linear map from (X, U) into Y.

The linearity of the response function implies in particular that the origin of the space X is an equilibrium point.

Definition 2.22 (Time invariant system) Let S_τ denote the delay map from a function space onto itself: $(S_\tau f)(t) = f(t - \tau)$. A dynamical system is said to be *time invariant* if

1. The input, output, and time spaces are closed under operation of S_τ for all $\tau \in \mathbb{R}$.
2. $\rho(x_0, u, t_0, t_1) = \rho(x_0, S_\tau u, t_1 + \tau, t_0 + \tau)$.

In particular, a system of the form

$$\dot{x} = Ax + Bu \tag{2.7}$$

$$y = Cx + Du \tag{2.8}$$

is linear and time invariant. Here (A, B, C, D) are matrices of appropriate dimensions. If the system is controllable, we can find outputs z_i such that the system is equivalent (by a linear coordinate tranformation) to m chains of integrators, where m is the number of outputs:

$$z_i^{(l_i)} = u_i. \tag{2.9}$$

This form is called the *Brunovsky normal form*

Definition 2.23 (Feedback linearizability) The time invariant nonlinear system

$$\dot{x} = f(x, u) \tag{2.10}$$

is *feedback linearizable* if there is a dynamic feedback

$$\dot{z} = \alpha(x, z, v) \tag{2.11}$$
$$u = \beta(x, z, v) \tag{2.12}$$

and new coordinates $\xi = \phi(x, z)$ and $\eta = \psi(x, z, v)$ such that in the new coordinates the system has the form:

$$\dot{\xi} = A\xi + B\eta \tag{2.13}$$

and the mapping ϕ maps onto a neighborhood of the origin. If $\dim z = 0$ then we say the system is *static* feedback linearizable.

The form in equation (2.13) is the standard form in linear systems theory. It is useful if one wants to design controllers for nonlinear systems around equilibrium points.

It might be that the system can be put in the form (2.13) but that the coordinate transformation is not valid in a neighborhood of the origin of the target system. In that case we can shift the origin of the linear system to put it in the form

$$\dot{\xi} = A\xi + B\eta + E. \tag{2.14}$$

We will call this a *state space affine* form. This form is called linear in [?], but most results in linear systems theory cannot be applied since the origin is not an equilibrium point. However, it is still useful in the context of trajectory generation. For example, a nonholonomic system in chained form ([?]) can be transformed to this state space affine form.

It is clear that all feedback linearizable (by static or dynamic feedback) systems are flat, since we can put them into Brunovsky normal form. The converse only holds in an open and dense set, as is shown by the following theorem. An analogous result was proven by Martin in a differentially algebraic setting [?, ?].

Theorem 2.24 *Every differentially flat system can be put in Brunovsky normal form in an open and dense set through regular endogenous feedback.*

Proof: Let J, J_t be the Cartan prolongations of I, I_t respectively. Then by Theorem 2.14, on an open and dense set, there is a prolongation by differentiation of J_t that is also a prolongation by differentiation of I_t, say J_{t1}. Let J_1 be the corresponding Cartan prolongation of J. Then J_1 is equivalent to J_{t1}, which is in Brunovsky normal form. In particular, since J_1 is a Cartan prolongation, it can be realized by regular endogenous feedback. ■

This proof relies on Theorem 2.14 which restricts its validity to an open and dense set. We conjecture that the result in Theorem 2.24 holds everywhere, but the above proof technique does not allow us to conclude that. The obstruction lies in certain prolongations that we cannot prove to be regular.

We emphasize here that even though flatness implies that we can find coordinates that put the system into the linear form (2.13) we do not require the underlying manifolds to be linear spaces. In this sense, flatness is an intrinsic property of a control system defined on a smooth manifold.

2.6 Flatness for Single Input Systems

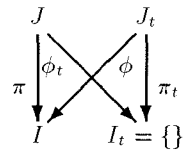
For single input control systems, the corresponding differential system has codimension 2. There are a number of results available in codimension 2 which allow us to give a complete characterization of differentially flat single input control systems. In codimension 2 every Cartan prolongation is a total prolongation around every point of the fibered manifold (see [?]), given our regularity assumptions 2.2, 2.5. This allows us to prove the following

Theorem 2.25 *Let I be a time invariant control system:*

$$I = \{dx_1 - f_1(x,u)dt, \ldots, dx_n - f_n(x,u)dt\},$$

where u is a scalar control, i.e., the system has codimension 2. If I is time-independent differentially flat around an equilibrium point, then I is feedback linearizable by static time invariant feedback at that equilibrium point.

Proof: Let I be defined on M with coordinates (x,u,t), let the trivial system I_t be defined on B_t with coordinates (y_0,t), let the prolongation of I_t be J_t, and let J_t be defined on M_t. This is illustrated below :



First we show that J_t can be taken as a Goursat normal form around the equilibrium point. In codimension 2, every Cartan prolongation is a repeated total prolongation in a neighborhood of every point of the fibered manifold ([?], Theorem 5). Let $I_{t0} = I_t, I_{t1}, I_{t2}, \ldots$ denote the total prolongations starting at I_t, defined on fibered manifolds $B_{t0} = B_t, B_{t1}, \ldots$. If y_1 denotes the fiber coordinate of B_{t1} over B_{t0}, then I_{t1} has the form $\lambda dt + \mu dy_0$, where either λ or μ depends non trivially on y_1. Since the last derived system of I does not drop rank at the equilibrium, neither does I_{t1} and we have that not both λ and μ vanish at the equilibrium. Now, $\mu \neq 0$ at the equilibrium point, since $y_0 \equiv c$ is a solution curve to I_t, which would not have a lift to I_{t1} if $\mu = 0$, since dt is required to remain the independence condition of all Cartan prolongations. From continuity $\mu \neq 0$ around the equilibrium point. So we can define $y_1 := -\lambda/\mu$, and I_{t1} can be written as $dy_0 - y_1 dt$. We can continue

this process for every Cartan prolongation, both of I_t and of I. This brings J_t in Goursat normal form in a neighborhood of the equilibrium point.

Now we will argue that we don't need to prolong I to establish equivalence. Since J is a Cartan prolongation, and therefore a total prolongation, its first derived system will be equivalent to the first derived system of J_t. Continuing this we establish equivalence between I and I_{tn}, where $I_{tn} = \{dy_0 - y_1 dt, \ldots, dy_{n-1} - y_n dt\}$. So we have $y = (y_0, \ldots, y_n) = y(x, u, t)$.

Next we will show that $y_0, \ldots, y_n$ are independent of time, and that $y_0, \ldots, y_{n-1}$ are independent of u. By assumption y_0 is independent of time. Since the corresponding derived systems on each side are equivalent, $dy_0 - y_1 dt$ is equivalent to the last one-form in the derived flag of I. Since the differential du does not appear in this one-form, y_0 is independent of u. Analogously, $y_i, i = 1, \ldots, n-1$ are all independent of u. Since the $y_i, i = 1, \ldots, n$ are repeated derivatives of y_0, and since I is time invariant, these coordinates are also independent of time.

We still have to show that the mapping $x \mapsto y$ is a valid coordinate transformation. Suppose $dy_0, \ldots, dy_{n-1}$ are linearly dependent at the equilibrium. Then, J_t drops rank at the equilibrium, and since we have equivalence, so would I. But from the form of I we can see this is not the case.

Therefore $y_i = y_i(x), i = 0, \ldots, n-1, y_n = y_n(x, u)$ and the system J_t is just a chain of integrators with input y_n. The original system I is equivalent to this linear system by a coordinate transformation on the states and a state dependent and time invariant feedback. This coordinate transformation is well defined around the equilibrium point. It is therefore feedback linearizable by a static feedback that is time invariant. Note that $\partial y_n / \partial u \neq 0$ because y_n is the only of the y variables that depends on u. ■

Example 2.26 Notice that in our definition the system

$$\begin{aligned} \dot{x}_2 &= u \\ \dot{x}_1 &= x_2^3 \\ y &= x_1 \end{aligned} \tag{2.15}$$

is not flat around the origin, because we get $u = \frac{\ddot{y}}{3\dot{y}^{2/3}}$ so that curves with $\dot{y} = 0$ and $\ddot{y} \neq 0$ have no lift. It is also not feedback linearizable at the origin.

We will now show that in the case of a time invariant system, we don't need the assumption of time invariant flatness to conclude static feedback linearizability. We will require the following preliminary result, which appeared in a proof in [?].

Lemma 2.27 *Let $\alpha = A_i(x, u)dx_i - A_0(x, u)dt$ be a one-form (using implicit summation) on a manifold M with coordinates (x, u, t), and suppose we can write $\alpha = dX(x, u, t) - U(x, u, t)dt$. Then we can also write α as $\alpha = dY(x) - V(x, u)dt$, i.e., we can take the function X independent of time and the input, and we can take U independent of time. If we know in addition that $\alpha = A_i(x)dx_i - A_0(x)dt$. then we can scale α as $\alpha = dY(x) - V(x)dt$, i.e., we can take V independent of u as well.*

Proof: See [?]. ■

The following theorem seems to be implied in [?], but the proof there refers to a general discussion of Cartan's method of equivalence as applied to control systems in [?]. We work out the proof for this special case.

Theorem 2.28 *A single input time invariant control system is differentially flat if and only if it is feedback linearizable by static, time invariant feedback.*

Proof: Sufficiency is trivial, so we shall only prove necessity. Let the control system be $I = \{dx_1 - f_1(x,u)dt, \ldots, dx_n - f_n(x,u)dt\}$, where u is a scalar control, i.e. the system has codimension 2. Let $\{\alpha^i, i = 1, \ldots, n\}$ and $\{\alpha_t^i, i = 1, \ldots, n\}$ be one-forms adapted to the derived flag of I, I_t respectively. Thus, $I^{(i)} = \{\alpha^1, \ldots, \alpha^{n-i}\}$ and $I_t^{(i)} = \{\alpha_t^1, \ldots, \alpha_t^{n-i}\}$. Since I does not contain the differential du, the forms $\alpha^1, \ldots, \alpha^{n-1}$ can be taken independent of u. Since I is time invariant, the forms $\alpha_1, \ldots, \alpha_n$ can be chosen independent of time. We can thus invoke the second part of Lemma 2.27 for the forms $\alpha^1, \ldots, \alpha^{n-1}$.

Assume $n \geq 2$. As in Theorem 2.25 we have equivalence between α^1 and $\alpha_t^1 = dy_0(x,t) - y_1(x,t)dt$ (if $n = 1$ we have $y_n = y_n(x,u,t)$, which we will reach eventually). Since I is time invariant we can choose α^1 time independent: $\alpha^1 = A_i(x)dx_i - A_0(x)dt$. From Lemma 2.27 we know that we can write α^1 as $dY_0 - Y_1 dt$ where Y_0, Y_1 are functions of x only.

Again according to Lemma 2.27, we can write $\alpha^2 = dV(x) - W(x)dt$. Now from,

$$\begin{aligned} 0 &= d\alpha^1 \wedge \alpha^1 \wedge \alpha^2 \\ &= -dY_1 \wedge dt \wedge dY_0 \wedge dV \end{aligned}$$

we know $V = V(Y_1, Y_0)$. And from

$$\begin{aligned} 0 &\neq d\alpha^2 \wedge \alpha^1 \wedge \alpha^2 \\ &= -dW \wedge dt \wedge dY_0 \wedge dV \end{aligned}$$

we know that $\gamma_1 := \partial V/\partial Y_1 \neq 0$. Then, writing $\gamma_0 := \partial V/\partial Y_0$, (and $\simeq$ denotes equivalence in the sense that both systems generate the same ideal),

$$\begin{aligned} \{\alpha^1, \alpha^2\} &\simeq \{dY_0 - Y_1 dt, \gamma_1 dY_1 + \gamma_0 dY_0 - W dt\} \\ &\simeq \{dY_0 - Y_1 dt, \gamma_1 dY_1 + \gamma_0 Y_1 dt - W dt\} \\ &\simeq \{dY_0 - Y_1 dt, dY_1 - (-\gamma_0 Y_1 + W)/\gamma_1 dt\} \\ &:= \{dY_0 - Y_1 dt, dY_1 - Y_2 dt\}. \end{aligned} \tag{2.16}$$

Where Y_2, defined to be $Y_2 = (-\gamma_0 Y_1 + W)/\gamma_1$, is independent of (t,u) since $(\gamma_1, \gamma_0, Y_1, W)$ are. One can continue this procedure, at each step defining a new coordinate Y_i. In the last step the variable $W = W(x,u)$ (this will also be the first step if $n = 1$), and therefore Y_n depends on u nontrivially. Hence we obtain equivalence between I and $\{dY_0 - Y_1 dt, \ldots, dY_{n-1} - Y_n dt\}$ with $Y_i = Y_i(x)$, $i = 0, \ldots, n-1$, and $Y_n = Y_n(x,u)$, i.e., feedback linearizability by static time invariant feedback. ■

Corollary 2.29 *If a time invariant single input system is differentially flat we can always take the flat output as a function of the states only:* $y = y(x)$.

None of these results extend easily to higher codimensions. The reason for this is that only in codimension two we can find regularity assumptions on the original system such that every Cartan prolongation is a total prolongation. This is related to the well known fact that for SISO systems static linearizability is equivalent to dynamic linearizability. For MIMO systems we cannot express these regularity conditions on the original system: we have to check regularity on the prolonged systems.

2.7 Examples

2.7.1 Generic Classes

If a linear system is controllable, it can be put into Brunovsky canonical form. The heads of the chains of integrators are the flat outputs.

All feedback linearizable systems can be put into Brunovsky canonical form and are therefore flat. The feedback linearization can be dynamic or static. All results about feedback linearizable systems, as in [?], apply to flatness.

A fully actuated Lagrangian system has the form

$$M(x, \dot{x})\ddot{x} + C(x, \dot{x})\dot{x} + K(x) = F \tag{2.17}$$

where $\dim F = \dim x$. Clearly, we can determine all states and F from the outputs x. These systems include most robotic manipulators, and indeed, computed torque is a well established control technique for trajectory tracking with robotic manipulators. In [?] necessary and sufficient conditions were given for Lagrangian systems with $\dim F = \dim x - 1$. The same author also gives conditions for flatness of codimension 3 systems in [?]. The *pure feedback form* described in [?] is feedback linearizable in an open and dense set and therefore automatically flat in that set. Finally, [?] gives a catalog of flat systems, including some examples not presented here.

2.7.2 Kinematic Car

Consider the equations of motion for a kinematic car (see Figure 2.1),

$$\begin{aligned} \dot{x} &= \cos\theta \cos\phi \; v_1 \\ \dot{y} &= \sin\theta \cos\phi \; v_1 \\ \dot{\theta} &= \frac{1}{l} \sin\phi \; v_1 \\ \dot{\phi} &= v_2. \end{aligned} \tag{2.18}$$

Here, (x, y) is the position of the rear axle, θ is the angle between the horizontal and the car, ϕ is the steering angle, v_1 is the forward velocity of the front wheels, v_2 is the steering angle velocity and l is the distance between front and rear axle.

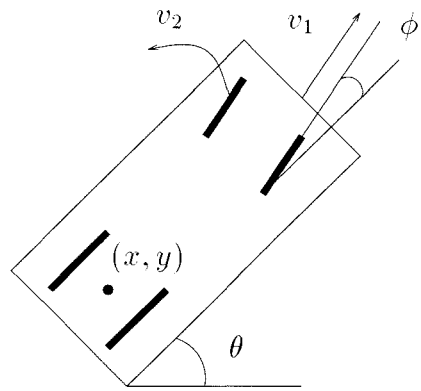


Figure 2.1 The kinematic car.

This system is flat with flat outputs (x, y), the position of the rear axle. If we want to back up a truck into a loading dock these outputs are the same as the tracking outputs. For other problems, e.g. when the driver is trying to negotiate a window at a drive-thru restaurant, it might be more appropriate to generate a trajectory for the front cab of the car. Then the tracking outputs are $(x + l\cos\theta, y + l\sin\theta)$, and the zero dynamics can be parametrized by ϕ. In general it is desirable to keep ϕ small. This can be achieved by setting up a cost criterion that minimizes a weighted integral of the tracking error and the magnitude of ϕ. Note that ϕ can be expressed in terms of the flat outputs as

$$\phi = \arctan(\frac{1}{l}(\dot{x}^2 + \dot{y}^2)^{3/2}, \dot{x}\ddot{y} - \dot{y}\ddot{x}). \tag{2.19}$$

2.7.3 Thrust Vectored Aircraft

The ducted fan is a model of a thrust vectored aircraft mounted on a stand, as shown in Figure 2.2 and Figure 2.3. The fan was built at Caltech to study the pitch dynamics of highly maneuverable aircraft. We refer the reader to [?] for a detailed description of this apparatus.

The fan mounted on the stand can be approximated by the *planar ducted fan* depicted in Figure 2.4. The planar ducted fan is thought to move in an (x, z) plane obtained by rolling out the sphere scribed out by the fan on the stand as (ϕ_1, ϕ_2) go through their range of values. The relation between the coordinates for the ducted fan on the stand and the planar ducted fan is simply given by

$$\begin{aligned} x &= R_f\phi_1 \\ z &= -R_f\phi_2 \\ \theta &= \phi_3, \end{aligned} \tag{2.20}$$

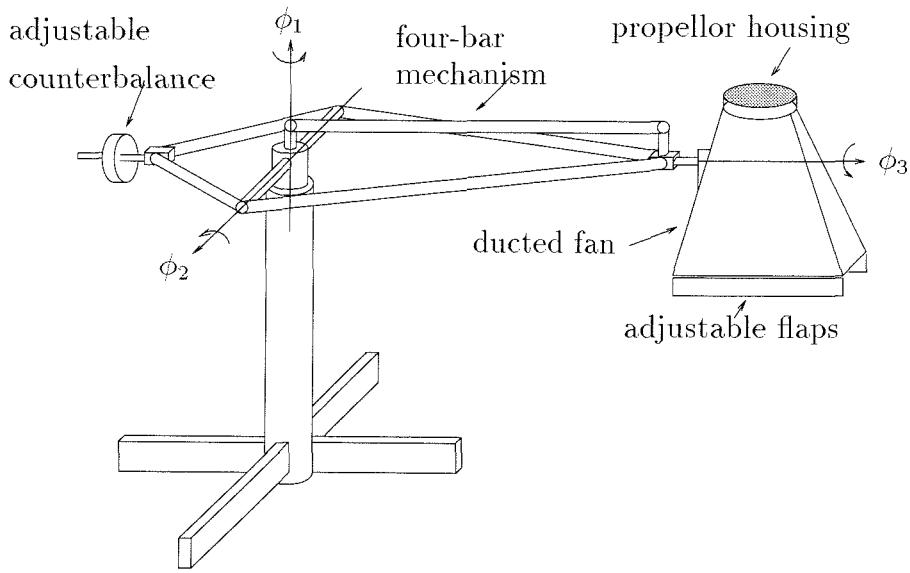


Figure 2.2 Ducted fan with stand.

where R_f is the length of the boom carrying the fan. In the planar idealization the fan has infinite travel in the x and z directions. Note that the z direction is taken positive downward to remain consistent with the convention used in aircraft dynamics. Since the planar fan is thought to move in the (x, y) plane, it is convenient to associate the variable y with the vertical position measured positive upward, so $y \equiv -z$. In plots we will use the variable y.

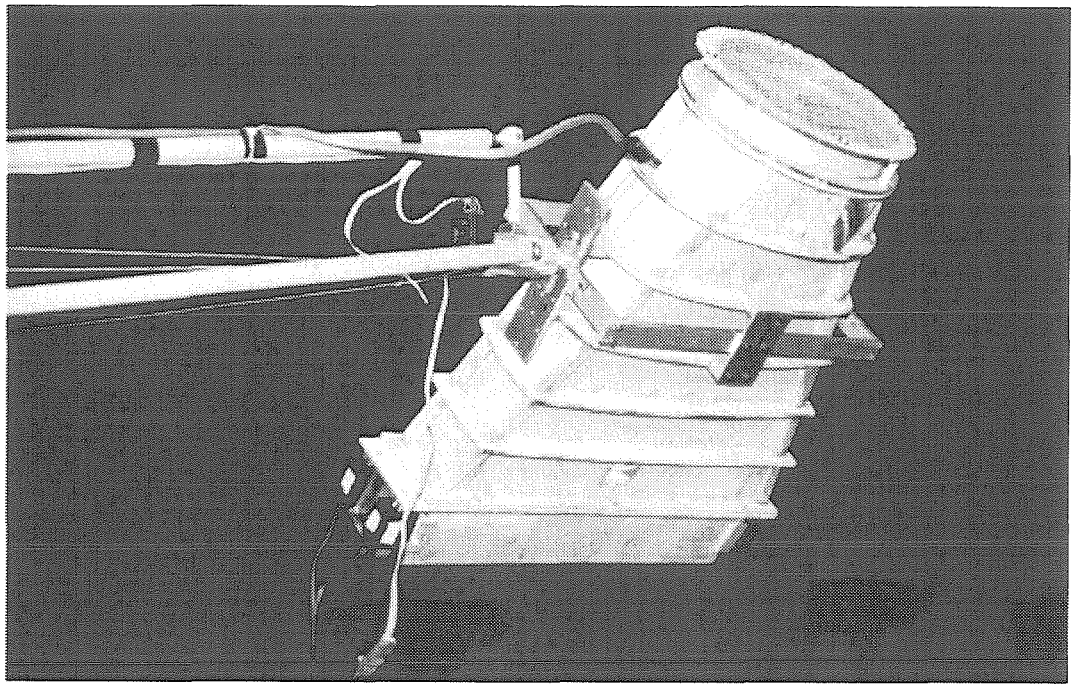


Figure 2.3 First generation Caltech ducted fan.

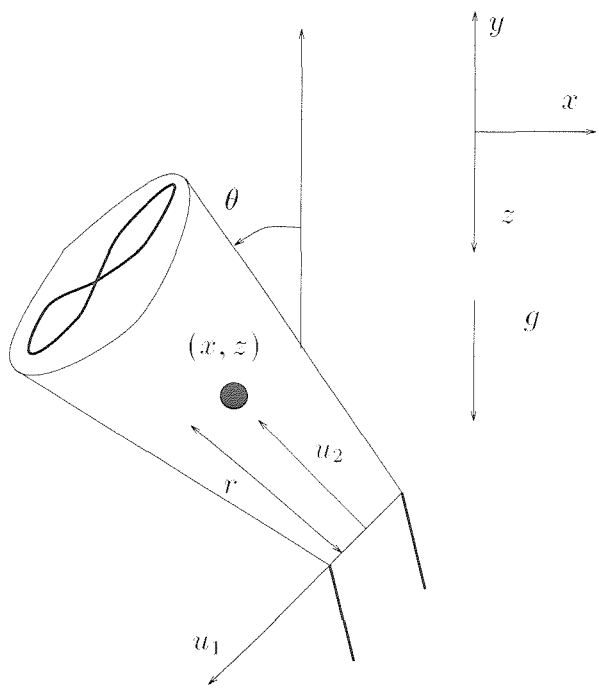


Figure 2.4 Planar ducted fan.

parameter	value	units
g	9.81	m/s^2
r	0.25	m
J	0.0475	kg m^2
m_x	4.19	kg
m_y	3.71	kg
m_g	0.27	kg

Table 2.1 Parameter values for the ducted fan

The fact that the fan is mounted on a stand introduces the following parasitic dynamics that are not present in the planar fan. The real ducted fan is mounted on a stand with a counterweight that moves in as the fan moves up. This results in inertial masses m_x and m_z in the x and z direction respectively, that change with the z coordinate. We do not take the variation of these inertial masses with z into account for the flat model but take their value around hover. The counterweight also results in an effective weight m_g different than the masses in x and z direction. In addition to these inertial effects, the interaction of the rotating propeller and the fan rotating around the ϕ_1 axis causes a Coriolis force. If we ignore these parasitic dynamics and the aerodynamic forces, the equations of motion for the planar ducted fan describe a flat system. This flat approximation of the ducted fan on the stand will be our prime example in the numerical and experimental data presented in later chapters.

We can apply an arbitrary force on the center of mass by adjusting the magnitude and the direction of the thrust, or equivalently, the parallel and perpendicular component of the thrust. After shifting the parallel thrust $u_2 \rightarrow u_2 + m_g g$ to compensate for gravity, the equations of motion for the planar ducted fan are:

$$\begin{pmatrix} m_x \ddot{x} \\ m_z \ddot{z} \\ J\ddot{\theta} \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ -\sin\theta & -\cos\theta \\ r & 0 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 + m_g g \end{pmatrix} + \begin{pmatrix} 0 \\ m_g g \\ 0 \end{pmatrix} \tag{2.21}$$

where (x, z) are the coordinates of the center of mass, θ is the angle with the vertical, u_1 is the force perpendicular to the fan body, u_2 is the force parallel to the fan body, r is the distance between the center of mass and the point where the force is applied, g is the gravitational constant, m_x, m_z is the inertial mass of the fan in the (x, z) direction respectively, m_g is the gravitational mass of the fan, and J is the moment of inertia of the fan. The tracking outputs are the (x, z) coordinates of the center of mass. The values of the parameters for the ducted fan that was built in our lab are given in Table **??**.

Note that these equations are almost identical to the ones presented in [**?**] and [**?**], except for the small parameter ϵ multiplying the u_2 term that occurs in those references. We do not impose this restriction here. Also, in our case $m_x \neq m_z$ in

general.

The planar approximation (??) is differentially flat. In [?] the flat outputs were shown to be

$$\begin{aligned} x_f &= x - \frac{J}{m_x r}\sin\theta \\ z_f &= z - \frac{J}{m_z r}\cos\theta. \end{aligned} \tag{2.22}$$

Note that these outputs are not fixed in body coordinates. We can dynamically feedback linearize this system by the following dynamic extension:

1. Add u_2 as a state, and let $\dot{u}_2 = u_3$.
2. Apply the following input transformation: $u_4 = -2u_1\dot{\theta} + u_3$.
3. Add u_4 as a state, and let $\dot{u}_4 = u_5$.

The extended system is

$$\begin{aligned} m_x\ddot{x} &= -m_g g\sin\theta + \cos\theta u_1 - \sin\theta u_2 \\ m_z\ddot{z} &= m_g g(-\cos\theta + 1) - \sin\theta u_1 - \cos\theta u_2 \\ J\ddot{\theta} &= ru_1 \\ \dot{u}_2 &= u_4 + 2u_1\dot{\theta} \\ \dot{u}_4 &= u_5 \end{aligned} \tag{2.23}$$

with new inputs (u_1, u_5). The coordinate transformation

$$(x_f, \ldots, x_f^{(3)}, z_f, \ldots, z_f^{(3)}) \mapsto (x, z, \theta, \dot{x}, \dot{z}, \dot{\theta}, u_2, u_4) \tag{2.24}$$

has determinant

$$-\frac{(m_g gr - J\dot{\theta}^2 + ru_2)^2}{m_x^2 m_z^2 r^2} \tag{2.25}$$

which is nonzero around the origin. The determinant of the I/O decoupling matrix,

$$\det\left(\frac{\partial(x_f^{(4)}, z_f^{(4)})}{\partial(u_1, u_5)}\right) = \frac{-m_g gr + J\dot{\theta}^2 - ru_2}{Jm_x m_z}, \tag{2.26}$$

is nonzero around the origin. Therefore the extended system has well defined relative degree around the origin, and is feedback linearizable. It is interesting to note that both the decoupling matrix and the coordinate transformation become singular if no gravity is present. The system will still be flat in zero gravity, since flatness is not restricted to an equilibrium point: in an open and dense set the system will still have linear structure. We will not explicitly use the feedback linearization, since the form in equation (??) is sufficient for trajectory generation purposes.

Note that the zero dynamics (with respect to outputs (x, z)),

$$\ddot{\theta} = \frac{m_g g r}{J} \sin\theta, \tag{2.27}$$

are unstable. Imposing a bound on θ will impose a bound on the zero dynamics. The variable θ can be expressed in terms of the flat outputs as

$$\tan\theta = \frac{-m_x \ddot{x}_f}{-m_z \ddot{z}_f + m_g g}. \tag{2.28}$$

2.7.4 Submarine

Consider the a rigid body symmetric about the y-axis with 3 forces acting in 1 point on the y-axis, depicted in Figure ??. This body could model an underwater vehicle, a zeppelin or a missile.

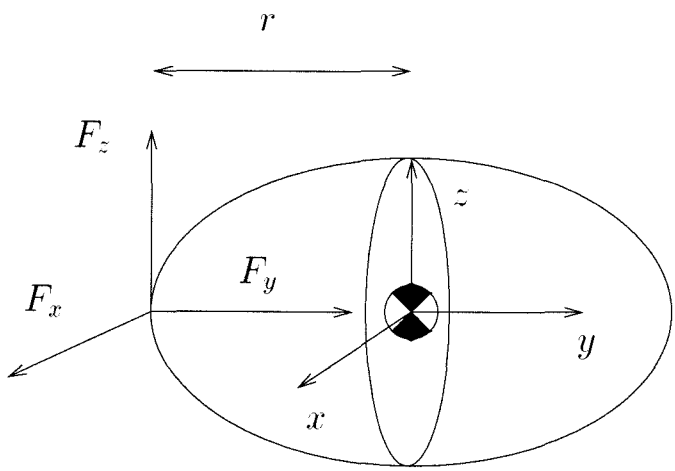


Figure 2.5 The axially symmetric body with 3 forces acting in a point.

Due to the axial symmetry we ignore the rotation about the y-axis. We have no actuating torque there, and do not measure the angle. In the language of geometric mechanics, we reduce the dynamics by the symmetry around the y-axis. Recall Euler's equations for a rigid body in body coordinates [?],

$$J\dot{\omega}^b + \omega^b \times J\omega^b = \tau^b, \tag{2.29}$$

where J is the inertia matrix which will be diagonal with $J_x = J_z$ due to axial symmetry. ω^b is the rotational velocity in body coordinates. Put the origin at the center of mass, and let the forces act at the point $(0, r, 0)$ (note that r is negative in the picture); then $\tau = (rF_z, 0, -rF_x)$.

Suppose we observe the point $p^b = (0, -\frac{J_x}{mr}, 0)$ on the body. This is the equivalent

of the center of oscillation for the ducted fan. Our observation in a spatial frame is

$$p^s = p_c^s + Rp^b \tag{2.30}$$

where p_c^s are the coordinates of the center of mass in the spatial frame, and R is the rotation matrix from body to spatial coordinates. Since we ignore rotation about y, R has 2 unknown parameters, say ϕ and θ. Differentiating equation (??) gives

$$\dot{p}^s = \dot{p}_c^s + R\dot{p}^b + R\hat{\omega}^b p^b, \tag{2.31}$$

where $\hat{\omega}p = \omega \times p$. Note that $\dot{p}^b = 0$, since p^b is a body fixed point.

Differentiating equation (??) and using Euler's equations and $F^s = RF^b = m\ddot{p}_c^s$ gives

$$m\ddot{p}^s = R\begin{pmatrix} -J_y\omega_x^b\omega_y^b/r \\ F_y + J_x((\omega_x^b)^2 + (\omega_z^b)^2)/r \\ -J_y\omega_y^b\omega_z^b/r \end{pmatrix} - \begin{pmatrix} 0 \\ mg \\ 0 \end{pmatrix}. \tag{2.32}$$

Now we need the extra assumption that $\omega_y^b \equiv 0$. Then we know the direction of the second column of R, that is, we know the attitude up to a rotation about the y-axis, which we ignore. This gives us θ and ϕ. One more differentiation gives ω_x^b, ω_z^b, then equation (??) gives us F_y, and the Euler equations give us F_x, F_z.

If we want to track the center of mass instead of the center of oscillation, we will have unstable zero dynamics, entirely analogous to the ducted fan.

Note that even though we have no direct actuation of the roll rotation, we can rotate about the y-axis by performing a sequence of noncommuting rotations about the x and z axes.

2.8 Summary

We have presented a definition of flatness in terms of the language of exterior differential systems and prolongations. Our definition remains close to the original definition due to Fliess [?, ?], but it involves the notion of a preferred coordinate corresponding to the independent variable (usually time).

Using this framework we were able to recover all results in the differential algebra formulation. In particular we showed that flat systems can be put in linear form in an open and dense set. This set need not contain an equilibrium point, and this linearizability therefore does not allow one to use most methods from linear systems theory. In other words, although flatness implies a linear *form*, it does not necessarily imply a linear *structure*. For a SISO flat system we resolved the regularity issue, and established feedback linearizability around an equilibrium point. We also resolved the time dependence of flat outputs in the SISO case.

The most important open question is a characterization of flatness in codimension higher than two. See [?, ?] for an answer to this question in some special cases.

Chapter 3

Trajectory Generation for Differentially Flat Systems

3.1 Introduction

As was announced in the first chapter, software tools form an important part of this dissertation. In this chapter we present some important trajectory generation problems for flat systems, and algorithms and software to solve them. All algorithms will describe which numerical computations have to be performed, and we will give an indication of the computational cost. All problems treated in this chapter are finite horizon and anti-causal in the sense that some information about the future and final behavior of the trajectory has to be given beforehand. This information can be the entire desired trajectory, or the points we want to steer between. The material in this chapter is based on [?].

Although the two degree of freedom scheme in Figure 1.2 is quite common [?, ?], implementation issues are usually ignored. We added an operator and an integrator box to the scheme to indicate emphasis on on-line input. We focus on digital implementation and computational feasibility. Work to this effect was presented in [?, ?], where trajectory computations were done in pseudo real time, i.e., pilot input was given on line, but trajectory output was generated at a rate several orders of magnitude slower than the controller rate. The particular application in [?] was trajectory generation for commercial aircraft between via points, which were supplied at intervals of several minutes. Clearly this allows fairly complicated computations to be performed for the trajectory generation. Our goal in this chapter is to present methods to reduce that computation to the order of seconds.

3.2 Problem Setup and Notational Conventions

Recall that the basic equations for flat systems presented in the previous chapter were:

$$\begin{aligned} z &= \psi(x, u, u^{(1)}, \ldots, u^{(l)}) \\ (x, u) &= \phi(z, z^{(1)}, \ldots, z^{(l)}) \end{aligned} \tag{3.1}$$

for some l. That is, we can express the state and the inputs as functions of the flat outputs and their derivatives. This was the original definition in the differential algebraic setting.

All code is written in ANSI C , all computation times refer to an Intel 486 DX2 CPU, running at 66 MHz. Throughout this chapter we will denote flat outputs by z and tracking outputs by y. We will be looking at trajectories over a finite time interval $[t_0, t_1]$. We will approximate trajectories by polynomials, since this allows us to perform derivative calculations symbolically.

All examples are based on the ducted fan presented in Section 2.7. We take into account the different inertial and gravitational masses, but no aerodynamic forces or stand dynamics. The extended system, i.e. the system that allows static feedback linearization, for this example has 8 states and 2 inputs.

3.3 Point-to-Point Steering Problems

The easiest tracking problem is where we want to steer from one point in state space to another point in state space. For this problem it is irrelevant whether the flat outputs are the tracking outputs or not, since we are given the entire state at two points in time. Suppose we want to steer from $x(t_0) = x_0$ to $x(t_1) = x_1$. Assume the inputs and their derivatives at both times are also specified. Then we can compute the flat outputs and their derivatives at the initial and final times. We parametrize the flat outputs z as

$$z_i(t) = A_{ij}\phi_j(t) \tag{3.2}$$

(using implicit summation) where the $\phi_j(t)$ are some basis functions. We need to solve for the coefficients A_{ij} in the following equations:

$$\begin{aligned} z_i(t_0) &= A_{ij}\phi_j(t_0) & z_i(t_f) &= A_{ij}\phi_j(t_0) \\ &\vdots & &\vdots \\ z_i^{(l)}(t_0) &= A_{ij}\phi_j^{(l)}(t_0) & z_i^{(l)}(t_f) &= A_{ij}\phi_j^{(l)}(t_f). \end{aligned} \tag{3.3}$$

We need enough basis functions so that these equations have a solution. If the dimension of the state is n and the dimension of the input is m, then we need $2(n + m(l + 1))$ coefficients. The point-to-point steering problem then amounts to solving a system of linear equations with $2(n + m(l + 1))$ unknowns. It is, of course, possible to overparametrize the flat outputs in Equation (??), thereby achieving an extra degree of freedom that will not only allow us to steer from an initial to a final point, but also minimize some cost function of interest. We will address this issue in Section ??.

After we compute the coefficients A_{ij} we need to compute the trajectory at a number of time points for our real time implementation from equation (??). The more time points we have, the more accurate our nominal trajectory, and the better the performance will be. Solving for the states and inputs amounts in general to

N	Time ([s])
50	0.66
100	1.05
200	1.76
300	2.58

Table 3.1 Computation time for point-to-point steering.

solving a nonlinear systems of equations for each desired trajectory point. Since this is an iterative process, the computation time is hard to predict. For the ducted fan we can find symbolic expressions for the states from the flat outputs and their derivatives, since

$$\tan\theta = \frac{-m_x\ddot{x}_f}{-m_z\ddot{z}_f + m_g g}, \tag{3.4}$$

and from θ we can find all states and inputs. Note that x and z here are the horizontal and vertical position of the fan respectively, and have nothing to do with the state and flat output of a system as in Equation (??). For a reasonable number of trajectory points, the computation of the states from the flat outputs will be longer than the computation of the coefficients from equation (??), even if we have symbolic expressions, as for the ducted fan. Therefore the computation time is mainly determined by the number of time points desired in the output trajectory. Table ?? lists the computation time vs. number of time points for the ducted fan. For this case we can solve for the states and inputs in closed form, and don't need to resort to numerical root finding. Each flat output is parametrized by 8 polynomials (of degree 0 to 7). If we do not require fixing the inputs at both ends, but only the states, we need the flat outputs and 3 derivatives, so that we need 8 polynomials for each flat output for the equations (??) to have a solution.

Figure ?? shows the trajectory for an initial state $(x, y, \theta, \dot{x}, \dot{y}, \dot{\theta}) = (0, 0, 0, 0, 0, 0)$ at $t_0 = 0$, and a final state $(x, y, \theta, \dot{x}, \dot{y}, \dot{\theta}) = (1.0, 0, 0, 0, 0, 0)$ at $t_1 = 3.0$. Remember that y is the vertical position, measured positive upward.

3.4 Least Squares Approximate Trajectories for Flat Systems

If the tracking outputs are the flat outputs and we are given a trajectory to track, we still want to parametrize the flat outputs by basis functions. The reason is that computation of the states requires differentiation of the flat outputs, up to several orders, which is a numerically ill conditioned computation. Large magnitudes of the derivatives can prevent convergence of the nonlinear solver of equation (??).

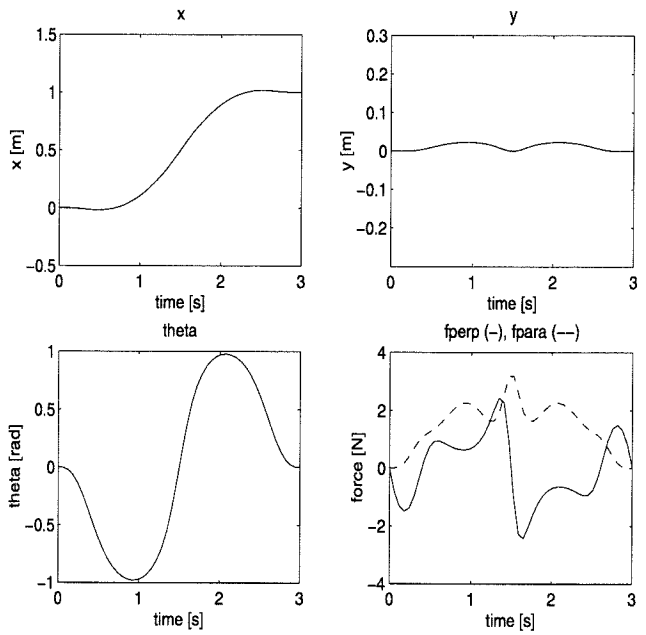


Figure 3.1 Trajectory for a point-to-point steering problem.

The parametrization of the outputs leads to the least squares problem:

$$\min_{A} \int_{t_0}^{t_1} \big(z_d(s) - A\phi(s)\big)^* W(s) \big(z_d(s) - A\phi(s)\big) ds \tag{3.5}$$

where A_{ij} is the coefficient of basis function j in flat output i, $\phi(t)$ is the vector of basis functions evaluated at time t, z_d is the desired flat output, and $W(t)$ is a time varying weighting matrix. This minimization has the closed form solution:

$$A = M^{-1}L \tag{3.6}$$

where

$$\begin{aligned} M_{ij} &= \int_{t_0}^{t_1} \phi_i(s) W_{ij}(s) \phi_j(s) ds \\ L_{ij} &= \int_{t_0}^{t_1} \phi_i(s) W_{ij}(s) z_{dj}(s) ds. \end{aligned} \tag{3.7}$$

Note that the problem is decoupled with respect to the different outputs: we can compute the coefficients for each flat output separately. In our implementation we approximate integration by summation. The computation of M^{-1} only has to be performed once, so that there is no great savings in picking orthogonal basis functions. For polynomial basis functions on a finite interval, the higher order polynomials have large magnitude at the boundaries. This results in numerical

inaccuracy over the rest of the interval, and therefore orthogonal basis functions are not always the best choice.

Since we are only minimizing an integrated error, the resulting trajectory does not necessarily start at the state we are at. However, we can fix initial and final conditions (or conditions at any time for that matter), by imposing linear constraints on the coefficients A_{ij} exactly as in equation (??). Suppose the linear constraints on A are given by $z = FA$. Then we have to find a particular solution to these equations, say $A_0 = F^{\dagger}z$, and we can reparametrize $A = A_0 + F^{\perp}A_1$, The modified error then becomes

$$z_d(t) - A_0\phi(t) - F^{\perp}A_1\phi(t) =: \tilde{z}_d(t) - \tilde{A}\phi(t) \tag{3.8}$$

with solution $A_1 = \tilde{M}^{-1}\tilde{L}$, where

$$\begin{aligned} \tilde{M}_{ij} &= F^{\perp *}\Big(\int_{t_0}^{t_1} \phi_i(s)W_{ij}(s)\phi_j(s)ds\Big)F^{\perp} \\ \tilde{L}_{ij} &= F^{\perp *}\int_{t_0}^{t_1} \phi_i(s)W_{ij}(s)(z_d(s) - A_0\phi(s))_j ds. \end{aligned} \tag{3.9}$$

Table ?? shows the computation time for the trajectory depicted in Figure ??. NPTS refers to the number of time points in the input trajectory, the pair NPOL is the number of polynomials used to approximate each flat output, and cost/NPTS is the approximation error over the number of points in the input trajectory. Since the error is linear in NPTS, this is the right cost measure. We note that the computation time is linear in the number of points, and that increasing the number of basis functions stops improving the error fairly soon.

NPTS	NPOL	Time ([s])	cost/NPTS
30	(4,4)	0.82	
60	(4,4)	0.98	
120	(4,4)	1.26	0.18
120	(6,6)	1.87	0.087
120	(8,8)	2.7	0.082

Table 3.2 Computation time for least squares approximation.

Figure ?? shows the desired and approximated trajectory for NPOL = (6,6). Note that the approximation does not go through the desired initial and final point.

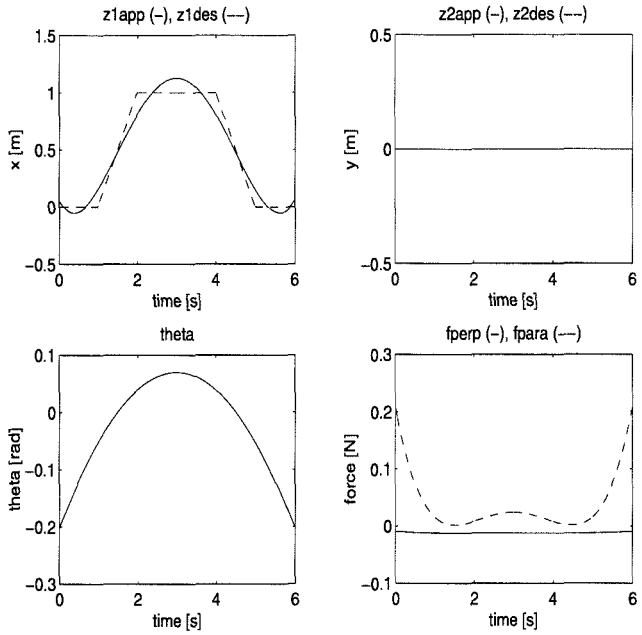


Figure 3.2 Trajectory for a least squares approximation problem.

3.5 Approximate Tracking for Non-minimum Phase Systems

In general the flat outputs that parametrize all system trajectories may not be the outputs that we want to track. For the ducted fan and the submarine we want to track the center of mass, which is different from the center of oscillation. For the kinematic car we might not be interested in the position of the rear axle, but of some other point. In general, any flat system with tracking outputs other than the flat outputs are examples of such systems.

Tracking and approximate tracking for non-minimum phase systems has received a fair amount of attention in the literature [?, ?]. A fundamental limitation in the tracking performance was demonstrated by Grizzle et al. in [?], where it was shown that under fairly mild conditions the stability of the zero dynamics is a necessary condition for asymptotic tracking with internal stability.

In [?] it is suggested to redefine the tracking outputs to the flat outputs, so that the tracking problem becomes trivial. However, this may have undesired effects for the zero dynamics of the original system. Tracking the flat outputs will allow exact tracking but might drive the zero dynamics of the original outputs to undesirable magnitude. If we maintain the original outputs, we can still parametrize all system trajectories with the flat outputs, but in general for each trajectory of the tracking outputs, we can find more than one trajectory of the flat outputs. This freedom can be used to advantage to minimize an additional cost function. Typically, we pick this cost function to bound the magnitude of the coordinates describing the zero

dynamics, or the actuator effort.

The stable inversion proposed in [?, ?, ?], is an iterative solution to the tracking problem with unstable zero dynamics. This solution offers exact tracking. First a preliminary input trajectory (prologue) is used to bring the state to a starting point on the unstable zero dynamics manifold while keeping the outputs zero. The actual stable inverse follows the desired output trajectory exactly while steering the internal dynamics from the unstable to the stable zero dynamics manifold. Finally an epilogue brings the state from the stable zero dynamics manifold to an equilibrium while keeping the outputs zero. During the actual tracking no bounds on the zero dynamics are imposed. The trajectory from the stable to the unstable zero dynamics manifold is obtained by repeated solution of a differential equation over the entire time interval. It also requires numerical differentiation of the desired trajectory, since the zero dynamics are dependent on this trajectory and its derivatives. A solution in the Fourier domain involving repeated convolution is presented in [?]. Our solution does not require a prologue to bring the zero dynamics to the unstable zero dynamics manifold, and is computationally much simpler. We can bound the zero dynamics during tracking, at the cost of a larger tracking error. The obvious drawbacks of our solution are that it only works with flat systems and only offers approximate tracking.

The approach proposed by Getz et al. in [?] is closer to ours in the sense that it trades off stability of the internal dynamics versus tracking performance. The state trajectory that this method tries to follow is a first order improvement on the one degree of freedom design, in the sense that the trajectory for the internal configuration is an instantaneous equilibrium generated by the output trajectory, but the first and higher derivatives of the internal configuration are set to zero. Therefore the total state trajectory is not feasible, in contrast with our approach.

We now proceed with the solution of the approximate tracking for flat non-minimum phase systems. The combined tracking and internal dynamics objective leads to the minimization problem:

$$\min_{A} \int_{t_0}^{t_1} \left(y(A,s) - y_d(s)\right)^* W(s) \left(y(A,s) - y_d(s)\right) + \lambda K(z, z^{(1)}, \ldots, z^{(l)}) ds, \quad (3.10)$$

where K is a function which penalizes the internal dynamics and λ trades off the tracking accuracy against the internal dynamics. With $\lambda = 0$ we will track exactly, within the accuracy of the basis function parametrization, but we have no control over the zero dynamics. Since the tracking outputs do not completely determine the dynamics of the system if they are not the flat outputs, $\lambda = 0$ will lead to an ill conditioned minimization problem: there will be infinitely many solutions for A leading to exact tracking. Therefore, for this problem to be well defined, we need a nonzero penalty on the internal dynamics. With λ large we will have poor tracking but small zero dynamics. The tradeoff between stability and tracking is an important feature of our method. Note that since the tracking outputs are no longer the flat outputs, the outputs y in Equation (??) are arbitrary functions of the coefficients A.

If the system has well defined relative degree ans *stable* zero dynamics, we can achieve asymptotic tracking by I/O linearization, as was mentioned in Chapter 1. However, even though the zero dynamics are stable, the magnitude of the internal states might be quite large. A cost criterion of the form (??) can also be applied to systems with stable zero dynamics to explicitly bound the magnitude of the internal variables. This only works for finite horizon trajectories, whereas the I/O linearization works for infinite horizon.

The minimization problem (??) is in general a nonconvex nonlinear minimization problem, so that we cannot guarantee convergence to a global minimum. We approximate the integral with a discrete sum. For each time point in this sum we need to compute the flat outputs and the states from the coefficients A_{ij}, and then evaluate the integrand. This results in long computation times. Computation time depends on the particular problem and the required accuracy. After the coefficients A_{ij} have been found, we still have to compute the state-input trajectory at a number of time points. This latter computation will only take a fraction of the time needed to compute the minimum of the cost (??), so that the minimization determines the computation time.

For the ducted fan, a sensible cost criterion reflecting the desire to keep the internal dynamics bounded is

$$\int_{t_0}^{t_1} \big(y(A,s) - y_d(s)\big)^* \big(y(A,s) - y_d(s)\big) + \lambda\theta^2 ds, \tag{3.11}$$

where the entries of f are polynomials. The trajectory $y(t)$ is depicted in the first 2 windows of Figures ?? and ??. Figure ?? shows the results for $\lambda = 0.1$. Figure ?? shows the results for $\lambda = 1.0$. The effect of increasing λ is as expected: it decreases the magnitude of θ at the expense of performance.

Table ?? shows various statistics for this trajectory, with fixed initial and final conditions. Fixing the initial and final value imposes 10 constraints on the basis functions. The column labelled NPTS denotes the number of points in the input trajectory (and therefore also in the output trajectory). The pair NPOL refers to the number of polynomial basis functions for each of the flat outputs. The number of basis functions over which we optimize is NPOL - 10, due to the fized initial and final values. The column "Time" is the computation time in seconds. The column "cost" refers to the achieved value of the minimization criterion (??). The cost is linear in "NPTS", and therefore, "cost/NPTS" gives a better description of the cost. The last column indicates the minimizations method: a Powell direction set method, a conjugate gradient method or a variable metric method, as described in [?].

The first three rows of the table compare the different optimization methods. The conjugate gradient method seems to give a good balance between computation time and achieved minimum. Comparing the second block of three rows with the first block shows that increasing the number of basis functions gives only a small decrease in the cost function. The last block of four rows shows that the both the computation time and the cost are linear in NPTS. The cost per point does

not decrease much by using more points. This means that we can subsample the input trajectory without incurring a great penalty on the computed trajectory, while saving substantially on computation time. The computation times are still quite long. They are definitely too long for real time applications, but are still feasible for pseudo real time applications like the one in [?].

The same remarks for fixing the initial conditions hold as in the least squares approximation, except that we now have a nonlinear minimization problem with linear constraints. Using the same parametrization $A = A_0 + F^{\perp} A_1$, we minimize over A_1. The important observation is that the number of parameters in the optimization does not increase by imposing initial conditions.

NPTS	NPOL	Time ([s])	cost	cost/NPTS	Method
60	(13,13)	54.1	1.4		Powell
60	(13,13)	15.16	1.45	0.024	Conj Grad
60	(13,13)	32.36	1.43		Var Met
60	(15,15)	112.4	1.01		Powell
60	(15,15)	18.02	1.21		Conj Grad
60	(15,15)	44.33	0.45		Var Met
30	(13,13)	8.07	0.86	0.029	Conj Grad
60	(13,13)	15.16	1.45	0.024	Conj Grad
120	(13,13)	25.8	2.72	0.023	Conj Grad
300	(13,13)	66.4	6.58	0.022	Conj Grad

Table 3.3 Computation times for cost minimization with non-minimum phase outputs.

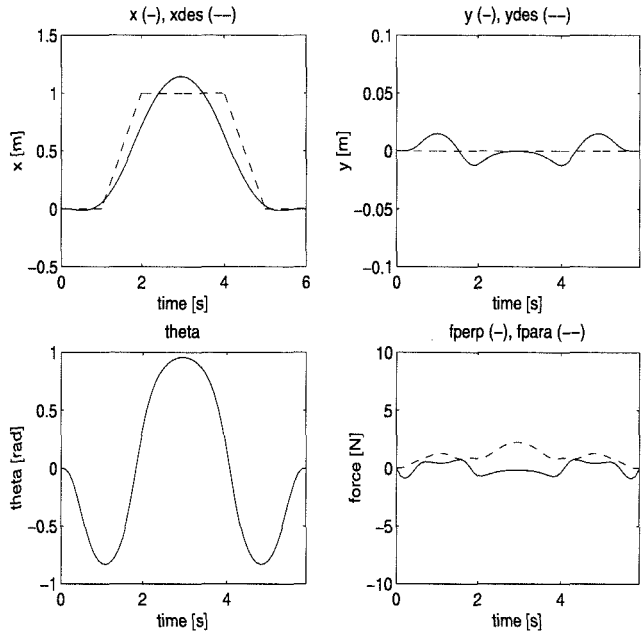


Figure 3.3 Trajectory for cost minimization with non-minimum phase outputs, $\lambda = 0.1$.

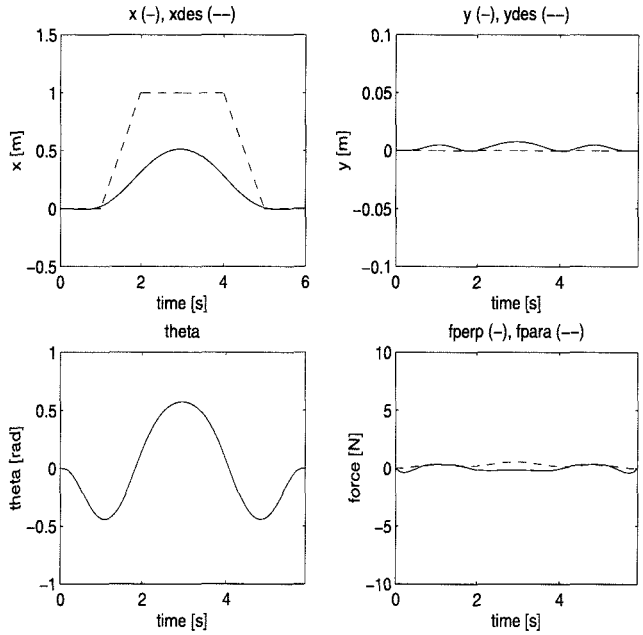


Figure 3.4 Trajectory for cost minimization with non-minimum phase outputs, $\lambda = 1.0$.

3.6 Experimental Data

In this section we present experimental data to validate the nonlinear control paradigm depicted in Figure 1.2. The data is taken with the Caltech ducted fan, described in Section 2.7. We compare a 1 degree of freedom design (Figure **??**), where only the desired output is fed forward, to a 2 degree of freedom design (Figure **??**), where we feed forward the entire state and input space trajectory. In both cases we use the same LQR controller to stabilize the system around the trajectory. This LQR controller was designed to stabilize the system around hover. We use the point-to-point steering technique from Section **??** repeatedly to compute the following trajectory for the approximate flat model of the ducted fan.

- Steer from $(0, 0)$ to $(1, 0)$ meter in 5 seconds.
- Stay at $(1, 0)$ and hover for 2 seconds.
- Steer from $(1, 0)$ to $(0, 0)$ meter in 5 seconds.
- Stay at $(0, 0)$ and hover for 3 seconds.

The two degree of freedom design gives a much more aggressive response, showing the validity of this approach. The reason is that the one degree of freedom design tries to keep the fan vertical while moving sideways. This is clearly not a feasible trajectory and therefore the performance degrades. Note the steady-state error in y caused by stiction in the stand.

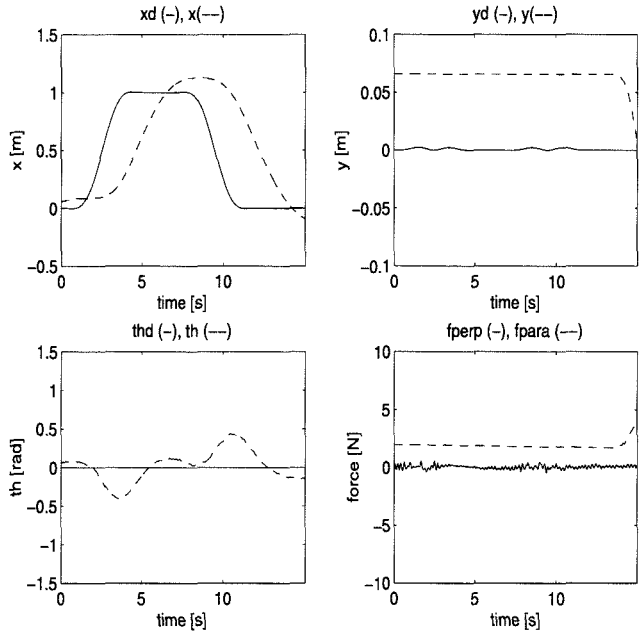


Figure 3.5 Experiment: one degree of freedom controller (no feedforward), repeated point-to-point steering.

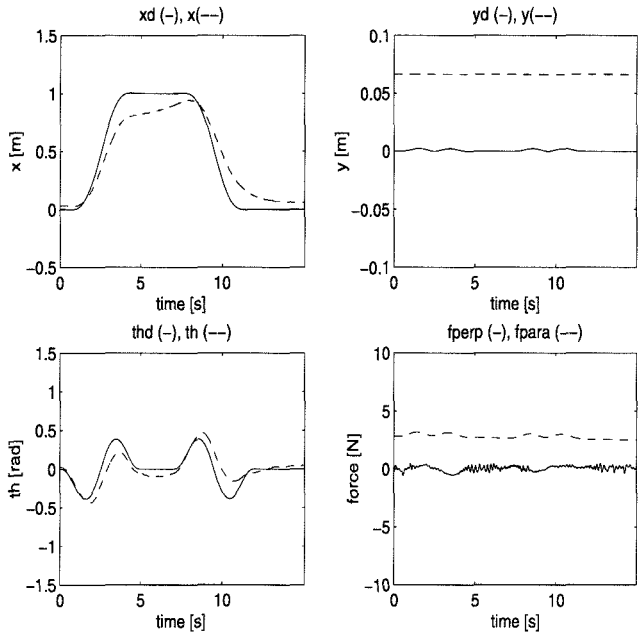


Figure 3.6 Experiment: two degree of freedom controller, repeated point-to-point steering.

3.7 Software

The software used to generate trajectories for the 3 problems in this chapter, and also to run the real time experiment in Section ?? is publicly available as a gzipped tar file through anonymous ftp from `avalon.caltech.edu:/pub/vannieuw/software/trajgen.tar.gz`. This file contains the libraries, examples and documentation. The routines are in ANSI C and will compile under different platforms. In particular, the simulations presented in this dissertation used the library compiled on a UNIX platform, and the real time experiments used the library compiled under MS-DOS. A Microsoft Windows graphics front end was also written to illustrate the speed of the routines. The numerical optimization routines are adapted from the Numerical Recipes Library developed by NAG [?], and a legal copy of these needs to be bought before using the library. The modification to the Numerical Recipes consist in improved memory management and graceful exiting. Matrices are reinitialized only when needed, and not at every call of the function. The maximum number of iterations can be given as a parameter to the function and upon reaching that number the routine will not dump out in the operating system but return with the best available answer so far. This is essential in real time optimization, since we only have finite time, and do not care about the best possible solution, but only about the best possible solution available within the limits of computation time. The real time software runs with Sparrow, a real time kernel for IBM PCs written at Caltech by Richard Murray and his group. Documentation on Sparrow can be found on the Web at URL `http://avalon.caltech.edu/~murray/sparrow`. Appendix ?? describes the functionality of the software and gives examples.

3.8 Summary and Conclusions

This chapter studied implementation issues related to a two degree of freedom nonlinear control paradigm. For differentially flat systems, we reviewed three different trajectory generation problems: point-to-point steering, least squares approximation, and cost minimization. These problems were evaluated on their computational cost. We presented approximate trajectory tracking for flat systems with zero dynamics as an alternative to stable system inversion and discussed its advantages and drawbacks. For all problems we presented sample trajectories, and for point-to-point steering we validated with experimental data. We briefly described a software library that solves the presented problems.

Chapter 4

Real-Time Trajectory Generation

4.1 Introduction

Whereas the previous chapter studied finite horizon problems with off-line calculations, the focus of this chapter will be real time trajectory generation for differentially flat systems. This is the problem of how to generate, possibly with some delay, a full state space and input trajectory in real time from an output trajectory that is given on-line, while allowing a tradeoff between stability and performance. We need the delay to ensure stability in the face of unstable zero dynamics. This chapter proposes two algorithms that solve the real time trajectory generation problem for differentially flat systems with (possibly non-minimum phase) zero dynamics, and analyze their convergence properties. The algorithms explicitly address the tradeoff between stability and performance in the form of a weighted cost criterion. The algorithms are validated in simulations and experiments with the vectored thrust ducted fan aircraft presented in Section 2.7. The work in this chapter appeared in abbreviated form in [?].

Looking back at the two degree of freedom control paradigm depicted in Figure 1.2, the trajectory can be generated off-line, in pseudo real time (i.e. at a rate a few orders of magnitude slower than the sampling rate), or in real time, depending on the particular problem. In the real time case, the trajectory is being updated at the same rate as new pilot input becomes available, with some delay due to computation time. If the system is non-minimum phase, the delay is also necessary to keep the internal dynamics bounded. A scheduled linear controller is used to correct for errors.

Related work is reported in [?, ?, ?], with the main difference that in those papers trajectory generation is performed essentially off-line. This requires the trajectory to be finite horizon and known in advance. Trajectory generation for non-minimum phase systems is discussed in [?, ?]. This approach results in anticausal trajectories and is therefore *a fortiori* off-line.

4.2 The Real-Time Trajectory Generation Problem

In this section we will try to come to a meaningful definition of the real-time trajectory generation problem, to motivate the algorithms presented in Section ??. First we need to distinguish between *trajectory tracking* which is the general problem, and *trajectory generation* which is one way to approach this problem.

The most straightforward approach to trajectory tracking is to subtract the plant output from the pilot output and feed this error signal to the controller. This is the so called "one degree of freedom" approach. We will show through simulations in Section ?? that this may lead to slow response times. The reason is that in one DOF design we are trying to track a drifting equilibrium configuration which is an unfeasible trajectory. Trajectory generation tries to remedy this problem by finding a feasible full state and input trajectory along which the system can be stabilized.

A further distinction can be made between off-line trajectory generation and on-line or real-time trajectory generation. In the first case, the trajectory is given to us ahead of time. In the second case, it becomes available as time proceeds. For minimum phase systems we can exactly follow a trajectory while maintaining internal stability. For non-minimum phase systems we need to introduce some sort of anticausality. For off-line trajectory generation a solution to this problem that allows exact tracking has been proposed in [?, ?]. An approximate solution for flat systems that allows a tradeoff between stability and performance, again for off-line trajectory generation, has been proposed in [?]. In this chapter we are concerned with on-line trajectory generation.

Some comments are in place about the qualifier "real-time" in "real-time trajectory generation." In daily parlance, real-time means "fast enough to be considered instantaneous." Of course, this depends on the time scale of the process under consideration, and we therefore specify "real-time" in this chapter as "computations being performed at the same rate as input becomes available from an operator." In the case of a pilot flying an airplane, this means the computations have to proceed faster than the time constant of the human motoric system, or in about 0.1 seconds.

Usually control objectives are stated as performance criteria subject to stability. For real-time trajectory generation we only have a finite time history of the desired trajectory available, and therefore stability as defined in an infinite time horizon does not make sense. Instead we can capture the notion of stability as some norm bound on the internal dynamics generated when following a desired trajectory. The "performance under stability" requirement then translates to minimizing a weighted norm between tracking error and magnitude of the internal dynamics. In agreement with $\mathcal{H}^\infty$ control theory we take this norm to be the L_2 norm on a finite time interval. This leads to the following cost to be minimized at each time instant:

$$\int_{t-T_d}^{t} (h(x) - y_d(s))^*(h(x) - y_d(s)) + \lambda K(x,u)ds \tag{4.1}$$

where K is an appropriate penalty on the internal dynamics, and T_d defines the time horizon, or the delay with which the trajectory is generated.

This formulation allows a tradeoff between performance and stability, as seen in

Chapter ??. We can increase stability at the expense of performance by increasing the penalty on the internal dynamics (i.e. λ). Since we have to minimize the cost in equation (??) at every time instant, we need to do this subject to fixed initial conditions, namely, the state that we happen to be at.

There are theoretical limits to the tracking performance that can be obtained in systems with unstable zero dynamics, as was shown in [?], and repeated in this dissertation in Theorem ?? in Appendix ??. Under mild conditions, a necessary condition for asymptotic tracking is that the system have stable zero dynamics. The authors prove this by constructing a signal that cannot be asymptotically tracked by non-minimum phase systems. An essential feature of this signal is that it has a time derivative with infinite support. One way to circumvent the non-minimum phase zero dynamics requirement is to restrict attention to asymptotic tracking of signals whose derivatives have finite support. More precisely, we define

Definition 4.1 (Eventually constant signals) The set of functions

$$S = \{y(t) \in \mathcal{L}_\infty(\mathbb{R}^m) | \; \exists \, t_s : \dot{y}(t) \equiv 0 \text{ for } t > t_s\}, \tag{4.2}$$

where t_s is not given in advance, is called the set of *eventually constant signals.*

Definition 4.2 (Asymptotic trajectory generation) We say an algorithm achieves *asymptotic trajectory generation* for a class of signals Y if the algorithm generates from $y_d \in Y$ a feasible full state and input trajectory (x_d, u_d) such that $\lim_{t\to\infty} h(x_d(t)) - y_d(t) = 0$ for all $y_d \in Y$.

Remark 4.3 The relevance of Theorem ?? for trajectory generation is born out by the fact that trajectory generation combined with a linear controller based on the Jacobi linearization of the plant will achieve asymptotic tracking of signals in $Y(\epsilon, N)$ for N large enough and ϵ small enough. ($Y(\epsilon, N) = \{y \mid \|y(t)\| \leq \epsilon, \ldots, \|y^{(N)}(t)\| \leq \epsilon, \forall t\}$, see Appendix ??) This follows from Lemma 4.5 in [?] and the fact that the higher order terms in the error system for $x - x_d$ are uniformly Lipschitz in time for desired signals in $Y(\epsilon, N)$. Hence asymptotically stable zero dynamics are also necessary for real-time trajectory generation, unless we relax the conditions of Theorem ?? somehow.

We require that our trajectory generation scheme achieve asymptotic trajectory generation for all signals in S. This comes down to requiring zero steady-state error.

Of course we need to make sure that eventually constant output signals lead to feasible state space trajectories. Hence the following assumption.

Assumption 4.4 We assume that to each value of the output y_d, there is an equilibrium value for the states and inputs, i.e. there exist (x_d, u_d) such that $y_d \equiv h(x_d)$, $f(x_d, u_d) \equiv 0$. We denote the mapping that maps each output value y_d to a full state and input space equilibrium by Eq, so that $f(Eq(y_d)) \equiv 0$, and $h(Eq(y_d)) = y_d$.

If this is not the case, we cannot maintain the output at the desired constant value. Based on the above discussion we propose to study the following problem:

Problem 4.5 (Real-time trajectory generation) Find an algorithm that calculates in real-time from $y_d(t)$ a feasible full state and input trajectory $(x_d(t), u_d(t))$ while allowing to trade off stability of the internal dynamics against tracking error, and such that

$$\lim_{t\to\infty} h(x_d(t), u_d(t)) - y_d(t) = 0 \tag{4.3}$$

for all $y_d \in S$.

One might object that this definition still allows the trajectory generation module to wait until the desired trajectory reaches its steady state value and then compute the trajectory off-line. We still would achieve asymptotic tracking. The key is that the time t_s after which $\dot{y}(t) \equiv 0$ is not given to us in advance, so that we cannot determine when to start the off-line computation. This point is philosophical though, since it should be clear that it is better to start acting when sufficient knowledge of the desired trajectory is available.

4.3 Two Algorithms For Trajectory Generation

We will now propose a solution to the above problem for differentially flat systems. First, we parametrize the flat outputs $z_i, i = 1 \ldots m$ by

$$z_i(t) = \zeta_i(x(t)) := A_{ij}\phi_j(t) \tag{4.4}$$

where the $\phi_j(t), j = 1 \ldots N$ are basis functions. This reduces the problem from finding a function in an infinite dimensional space to finding a finite set of parameters. At each time t we have available to us the desired output over the time interval $[\tau_0, \tau_f] := [t - T_d, t]$. Steering from an initial point in state space to a desired point in state space is trivial. We have to calculate the values of the flat outputs and their derivatives from the desired points in state space and then solve for the coefficients A_{ij} in the following system of equations:

$$\begin{array}{cc} z_i(\tau_0) = A_{ij}\phi_j(\tau_0) & z_i(\tau_f) = A_{ij}\phi_j(\tau_f) \\ \vdots & \vdots \\ z_i^{(l)}(\tau_0) = A_{ij}\phi_j^{(l)}(\tau_0) & z_i^{(l)}(\tau_f) = A_{ij}\phi_j^{(l)}(\tau_f). \end{array} \tag{4.5}$$

To streamline notation we write the following expressions for the case of *one* flat output only. The multi-output case follows by repeatedly applying the singe output case, since the algorithm decouples in the flat outputs. Let $\Phi(t)$ be the $l+1$ by N matrix $\Phi_{ij}(t) = \phi_j^{(i)}(t)$ and let

$$\begin{aligned} \bar{z}_0 &= (z_1(\tau_0), \ldots, z_1^{(l)}(\tau_0)) \\ \bar{z}_f &= (z_1(\tau_f), \ldots, z_1^{(l)}(\tau_f)) \\ \bar{z} &= (\bar{z}_0, \bar{z}_f). \end{aligned} \tag{4.6}$$

Then the constraint in equation (??) can be written as

$$\bar{z} = \begin{pmatrix} \Phi(\tau_0) \\ \Phi(\tau_f) \end{pmatrix} A =: \Phi A. \tag{4.7}$$

That is, we require the coefficients A to be in the plane defined by equation (??). The only condition on the basis functions is that Φ is full rank, in order for (??) to have a solution. We can solve these equations at each sample instant to generate a trajectory from the current state to the desired output a certain time T_d later. We augment this desired output to a desired full state and input by mapping it onto an equilibrium with the mapping Eq. On this trajectory we pick a state corresponding to some time $\tau \in [\tau_0, \tau_f]$ and use this as the instantaneous desired state for the linear controller. This leads to the first algorithm:

Algorithm 1 *Given: the delay time T_d, the current flat flag $\bar{z}_0$, the desired output y_d. At each sampling instant t_k:*

1. *Let $\tau_f = t_k$, $\tau_0 = t_k - T_d$, $\bar{z}_f = \bar{\zeta}(Eq(y_d(t_k)))$.*
2. *Compute a trajectory of the flat outputs by solving $\bar{z}_0 = \Phi(\tau_0)A$, $\bar{z}_f = \Phi(\tau_f)A$ for A.*
3. *Compute a point on that trajectory with $\bar{z}_1(\tau) = \Phi(\tau)A$ where $\tau \in [\tau_0, \tau_f]$.*
4. *Solve for $(x_1(\tau), u_1(\tau))$ from $\bar{z}_1(\tau)$.*
5. *$(x_1(\tau), u_1(\tau))$ is the next desired state and input to feed forward at time t_k.*

The times τ_* are "virtual" times within the algorithm that shift along as physical time proceeds. They are reassigned with every new sample time. The times t_* are physical times. This algorithm steers us from the current position to an equilibrium state with the desired values for the outputs. We generate a trajectory over the time interval $[t_k - T_d, t_k]$, and pick a time τ and corresponding point (x_1, u_1) on this trajectory. This will be the desired state to steer to. We repeat this process at every sampling instant. This is illustrated in Figure ??. The solid line is the pilot input, the dashed line is the generated trajectory at sampling time τ_0, the dotted line is the trajectory generated one period later.

This algorithm does not involve the explicit minimization of a cost function to trade off stability versus performance. We will show through simulations in Section ?? that the parameter T_d regulates this tradeoff. Increasing T_d will increase stability at the expense of performance.

We can bypass solving for the coefficients A_{ij} in the matrix A by noting that

$$\bar{z}_1 = \Phi(\tau)\Phi^{-1}\bar{z} =: F(\tau)\bar{z}_0 + G(\tau)\bar{z}_f. \tag{4.8}$$

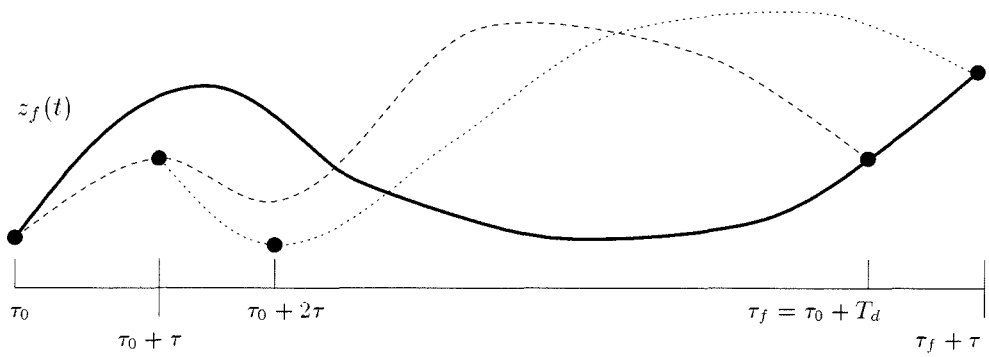


Figure 4.1 Algorithm for real-time trajectory generation.

If we execute this scheme every sample instant we get a dynamical equation for $\bar{z}_1 =: \bar{z}_{k+1}$ for each $\bar{z}_0 =: \bar{z}_k$, namely:

$$\bar{z}_{k+1} = F(\tau)\bar{z}_k + G(\tau)\bar{z}_f(k) \tag{4.9}$$

which has the desired output $\bar{z}_f(k) = \zeta(Eq(y_d(t_k)))$ at time instant k for its input.

Theorem 4.6 *There is a $\tau \in [\tau_0, \tau_f]$ such that Algorithm ?? achieves real-time asymptotic trajectory generation of all desired outputs in S.*

Proof: We will show that $F(\tau)$ is stable for appropriate choice of τ, and then that the steady state error is zero for $y_d \in S$. Since we constructed the $F(\tau)$, $G(\tau)$ to steer us from $\bar{z}_0$ to $\bar{z}_f$, it follows that $G(\tau_f) = 0$ and $F(\tau_f) = I$. So for $\tau = \tau_f$ all eigenvalues of $F(\tau)$ are at the origin. Since the eigenvalues of $F(\tau)$ are continuous functions of τ, there exists a $\tau \in [\tau_0, \tau_f]$ such that the eigenvalues of $F(\tau)$ are in the open unit circle. Now $y_d \in S$ means that there is a k_s such that $\bar{z}_f(k)$ is a constant, say $\bar{z}_f$, for all $k > k_s$. Therefore $\bar{z}_k$ converges to a constant value, say $\bar{z}_\infty$ which will be a multiple of $\bar{z}_f$ due to linearity of (??). So $\bar{z}_\infty = \gamma \bar{z}_f$, where γ depends only on F and G. Since there is a trajectory from $\bar{z}_\infty$ to $\bar{z}_f$ that will bring us closer to $\bar{z}_f$ for appropriate choice of τ, we have $\gamma = 1$ for that value of τ. Then we have $\lim_{k\to\infty} \bar{z}_k = \bar{z}_f$. ■

Picking $T_d = t_k - t_{k-1}$ in the above scheme corresponds to a one step deadbeat controller. This requires large control signals which might saturate the actuators. Clearly, there is a tradeoff between performance and control effort and bandwidth. Note that the matrices $F(\tau)$ and $G(\tau)$ are fixed once τ is selected, and can be computed ahead of time. We should mention that it is not hard to find τ such that $F(\tau)$ is stable. In fact, it requires considerable effort to construct a set of basis functions and a τ such that $F(\tau)$ is unstable. For polynomial basis functions any $\tau \in]\tau_0, \tau_f[$ will do. This follows from the fact that the degree of a polynomial is an upper bound on the number of its zeros.

Step 2 in Algorithm ?? computes a trajectory between the flat flags $\bar{z}_0$ and $\bar{z}_f$ by using the point-to-point steering algorithm of Chapter ??. In fact, we can use

any of the trajectory generation problems presented in that Chapter to generate the trajectory. It just so happens that the point-to-point steering problem is particularly attractive since it results in a linear update for the flat flag, as in Equation (??). In particular, we can augment this algorithm with an additional minimization that allows tradeoff between stability and performance as discussed in Sections ?? and ??. The cost criterion takes the form:

$$J = \min_{A} \int_{\tau_0}^{\tau_f} (y(A,s) - y_d(s))^*(y(A,s) - y_d(s)) + \lambda K(A,s)ds \tag{4.10}$$

subject to $\bar{z}_0 = \Phi(\tau_0)A, \bar{z}_f = \Phi(\tau_f)A$. Here y is the tracking output, and y_d the desired tracking output. K is a function that bounds the internal dynamics. We can perform this minimization by finding a particular solution that satisfies the initial and final constraints: $A_0 = \Phi^\dagger \bar{z}$, and parametrizing the general solution as $A = A_0 + \Phi^\perp A_1$ where $\Phi^\perp$ is a basis for the nullspace of Φ. This optimization problem is in general nonlinear and nonconvex. We therefore have to resort to an iterative scheme. Since the optimization has to be performed in real-time, we might not be able complete the minimization procedure and have to preempt the procedure. We will show that this will not result in loss of convergence. This leads to the following algorithm:

Algorithm 2 *Given: the delay time T_d, the current flat flag $\bar{z}_0$, the desired output y_d. At each sampling instant t_k:*

1. *Let $\tau_f = t_k$, $\tau_0 = t_k - T_d$, $\bar{z}_f = \bar{\zeta}(Eq(y(t_k)))$.*
2. *Compute a trajectory for the flat outputs by finding a particular solution A_0 to $\bar{z}_0 = \Phi(\tau_0)A$, $\bar{z}_f = \Phi(\tau_f)A$.*
3. *Optimize A_1 to minimize J in equation (??).*
4. *Let $A = A_0 + N^\perp A_1$.*
5. *Compute a point on the nominal trajectory with $\bar{z}_1(\tau) = \Phi(\tau)A$ where $\tau \in [\tau_0, \tau_f]$.*
6. *Solve for $(x_1(\tau), u_1(\tau))$ from $\bar{z}_1(\tau)$.*
7. *$(x_1(\tau), u_1(\tau))$ is the next desired state and input to feed forward at time t_k.*

Note that the optimization over A_1 can be preempted if computation time runs out.

Theorem 4.7 *There is a $\tau \in [\tau_0, \tau_f]$ such that Algorithm ?? achieves real-time asymptotic trajectory generation of all desired outputs in S.*

Proof: We will show that $\bar{z}_1$ converges to a constant value for constant $\bar{z}_f$, and then that this constant value equals $\bar{z}_f$. Even though we cannot dispense with the computation of the coefficients A as we could in Algorithm ??, we know that for

$\tau = \tau_f$ the algorithm steers to the desired output in one step. Regardless of the values for A_1, from continuity of $\bar{z}_1 = \Phi(\tau)A$ in τ, we can find a τ such that

$$\|\bar{z}_1 - \bar{z}_f(k)\| < \|\bar{z}_0 - \bar{z}_f(k)\| \tag{4.11}$$

so that if $\bar{z}_f(k)$ is a constant for $k \geq k_s$, say $\bar{z}_f$, we achieve convergence to a constant value for $\bar{z}_k$. Similar to the proof of Proposition ?? we can show that this constant value has to be $\bar{z}_f$. ■

It might seem curious at first sight that convergence of Algorithm ?? does not depend on the cost criterion J. On second thought this is quite advantageous since we cannot guarantee that the optimization of J converges in the allotted computation time. Preemption of the minimization will not result in loss of convergence. The additional optimization allows us to get better performance (in the sense of a lower cost criterion J) if the computation time allows it. If no improvement can be obtained, the algorithm returns the solution of the point-to-point steering algorithm ??. This is essential for convergence.

It is somewhat unsatisfactory that we have to fix the final conditions in the above algorithms. For nonlinear systems we have in general multiple equilibria corresponding to the same output values, most of which are undesirable. Without fixing the final condition we cannot guarantee that the trajectory will converge to the desired equilibrium, even though we still get asymptotic tracking for signals in S. Indeed, simulations showed that the trajectory might end up in an undesired equilibrium.

At first sight, Algorithm ?? seems to exhibit some similarity to Model Predictive Control (MPC), as advocated in [?]. In MPC, a finite or infinite horizon cost criterion on the states and inputs is minimized over the inputs at each sampling time, and the first input of the optimal sequence is applied. This process is repeated at each sampling time. The cost criterion contains predictions of future states, based on the current state and a model, hence the name MPC. MPC does not, however, address the issue of generating a feasible full state and input nominal trajectory, but assumes these are given, so that the cost criterion penalizes the deviation of the predicted trajectory from the nominal trajectory, and the computed optimal input is added to the nominal input. MPC seems to have been most successful in chemical process control, where the desired trajectories are set-point changes, and the slow time constants accommodate the intensive computations.

4.4 Simulations

In this section, simulations are performed with an approximate flat model of the Caltech ducted fan, presented in Section 2.7. The code used to generate the real-time trajectories for the simulations in this section and the experiments in the next section is added as a module to the trajectory generation library described in Appendix ??. and is available through anonymous ftp from `avalon.caltech.edu:/pub/vannieuw/software/trajgen.tar.gz`.

We simulate on-line pilot input as a file from which successive samples are read every sample instant. The pilot command is input to a trajectory generation module, whose output serves as a nominal trajectory. We wrap a simple full state LQR controller around a nonlinear model of the fan. This controller was designed to stabilize hover. See [?] for a detailed analysis of several controller designs for this experiment. We assume we have knowledge of the full state. On the experiment this is achieved by differentiating and filtering the position signals.

The nominal trajectories are generated with the approximate flat model for the fan presented in 2.7. The nonlinear model used for simulation takes into account the aerodynamic drag, inertial effects from the rotating propeller, changing inertias with altitude, and viscous friction. This model is more elaborate than the flat approximation used to generate the nominal trajectories, but is no longer flat. We use this more detailed model to allow comparison with the experimental results in the next section.

First we show how the ducted fan behaves without feed forward. The pilot input is a 1.0 meter step in the x direction at constant altitude. The pilot input is used to generate an error signal to the controller. At each time instant we stabilize around the equilibrium point generated by setting x and y equal to the pilot input, and all other states equal to 0. This is the conventional "one degree of freedom" controller. Figure ?? shows that the trajectory followed by the fan lags far behind the desired trajectory.

In this plot and subsequent plots, the pilot input is denoted by (xp, yp), the generated desired trajectory (which is identical to the pilot input in the one degree of freedom case) is denoted (xd, yd), whereas the variable name without suffix denotes the real (experimental or simulated) time trace of a quantity. The force parallel to the fan shroud is denoted "fpara," the force perpendicular to the fan shroud is called "fperp."

Next we show for Algorithm ?? plots of the pilot input, the generated trajectory, the simulated trajectory for the (x, y) position of the fan, as well as the generated and simulated trajectory for θ and the nominal forces. Figure ?? shows these for a delay time of $T_d = 60$ sampling periods of $T_s = 0.01$ seconds. We see that the fan follows the pilot input much better than in the one degree of freedom design. Clearly, there is an advantage in real-time trajectory generation. Figure ?? shows these for a delay time of $T_d = 100$ sampling periods. It is clear that the larger delay results in better stability, i.e. lower magnitude of θ and the nominal forces, but poorer performance, since the delay is bigger.

Figures ?? and ?? both show a large error in θ right after the first and second peak of the nominal θ trace. We suspect this is caused by the inertia changing with altitude, which is not taken into account in the flat model, but is present in the simulation model. Note that the errors occur simultaneously with a substantial error in altitude y.

We tested Algorithm ?? in simulation. It behaves as expected, in the sense that it penalizes the cost. The major problem is that the optimization is a factor of 10 too slow for realistic operator sampling rates. Improvement of this optimization is a subject of current research.

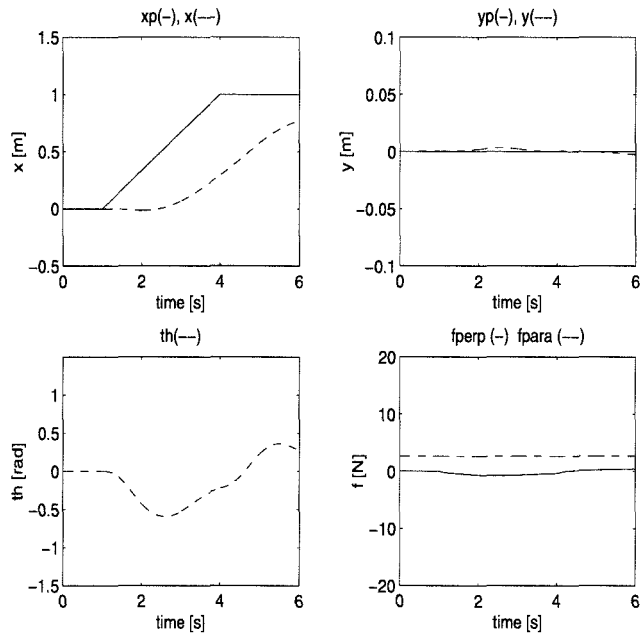


Figure 4.2 Simulation: one degree of freedom controller. Step in x of 1 meter.

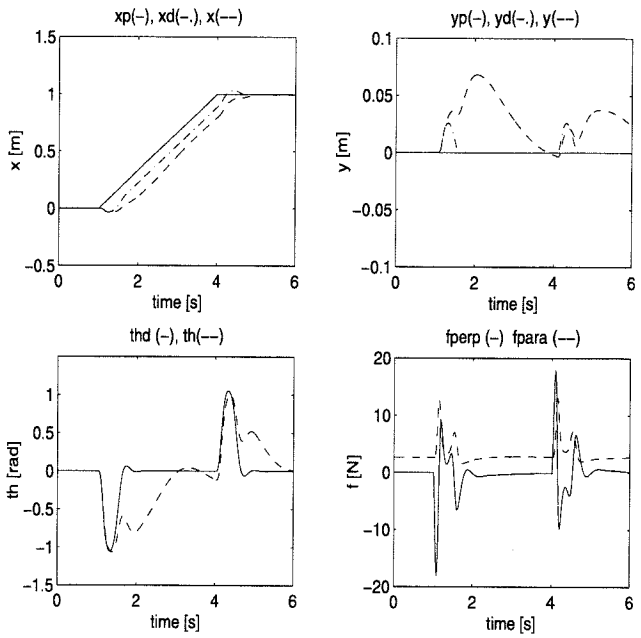


Figure 4.3 Simulation: algorithm ??, $T_d = 60 \times T_s = 0.6$ s. Step in x of 1 meter.

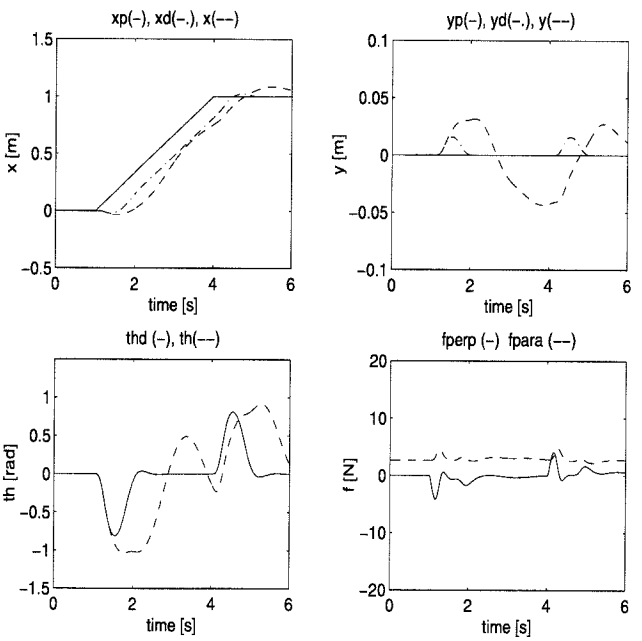


Figure 4.4 Simulation: algorithm ??, $T_d = 100 \times T_s = 1.0$ s. Step in x of 1 meter.

4.5 Experiments

Algorithm ?? was implemented on the experimental apparatus. The pilot input comes from a joystick with two degrees of freedom. We run the trajectory generation algorithm at 100 Hz, and the controller at 200 Hz. The delay time $T_s = 1.0$ seconds, corresponding to 100 samples for the trajectory generation algorithms.

We conducted 2 experiments. The first one was the one degree of freedom controller: the pilot signal was used to generate an error signal in the output around which the fan was stabilized. The results are depicted in Figure ??. The generated desired trajectory, is identical to the pilot input in this case.

In the second one, we used the pilot signal to generate a trajectory, as described in this chapter. The results are depicted in Figure ??. The desired trajectory (xd, yd) is generated by the trajectory generation module and is no longer equal to the pilot input (xp, yp). Since the pilot input is given real-time the experiments are not repeatable. We can draw some qualitative conclusions though.

The real-time trajectory generation algorithm gives a more aggressive response, at the expense of more oscillations in the pitch angle θ. This could likely be remedied by implementing Algorithm ??. Even so, the mean squared pitch error for the real-time trajectory generation is less than for the one degree of freedom controller. The real-time implementation performs worse than the simulations. We submit that this is due to noise in the pilot input and plant uncertainty. The pilot input from the joystick has a dead zone of about 5 ticks on a total range of 250. The flat model used to generate trajectories is of course only an approximation of the real dynamics. Also, it can be seen from the plots that the required actuator bandwidth is high. Limiting the nominal actuator bandwidth can be included as part of the cost criterion by weighting basis functions with high frequency content different than basis functions with low frequency content.

Due to the high computational requirements of the second algorithm, it is not tested in the experiments.

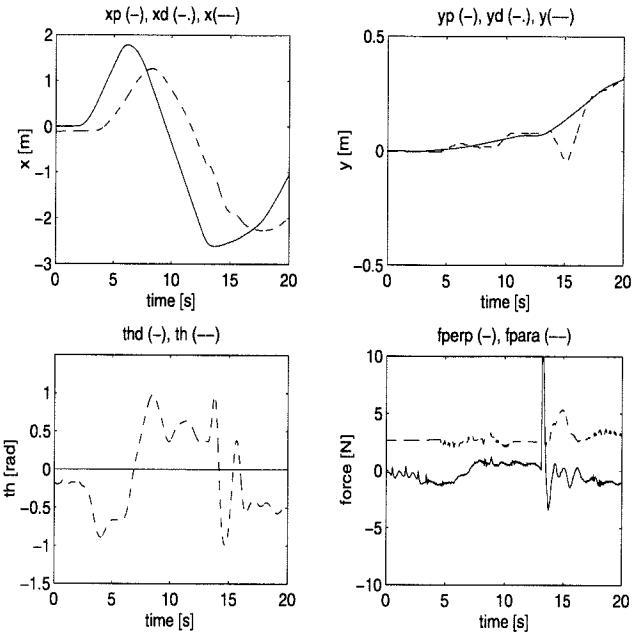


Figure 4.5 Experimental data: one degree of freedom controller.

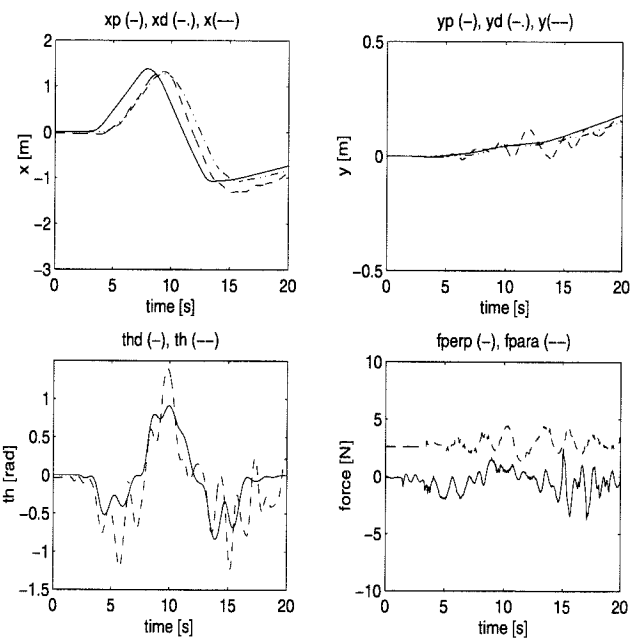


Figure 4.6 Experimental data: real-time trajectory generation.

4.6 Conclusions

In this chapter, we proposed a formulation for the real-time trajectory generation problem. We described two algorithms for real-time trajectory generation for differentially flat systems with unstable zero dynamics, and proved stability and convergence properties. The first algorithm generated a trajectory that steers from the current position to a desired final position given by the pilot input. We can trade off stability versus performance by varying the delay time. The second algorithm steers to a desired final position while minimizing a cost criterion, that typically limits the magnitude of the zero dynamics. The algorithms were validated with simulations and experiments.

Chapter 5

Perturbations to Flatness: Mode Switching

5.1 Introduction

In the previous chapters we dealt with differentially flat systems. Most systems are not differentially flat, and in this chapter we investigate some extensions that make the tools from differential flatness applicable to non-flat systems. The approach we take here is to find a *flat approximation*. Of course, any system can be approximated by a flat system, but for this procedure to be meaningful, the flat approximation has to be close in some sense. This requires a meaningful metric on systems, and at this point it is unclear what a good flat approximation is. Some work in this direction is reported in [?, ?], which uses ideas based on Frobenius' theorem. In some cases, the flat approximation is natural, in the sense that the system is the sum of a flat system plus perturbing terms. This happens to be the case for the ducted fan, described in Section 2.7, if we include the aerodynamic forces. These forces only become significant at high velocities. In order to investigate the effect of aerodynamic perturbation terms, a new fan was built with a wing and an aerodynamically shaped shroud, see Figure ??. In forward flight, the wing carries the weight of the fan, and to take this effect into account, one clearly has to consider the aerodynamic forces. This chapter present some methods to accommodate perturbations to flatness, guided by the problem of mode switching.

Commercial and military aircraft operate in different modes during flight, corresponding to flight regime, controlled variables and used actuators. In a landing mode, for example, the controlled variables would be the position of the aircraft with respect to the runway. In a climbing mode the primary controlled variable is altitude. In other modes we are concerned with velocity rather than position. Different modes exhibit vastly different aerodynamic properties.

In high performance aircraft, it is important that mode switches occur fast. This can be accomplished by calculating a nominal trajectory that brings the aircraft from one mode to the other. The nominal trajectory gives a more aggressive response than point stabilization around a changing output, while reducing control effort. For aggressive maneuvering we should exploit the nonlinearities of the aircraft to obtain better performance. We use differential flatness of the approximate model of the pitch dynamics of the ducted fan to achieve fast switching between those modes.

First we will present the extended model of the thrust vectored aircraft, that includes aerodynamic forces. This repeats some of the material in Section 2.7, but we feel it adds to the clarity to present the model in its totality. We will also adopt the notation that is customary in aerodynamics to describe the various aerodynamic effects. Then we will discuss some issues involved in mode switching. The central part of the chapter is a discussion of methods to deal with perturbations to flatness, based on Lyapunoff arguments. Simulations and experimental data are provided to validate the approach. This chapter is based on [?].

5.2 Model: General Pitch Dynamics

In this section we will present the pitch dynamics corresponding to various flight modes. Consider the thrust vectored aircraft with wing depicted in Figures ?? and ??.

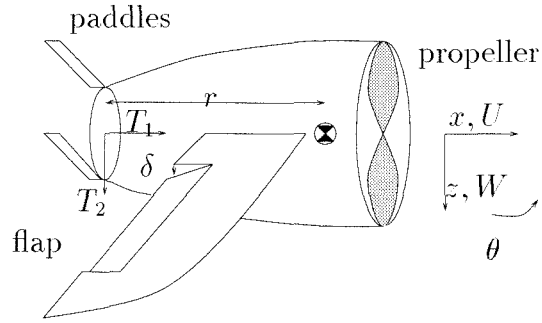


Figure 5.1 Thrust vectored aircraft with wing.

For hover to hover transitions we want to control the position of the aircraft with respect to a fixed spatial frame. Hover to hover transitions typically happen at low speed. We therefore ignore the aerodynamic effects of drag and lift. The pitch dynamics in spatial coordinates are then

$$\begin{pmatrix} m\ddot{x} \\ m\ddot{z} \\ J\ddot{\theta} \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \\ 0 & r \end{pmatrix} \begin{pmatrix} T_1 \\ T_2 \end{pmatrix} + \begin{pmatrix} 0 \\ mg \\ 0 \end{pmatrix}. \tag{5.1}$$

The inputs T_1 and T_2 are the axial and perpendicular components of the thrust. Typically T_1 is much larger than T_2. The pitch angle θ is measured with respect to the horizontal. It will be convenient to have a symbol , Θ, for the angle with the vertical, measured positive in the same direction as the pitch angle.

It was shown by Martin et al. in [?] that if we ignore the aerodynamic terms, the system (??) becomes *flat*, with flat outputs given by the coordinates of the *center*

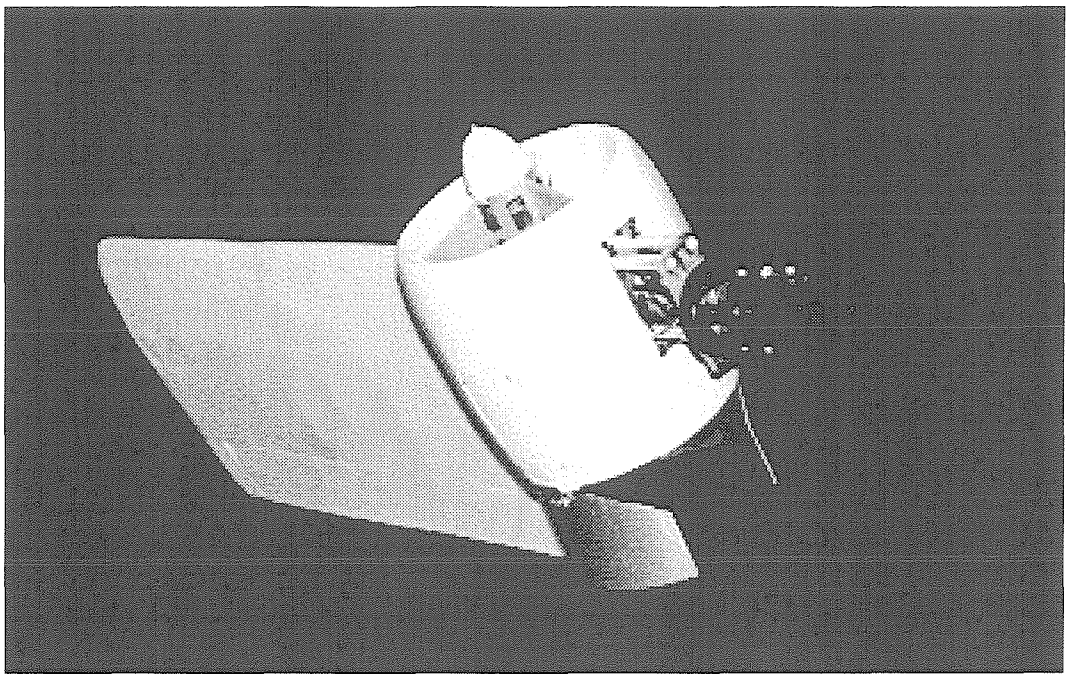


Figure 5.2 Second generation Caltech ducted fan with wing.

of oscillation:

$$\begin{aligned} x_f &= x + \frac{J}{mr}\cos\theta \\ z_f &= z - \frac{J}{mr}\sin\theta. \end{aligned} \tag{5.2}$$

Flatness means that all states and inputs of the system can be expressed as functions of the flat outputs and their derivatives. In particular, in the above system,

$$\begin{aligned} \ddot{x}_f &= \ddot{x} + \frac{J}{mr}(-\sin\theta\,\ddot{\theta} - \cos\theta\,\dot{\theta}^2) \\ \ddot{z}_f &= \ddot{z} + \frac{J}{mr}(-\cos\theta\,\ddot{\theta} + \sin\theta\,\dot{\theta}^2) \end{aligned} \tag{5.3}$$

so that

$$\tan\theta = \frac{-\ddot{z}_f + g}{\ddot{x}_f}. \tag{5.4}$$

This equation gives us θ, and from θ and (x_f, z_f) we can find the other states. We can resolve the modulo π redundancy in the tangent function by requiring that the nose is always pointing forward.

For flat systems, trajectory planning becomes trivial: we can design a trajectory in the lower dimensional output space and lift it to the full state and input space.

For mode switching, we are interested in steering from an initial condition to a final condition. This gives prescribed values for the flat outputs and their derivatives at the initial and final time, which we can then link by an arbitrary curve in output space. Typically, one parametrizes the curves by basis functions, and then solves for the coefficients to match the initial and final conditions. See Chapter ?? for a more detailed treatment of trajectory planning for flat systems.

For forward flight, we ignore the position variables, using their velocities instead. The pitch dynamics are more conveniently given in body coordinates, since the aerodynamic forces are most conveniently written in body coordinates:

$$\begin{pmatrix} m\dot{U} \\ m\dot{W} \\ J\dot{Q} \end{pmatrix} = \begin{pmatrix} T_1 \\ T_2 \\ rT_2 \end{pmatrix} + mg \begin{pmatrix} -\sin\theta \\ \cos\theta \\ 0 \end{pmatrix} + m \begin{pmatrix} -WQ \\ UQ \\ 0 \end{pmatrix} \tag{5.5}$$

where $Q = \dot{\theta}$ is the pitch rate, U is the forward velocity and W is the downward velocity (measured positive downward). In these equations, we still ignore aerodynamic coefficients.

This system is also flat, with flat outputs given by the velocity of the center of oscillation, but written in quantities occurring in Equation (??):

$$\begin{aligned} \begin{pmatrix} \dot{x}_f \\ \dot{z}_f \end{pmatrix} &= \begin{pmatrix} \dot{x} - \frac{J}{mr}\dot{\theta}\sin\theta \\ \dot{z} - \frac{J}{mr}\dot{\theta}\cos\theta \end{pmatrix} \\ &= \begin{pmatrix} U\cos\theta + W\sin\theta - \frac{J}{mr}Q\sin\theta \\ -U\sin\theta + W\cos\theta - \frac{J}{mr}Q\cos\theta \end{pmatrix}. \end{aligned} \tag{5.6}$$

Note that this is *not* the velocity of the center of oscillation in body coordinates. The system is not flat with respect to these latter outputs. Taking derivatives of (??),

$$\begin{pmatrix} \ddot{x}_f \\ \ddot{z}_f \end{pmatrix} = \begin{pmatrix} (-\frac{J}{mr}Q^2 + \frac{T_1}{m})\cos\theta \\ g + (\frac{J}{mr}Q^2 - \frac{T_1}{m})\sin\theta \end{pmatrix}, \tag{5.7}$$

and again

$$\tan\theta = \frac{-\ddot{z}_f + g}{\ddot{x}_f}, \tag{5.8}$$

from which we can recover θ, and hence Q, U, W, T_1 and T_2. There is a sign ambiguity in θ.

Compare the system (??) with the full pitch dynamics of an aircraft with aero-

dynamic surfaces [?]:

$$\begin{pmatrix} m\dot{U} \\ m\dot{W} \\ J\dot{Q} \end{pmatrix} = \begin{pmatrix} T_1 \\ T_2 \\ rT_2 \end{pmatrix} + mg \begin{pmatrix} -\sin\theta \\ \cos\theta \\ 0 \end{pmatrix} + m \begin{pmatrix} -WQ \\ UQ \\ 0 \end{pmatrix} + \frac{1}{2}\rho V^2 S \begin{pmatrix} C_x(\alpha, \delta_e) \\ C_y(\alpha, \delta_e) \\ \bar{c}C_\theta(\alpha, \delta_e) \end{pmatrix}, \tag{5.9}$$

where $V^2 = U^2 + W^2$ is the square of the absolute velocity, ρ is the density of air, S is the wing surface, $\bar{c}$ is the mean aerodynamic chord, δ_e is the flap angle of the control surface, in our case an elevator, and $\alpha = \arctan\frac{-W}{U}$ is the angle of attack. The functions C_x, C_y, C_θ represent aerodynamic lift, drag and moment.

It may seem from the equations (??) that we are dealing with a fully actuated system. However, the control surface is only effective in a narrow range around zero angle of attack, and for fast maneuvering we are interested in regimes with high angle of attack. We can use the flap to trim and control the equilibrium in forward flight, but not to switch modes.

It may be that we are not interested in the x-position, but only in the forward velocity, and also position in the spatial z-coordinate, as in constant altitude flight. Then we can use the x velocity and the z position of of the center of oscillation as our flat outputs. Similarly in the unlikely case that we want to regulate vertical velocity and horizontal position.

In this chapter we propose to calculate nominal trajectories for the flat system (??) and use these as nominal trajectories for the non-flat system (??). We investigate several straightforward extensions to flatness to deal with the aerodynamic terms that perturb flatness.

5.3 Model: Aerodynamic Forces from Wind Tunnel Data

In this section we model the thrust vectored aircraft that was designed and built in our lab, see Figure ??. To allow unlimited travel in the horizontal direction, the aircraft is mounted on a horizontal boom that rotates around a central post. There are 3 main parts to the modelling. The horizontal boom, the shroud containing a propeller driven by an Astroflight Cobalt 40 electric motor, and the wing. Due to the rotation, all parts move at a different horizontal velocity. The shroud is attached to the boom in its center of mass.

The gravitational mass of the fan is offset by a counterweight in the center of the central post, to allow the 200 W motor to lift the weight of the fan and the boom in hover. To reduce the inertial mass in the vertical direction, this counter weight is attached through a pulley with gear ratio 1:5. This results in different gravitational masses and inertial masses in the x and z direction respectively: $m_g = 0.46$ kg, $m_x = 4.9$ kg, $m_z = 8.5$ kg. It was shown in Section 2.7 that using different masses does not affect flatness of the system (??). The values of the remaining parameters in Equation (??) are $r = 0.12$ m, $J = 0.0323$ kg m^2 and $g = 9.8$ m/s^2.

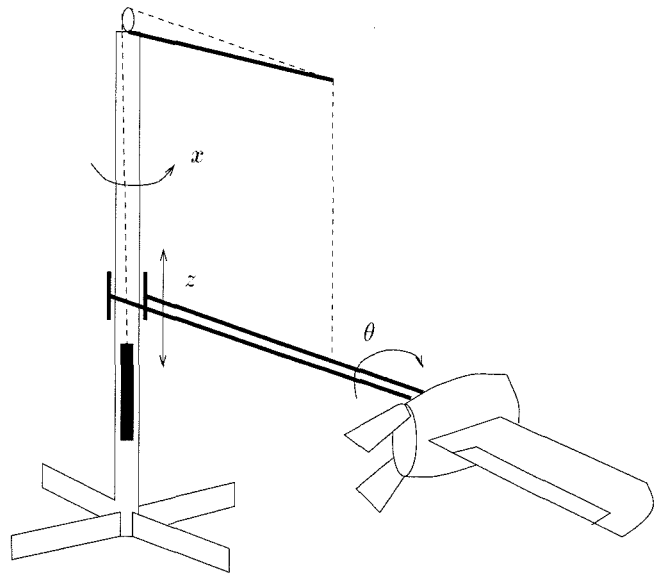


Figure 5.3 Thrust vectored aircraft on stand.

The boom is modelled as a cylinder, providing drag only and no lift and moment. The drag is

$$D_b = 0.5 C_d D \rho \omega^2 \int_0^{l_b} r^2 dr \tag{5.10}$$

where $C_d = 1.2$ is the drag coefficient of a cylinder in uniform flow, $\rho = 1.2247$ kg/m^3 is the density of air at sea level, $l_b = 1.7524$ m is the length of the boom, $D = 0.0635$ m is the diameter of the boom, $\omega = v_x / l_f$ is the angular velocity of the boom, v_x is the horizontal velocity of the fan, and $l_f = 2.032$ m is the distance from the central post to the center of the shroud.

The wing is a NACA 0015 airfoil [?]. This airfoil is symmetric, since we want to fly in both positive and negative direction. The wing was put in the wind tunnel on a force-torque sensor to measure the forces. The experimental data for lift, drag and moment agrees well with the theoretical values for lift, drag and moment. In particular, the aerodynamic center is at the quarter chord point for low angle of attack and over the range of wind speeds we are interested in, $0 \leq v_x \leq 12$ m/s. Since different cross sections of the wing move at different speed, we find for the total lift of the wing

$$L_w = 0.5 C_L c \rho \omega^2 \int_{l_{w0}}^{l_{w1}} r^2 dr \tag{5.11}$$

where $l_{w0} = 2.15$ m is the distance from the center to the inside edge of the wing, $l_{w1} = 2.81$ m is the distance from the center to the outside edge of the wing, $c = 0.35$ m is the aerodynamic chord, and C_L is the lift coefficient determined experimentally.

Note that C_L is linear in the angle of attack, α, for α smaller than the stall angle of attack, which is about 15 degrees for the NACA 0015. Similar expressions hold for the drag and the moment on the wing. The wing has an elevator hinged at the 3/4 chord point, with a range of $-60 \leq \delta_e \leq 60$ degrees. The lift is also linear in the elevator deflection for small angles of attack.

The shroud is most difficult to model. Fundamental aerodynamic theory does not provide expressions for lift, drag and moment coefficients. We have to rely completely on experimental data. The lift, drag and moment are highly dependent on fan speed, wind speed, angle of attack and flap angle. We approximated the lift, drag and moment by a 4th order polynomial in the independent variables, restricting terms to be linear in wind speed and cubic in α. Including all cross terms, and pruning terms with small coefficients, this results in 50 terms.

The boom and wing forces are transformed to spatial axes and the horizontal component is scaled so as to act on the center of the shroud by factors l_f/l_{b2} and l_f/l_{w2} respectively. Here $l_{b2} = 0.88$ m is the distance from the center of the stand to the center of the boom, and $l_{w2} = 2.52$ m is the distance to the center of the wing.

Since we have aerodynamic data only for wind speeds $3 \leq V \leq 12$ m/s, we interpolate the above aerodynamic model with an analytic model around hover given by Equation (??) with the different inertial and gravitational masses m_x, m_z, m_g substituted.

Figure ?? shows the lift, drag and moment as a function of airspeed and angle of attack, for $\delta_e = 0$, horizontal flight, zero paddle deflection, and zero thrust. Note the interesting sign reversal of the moment coefficient beyond the stall angle of attack. In this regime, the pitch moment dynamics are unstable. For small angles of attack, the center of pressure is aft of the center of mass, by a distance of $d = 0.02$ m, resulting in stable pitch dynamics.

An interesting feature of the system is that level flight is not possible at all pitch angles. There is a range, roughly from $-60 \leq \alpha \leq 60$ degrees, beyond which the wing cannot generate enough lift to compensate for the weight (5 N). It will appear crucial for mode switches to travel through this regime in a smooth manner. Appropriate feedforward is instrumental in achieving this transition. Even though the wing can generate enough lift for a range of angles of attack beyond stall, it is undesirable to fly at such high angle of attack for extended time since the increased drag results in much higher fuel consumption.

5.4 Model: Aerodynamic Forces from Theory

In this section we will use aerodynamic theory to model the aerodynamic forces for the thrust vectored aircraft depicted in Figure ??. We derive approximate expressions for the aerodynamics coefficients using [?]. This serves to get insight in the various aerodynamic effects and to check the experimental data from wind tunnel tests obtained in the previous section.

The boom is modelled as a cylinder in uniform flow, just as in Section ??. We model the fan body as a sphere, for which the drag coefficient is approximately given

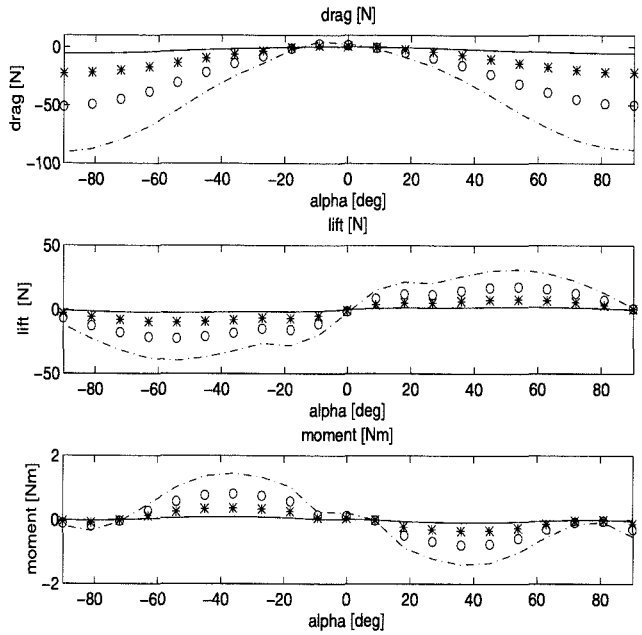


Figure 5.4 Experimental lift, drag and moment on the thrust vectored aircraft, for V = 3(-), 6(*), 9(o) and 12(-.) m/s

by:

$$C_{Db} = \begin{cases} 24/Re \text{ for } Re < 60 \\ 0.4 \text{ for } Re >= 60 \end{cases} \tag{5.12}$$

where $\mathrm{R}\epsilon = Vl\rho/\mu = Vl/\nu$ is the Reynolds number . For our fan, $l = 0.3$ m, is the diameter of the shroud, $\nu = 1.5 \times 10^{-5}$ is the kinematic viscosity of air.

The wing is a NACA 0015 airfoil [?]. This airfoil is symmetric, and the lift coefficient for finite aspect ratio $AR = b/\bar{c}$ and small angle of attack ($\|\alpha\| \leq 15$ degrees), is

$$C_{Lf} = \frac{2\pi}{1 + \frac{2}{AR}}\alpha. \tag{5.13}$$

The drag for small angle of attack satisfies

$$C_{Df} = (0.0031 + \frac{1}{\pi AR})C_{Lf}^2 + 0.006 \tag{5.14}$$

provided $\|\alpha\| \leq 15$ degrees. The wingspan is $b = 0.6$ meter, the mean aerodynamic chord is $\bar{c} = 0.35$ meter.

Since we are interested in maneuvers with high angle of attack, we need to model lift and drag in the stall region too. In the stall region, $\|\alpha\| > 15$ degrees, we model the wing as a flat plate in uniform flow, for which $C_D = 1.2$ for aspect ratios

$1 < L/d < 5$ and $\mathrm{R}\epsilon > 1.0 \times 10^3$. For the magnitude of the uniform flow we take the component of the air velocity perpendicular to the wing. The force on the wing then decomposes into lift and drag as follows:

$$\begin{aligned} C_{Df} &= 1.2 \sin\alpha \sin\alpha \\ C_{Lf} &= 1.2 \cos\alpha \sin\alpha. \end{aligned} \tag{5.15}$$

This is a very crude approximation, and here a validation with wind tunnel data is indispensable.

From aerodynamic theory, the effect of the flap is to increase the lift coefficient by an amount

$$\Delta C_L = 2(\pi + \sin\theta_h - \theta_h)\delta_e \tag{5.16}$$

where $\theta_h = -60$ degrees for our flap, which is hinged at the 3/4 chord point, and δ_e is the flap deflection.

We find the moment coefficient from the moments of lift and drag around the center of mass:

$$C_m = -dC_{Dw}\sin\alpha + dC_{Lw}\cos\alpha \tag{5.17}$$

where d is the distance between the center of mass and the center of pressure. A positive sign indicates a center of mass in front of the center of pressure, that is, a stable aircraft. We assume here that the center of pressure is constant with angle of attack, which is true for a symmetric airfoil, i.e. as long as $\delta_e = 0$. For nonzero elevator deflection, this assumption breaks down. For the Caltech ducted fan, the center of pressure is at the quarter chord point, and the center of mass of the shroud/wing assembly is 0.07 m from the leading edge of the wing. The distance from the center of mass to the center of pressure then is $0.09 - 0.07 = 0.02$ cm. Hence the aircraft is statically stable.

Figure ?? shows the lift, drag and moment as a function of airspeed and angle of attack, for $\delta_e = 0$, horizontal flight, zero paddle deflection, and zero thrust. The plots look qualitatively the same as those obtained from wind tunnel data, Figure ??, but the magnitudes are much smaller for the theoretical model. We submit this is due to the fact that the shroud will experience some lift, and therefore drag, which adds to the theoretical expressions. The sign reversal in the moment curve does not take place, due to the assumption of constant center of pressure. At high angle of attack the center of pressure shifts backward, resulting in static instability. The decrease and increase in lift beyond the stall angle of attack is predicted by the theory, as is the notch in the moment curve at the stall angle of attack (15 degrees).

5.5 Flight Modes

Since our experimental apparatus does not allow unlimited travel in the z direction, we only distinguish between hover and forward flight at constant altitude. This gives 4 possible transitions, the transition from forward flight to forward flight (at

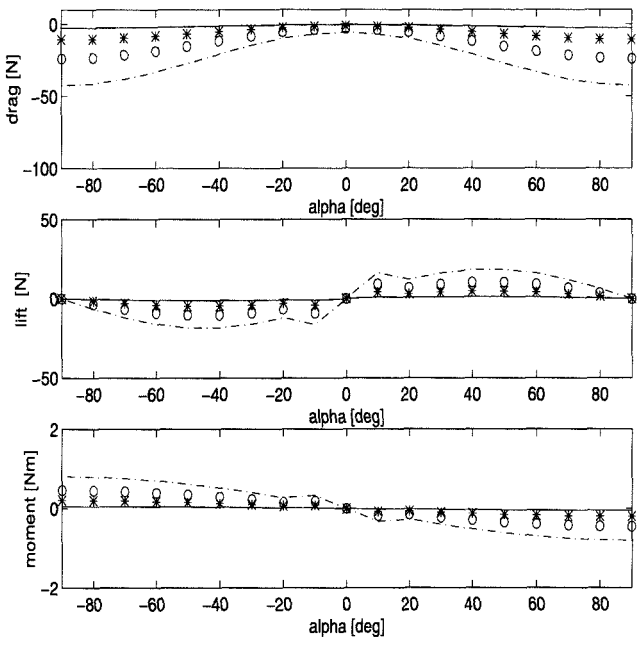


Figure 5.5 Theoretical lift, drag and moment on the thrust vectored aircraft, for V = 3(-), 6(*), 9(o) and 12(-.) m/s

a different velocity) being most typical for normal flight, the transition from hover to forward flight being most interesting due to the regime where the wing cannot carry the weight of the fan. For a mode switch we need to:

1. Determine the desired forward velocity and corresponding pitch and elevator trim.
2. Calculate a nominal velocity, pitch and elevator profile.
3. Apply to the appropriate controller for the transition.
4. Switch to the controller for the destination flight regime.

The computation time for the second step is determined by the dimension of the state, and the number of way points desired along the trajectory. For the system (??) it takes about 1 second for 50 points on a PC with a 66MHz Intel 486 DX2 microprocessor. This is with a controller running in the interrupt loop, i.e. the typical conditions one would have during flight. The computation time is roughly linear in the number of way points.

5.6 Pitch Dynamics and Pitch Trim Table

A first simple extension to the model (??) is to include the disturbance terms in the desired steady state. Even though the model (??) does not include aerodynamic

forces, we need to account for these to establish steady-state forward flight. That is, we need to apply a certain force to balance drag, and can use lift to balance gravity. If we restrict ourselves to horizontal forward flight, then from the equations (??) we can determine which forward velocity and pitch angle combinations establish a trim condition, i.e. lift equal to weight and zero moment. For certain forward velocities there will be more than one pitch value that establishes trim. One corresponds to high angle of attack and therefore high drag, one corresponds to low angle of attack, and low drag. It is the latter solution we are interested in for steady forward flight. From the forward velocity we calculate the necessary pitch angle, which equals the angle of attack in horizontal flight, to generate the desired lift. We tabulate the pitch-speed trim conditions and interpolate from this table to find the pitch trim corresponding to a certain airspeed. The moment generated by the lift is balanced by the perpendicular thrust.

5.7 Flat Systems with Perturbations

In this section we investigate some aspects of the approximation of a non flat system by a flat one. Suppose we have a nominal system that is flat, and generate a nominal trajectory for it. We design a scheduled controller that stabilizes the system around the trajectory. Then the error system becomes a linear system plus error terms:

$$\dot{x} = Ax + Bu + h(t, x). \tag{5.18}$$

Note that $h(x)$ contains perturbations to flatness and errors in the linearization, which might change along the trajectory. Therefore, $h(x)$ may still contain linear terms in x. Suppose we design a linear full state controller $u = Kx$ which gives us a quadratic Lyapunoff function $V(x) = x^*Px$ such that

$$\dot{V}(x) = -x^*Qx + 2x^*Ph \tag{5.19}$$

for the linear system plus error. Note that most linear controller design techniques provide a quadratic Lyapunoff function. Now allow a correction in the input: $u = Kx + \delta u$, then we get

$$\dot{V}(x) = -x^*Qx + 2x^*Ph + 2x^*PB\delta u. \tag{5.20}$$

It follows that setting

$$\delta_1 u = -B^\dagger h = -(B^*B)^{-1}B^*h \tag{5.21}$$

will always decrease the perturbation in the derivative of V. However, this error might actually be negative and help us decrease V. It is therefore advisable to check for the sign of this error before applying the correction (??). Moreover, we can do better: if $x \notin \mathcal{N}(B^*P)$, where $\mathcal{N}$ denotes the null space, we can cancel the

perturbation in (??) completely with

$$\delta_2 u = -(x^* PB)^\dagger x^* Ph. \tag{5.22}$$

Once again, we should only do this when the perturbation is positive. Obviously, even though the null space of B^*P is a thin set, the correction blows up for x almost aligned with the null space. We therefore need to incorporate some switching logic checking for the magnitude of x^*PB, resulting in a discontinuous control law:

$$\delta u = \begin{cases} \delta_2 u \text{ for } x^*PB > \epsilon \text{ and } x^*Ph > 0 \\ 0 \text{ otherwise} \end{cases} \tag{5.23}$$

where ϵ is some bound corresponding to the actuator limits. The theoretical justification for differential equations with a discontinuous right-hand side can be found in [?]. Discontinuous control laws are routinely used in sliding mode control [?].

One might even improve on Equation (??) and not only cancel the perturbation, but make the derivative of $V(x)$ arbitrarily negative. It is clear that there has to be some limit to this correction based on actuator saturation and the information encoded in the controller design. This is a topic of future research.

5.8 Simulations

We try the simple extensions to deal with perturbations to flatness on the model (??). We use the model (??) to generate the trajectory, and simulate on the model (??), which includes aerodynamic terms. We simulate the aircraft with one wing only. This is because our experimental aircraft is mounted on a stand, and has only 1 wing. For two wings we would simply multiply the wing span by 2.

We denote the angle with the vertical as $\Theta = \theta - \pi/2$. Then hover corresponds to $\Theta = 0$. We plot the nominal (solid) and simulated (dashed) traces of the forward velocity v_x, the altitude $y = -z$, Θ, and the perpendicular (solid) and parallel (dashed) thrust. The suffix 'd' denotes the desired, or nominal path of a variable.

The maneuver we are trying to follow is a transition from hover to forward flight, at a speed of 6 m/s, in 6 secs. The corresponding angle of attack that provides a lift equal to the weight (4.6 N) at this speed is $\alpha = 13$ degrees or $\Theta = -1.34$ radians. Since in the transition regime, the fan cannot lift its own weight, the transition will result in a drop in altitude in the nominal trajectory.

The controller used to stabilize around the trajectory is a gain scheduled LQR controller. The scheduling variable is pitch. We design controllers for $\Theta = 0.0$, $\Theta = \pi/2$ radians and $\Theta = -\pi/2$ radians, and switch with hysteresis between these

three as follows:

$$\begin{aligned} K &:= K_{-\pi/2} \text{ if } K = K_0 \text{ and } \Theta < -0.7\pi/2 \\ K &:= K_0 \text{ if } K = K_{-\pi/2} \text{ and } \Theta > -0.3\pi/2 \\ K &:= K_0 \text{ if } K = K_{+\pi/2} \text{ and } \Theta < +0.3\pi/2 \\ K &:= K_{+\pi/2} \text{ if } K = K_0 \text{ and } \Theta > 0.7\pi/2. \end{aligned} \tag{5.24}$$

The controller has integrators on both v_x and z to ensure zero steady-state error, which is important in set point regulation.

First we examine what happens if we use no model based nominal trajectory, but try to follow a linear interpolation between hover and forward flight. Figure ?? shows that the lack of an appropriate feedforward term results in severe pitch oscillations and poor altitude response.

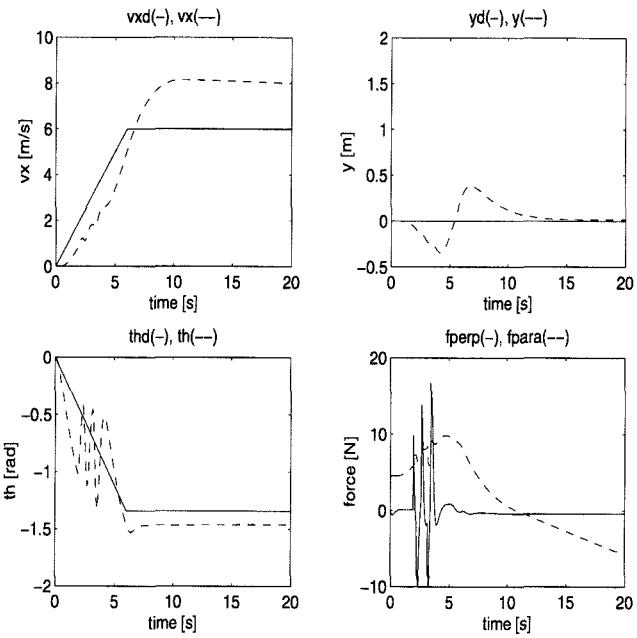


Figure 5.6 Simulation: maneuver from hover to forward flight, one degree of freedom design.

Next we use the flat system (??) to generate a nominal trajectory steering to an equilibrium of the nominal flat system. This is obviously not an equilibrium of the real system, and the effect is displayed in Figure ??. The nominal trajectory wants to steer back to zero angle with the vertical, and this results in dramatic excursions in altitude. Note that both here and in Figure ?? we steer to the right trim conditions due to the integrator on the velocity.

In Figure ?? we generate a nominal trajectory that steers to the right trim condition, corresponding to (??). We see clear improvement: the gain in altitude is significantly less. Notice the nonzero perpendicular force necessary to balance the

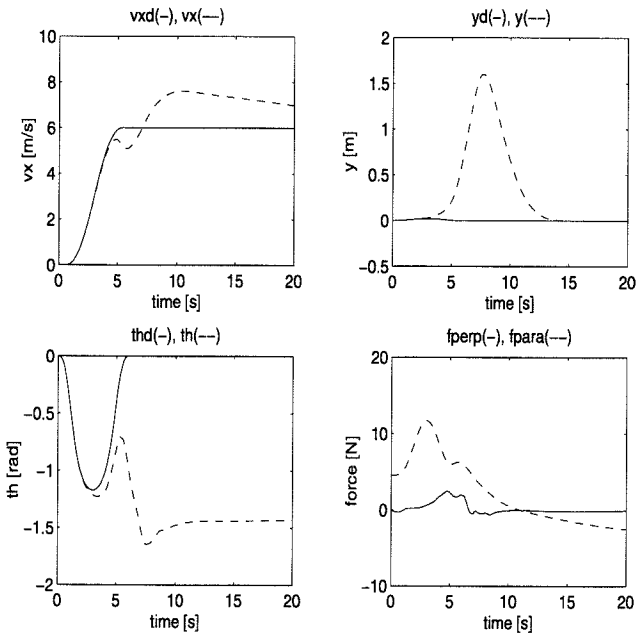


Figure 5.7 Simulation: maneuver from hover to forward flight, two degree of freedom design using flat system.

moment on the wing due to nonzero moment coefficient C_m.

Finally we compensate for the aerodynamic terms by a projection on the input space as described in Section ??. Figure ?? shows that we get a worse response if we apply input correction (??). We checked that the perturbation in (??) is negative most of the time. Figure ?? shows that applying the correction (??) does give a barely noticeable improvement in performance. Again, we checked that the correction is zero most of the time.

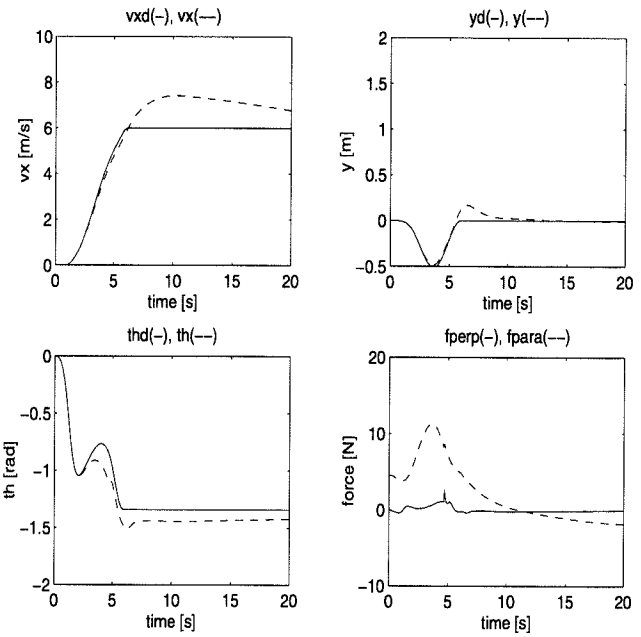


Figure 5.8 Simulation: maneuver from hover to forward flight, two degree of freedom design steering to aerodynamic trim.

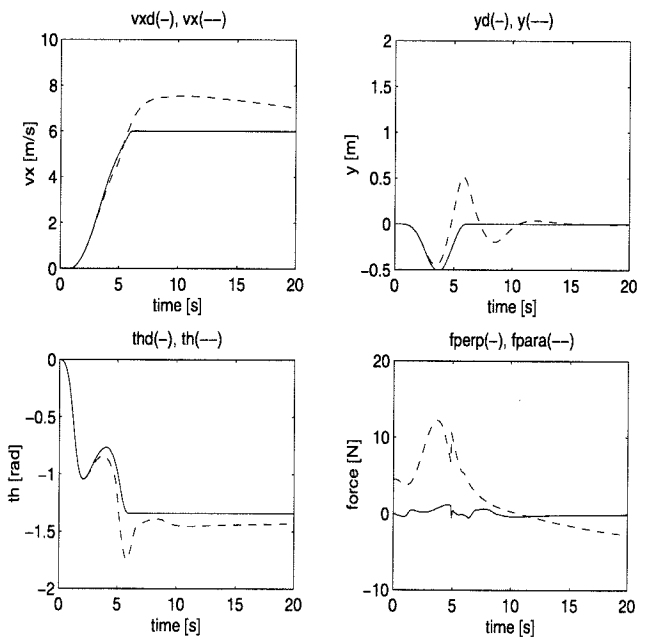


Figure 5.9 Simulation: maneuver from hover to forward flight, two degree of freedom design steering to aerodynamic trim, compensating for aero terms with projection, (??).

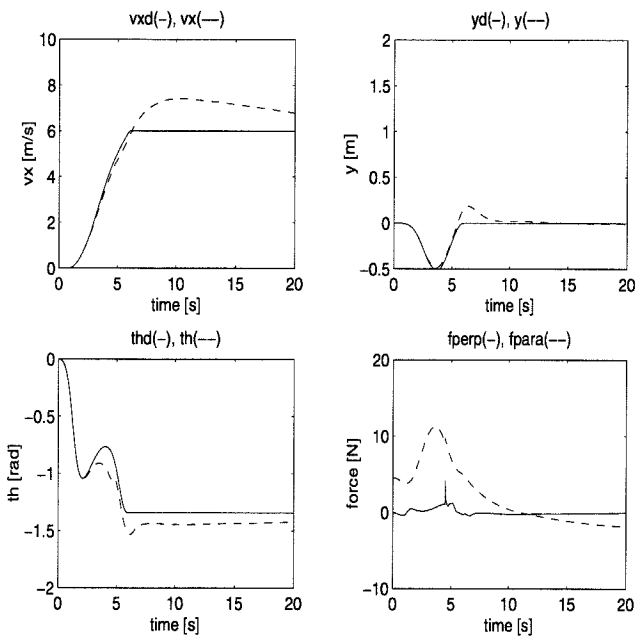


Figure 5.10 Simulation: maneuver from hover to forward flight, two degree of freedom design steering to aerodynamic trim, compensating for aero terms with Lyapunoff, (??).

Hover to forward flight:	no correction	Lyapunoff	projection
Figure:	??	??	??
e_y	0.0033	0.0044	0.0292
e_{v_x}	0.78	0.73	0.97
e_θ	0.01	0.01	0.02
Forward flight to forward flight:	no correction	Lyapunoff	projection
Figure:	??	??	??
e_y	6.46	6.12	17.6
e_{v_x}	2.82	2.74	3.31
e_θ	0.04	0.03	0.06

Table 5.1 L_2 errors for various input corrections

We also examine a more aggressive maneuver from forward flight in the positive direction, $v_x = +6$ m/s to forward flight in the negative direction, $v_x = -6$ m/s, in 12 seconds. Figure ?? shows the response with steering to the appropriate trim and two degree of freedom design. This is the counterpart of Figure ??. Figure ?? is the counterpart of Figure ?? and shows that adding the projective correction in Equation (??) deteriorates the performance. Figure ?? is the counterpart of Figure ??, and adds the Lyapunoff correction from Equation (??). Again the responses of Figures ?? and ?? are virtually identical, but the Lyapunoff based correction shows small improvement. Table ?? shows the L_2 error between nominal and simulated trajectory for the cases with no correction, with a projective correction, as in Equation (??), and a Lyapunoff correction as in Equation (??). Note the particularly poor response in altitude for all 3 cases. This is because the nominal trajectory prescribes a nose up pitching motion, and a decrease in altitude at the same time. These motions are perfectly feasible when no aerodynamic effects are present, however, due to the aerodynamic lift, the effect of pitching the nose up is a great increase in lift, causing an increase in altitude. Following the pitch trajectory is more important than following the altitude trajectory, since pitching the nose up is the best way to lose speed. Without the effect of drag, resulting from pitching the wing into the wind, the controller will command a negative thrust to decrease the velocity, which is impossible, since the propeller only rotates one way.

The projection based correction gives larger errors in both cases, the Lyapunoff based correction gives a slight improvement. We conclude that steering to the right pitch trim is crucial. Compensation for aerodynamic forces is not.

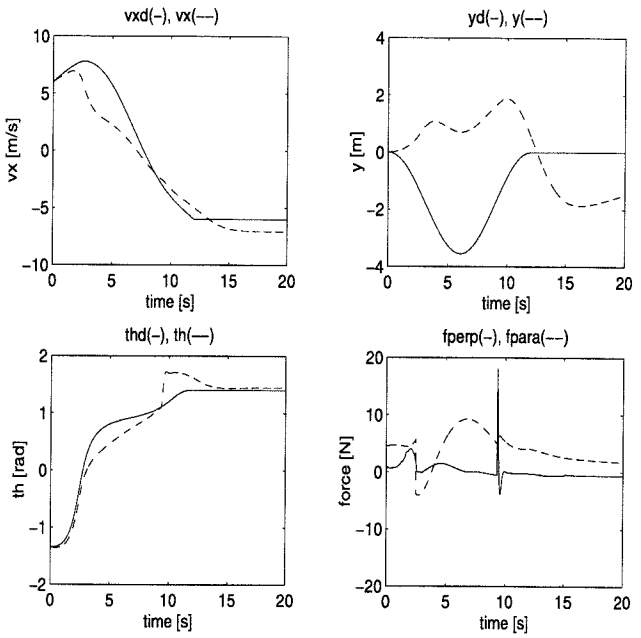


Figure 5.11 Simulation: maneuver from forward flight to forward flight, two degree of freedom design steering to aerodynamic trim, not compensating for aero terms.

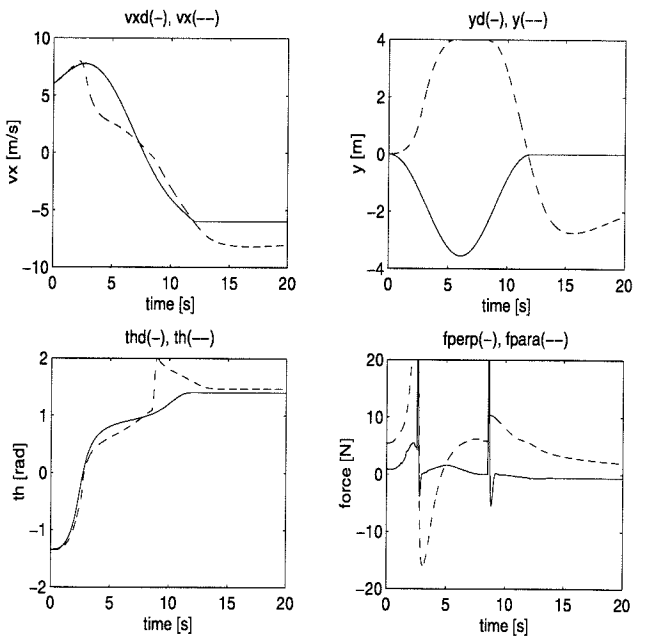


Figure 5.12 Simulation: maneuver from forward flight to forward flight, two degree of freedom design steering to aerodynamic trim, compensating for aero terms with projection, (??).

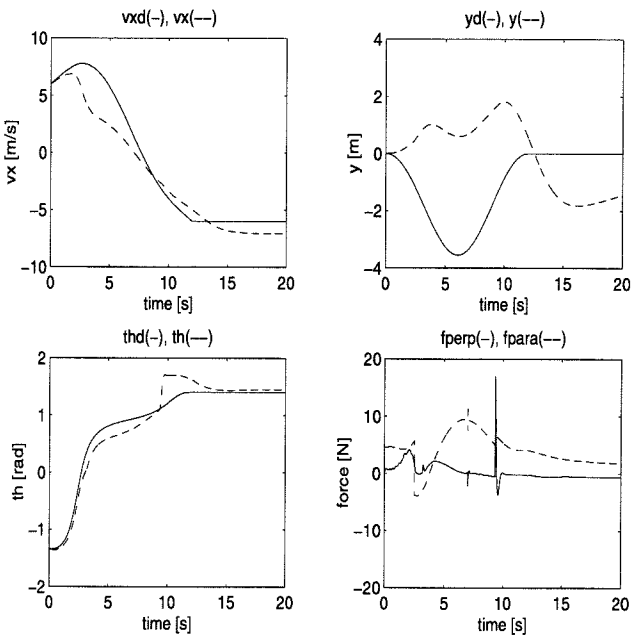


Figure 5.13 Simulation: maneuver from forward flight to forward flight, two degree of freedom design steering to aerodynamic trim, compensating for aero terms with Lyapunoff, (??).

5.9 Experiments

In Chapter ?? we reported that for hover to hover transitions two degree of freedom design improves significantly over one degree of freedom design. Figures ?? and ?? show the same experiment for the fan described in this chapter, with and without feedforward respectively. The results are similar: the use of feedforward achieves more aggressive tracking. The steady-state error in the attitude is due to stiction in the stand. To verify that these results are not a stroke of luck, we ran 10 instances of the same experiments, and extracted the minimum and maximum over all runs for the quantities of interest. Figures ?? and ?? show the nominal, minimum and maximum for all traces for the one and two degree of freedom design respectively. For x, y and θ, the nominal trajectory is plotted as a solid line, and the minimum and maximum over 10 runs are dashed lines. For the forces we plot just the experimental minimum and maximum and not the nominal trace. The experiments show good repeatability. The repeatability in the one degree of freedom design in Figure ?? is remarkable. The traces almost coincide. Yet, the two degree of freedom design in Figure ?? is consistently better. Apparently, the more aggressive tracking causes more variation in the trajectories. The altitude response for the one degree of freedom design is better, most likely because we do not command a change in altitude, and the stiction keeps the fan from moving up or down. The controller is not designed to overcome stiction. The two degree of freedom design does command a change in altitude, from which the controller cannot recover. Hence the poorer response in altitude for the two degree of freedom design.

Next we report experimental results for the mode switching from hover to forward flight, that was simulated in the previous section. We use the same pitch scheduled LQR controller as in the simulations. We measure x, z and Θ directly, and obtain their velocities by a digital filter.

Motivated by the simulation results we only compare trajectories that steer to the right trim without compensating for aerodynamic terms. Figures ?? and ?? show experimental data for a transition from hover to forward flight at 5 m/s in 5 seconds, for one and two degrees of freedom respectively. Even though the nominal trajectory calculated for the flat system and the linear interpolation for the one degree of freedom design look very similar, the improvement of two degree of freedom design is dramatic. The linear interpolation went unstable and we had to cut the power to the fan to prevent damage. Note the significant error in altitude in Figure ?? which is due to stiction in the stand.

Again, to verify repeatability of the above experiment, we perform 10 runs of this experiment and collect the minimum and maximum. The result is plotted in Figure ??. For v_x, y and θ we plot the nominal trajectory as a solid line, and the upper and lower bound as dashed lines. For the forces we plot just the upper and lower bound, and no nominal trajectory. The repeatability is remarkable. Since the one degree of freedom design was unstable, it does not make sense to compare multiple runs for this experiment.

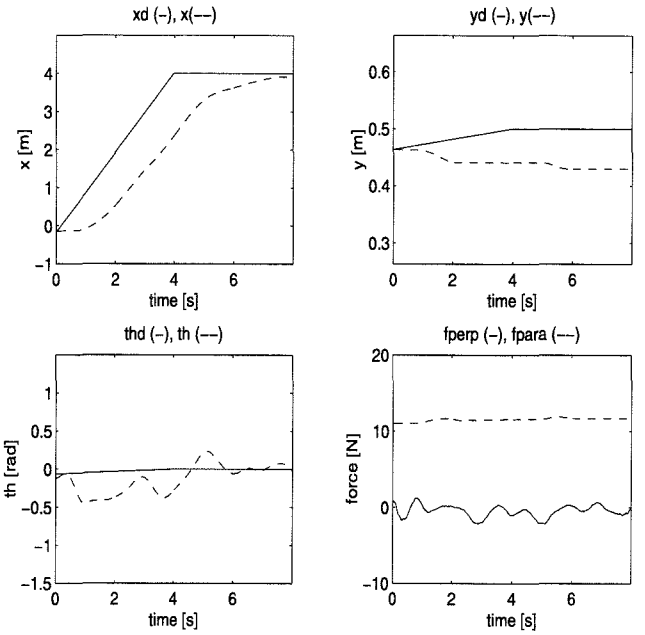


Figure 5.14 Experiment: horizontal step, one degree of freedom.

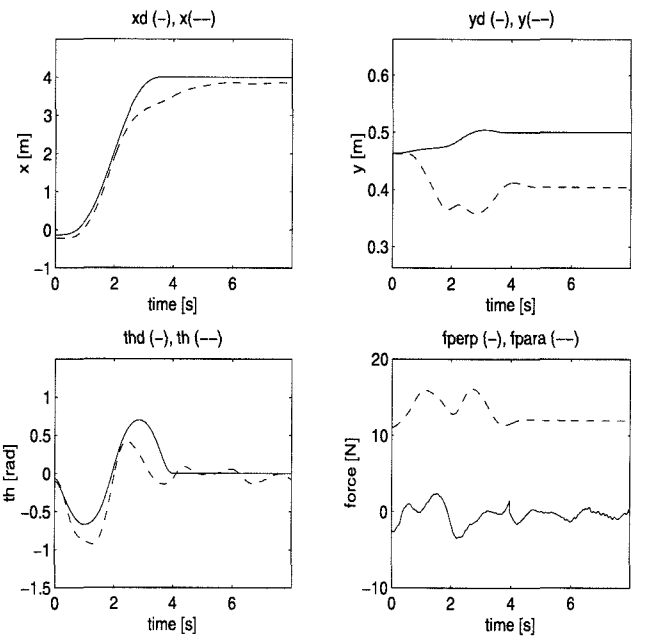


Figure 5.15 Experiment: horizontal step, two degree of freedom.

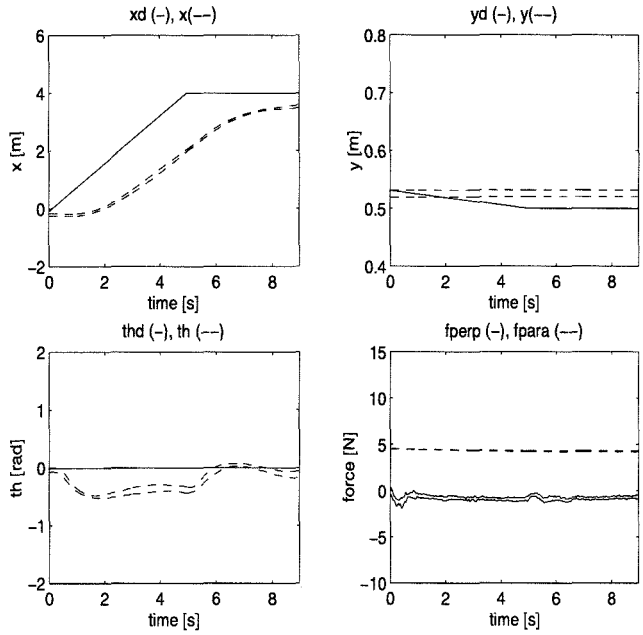


Figure 5.16 Experiment: horizontal step, one degree of freedom. Upper and lower bound over 10 runs.

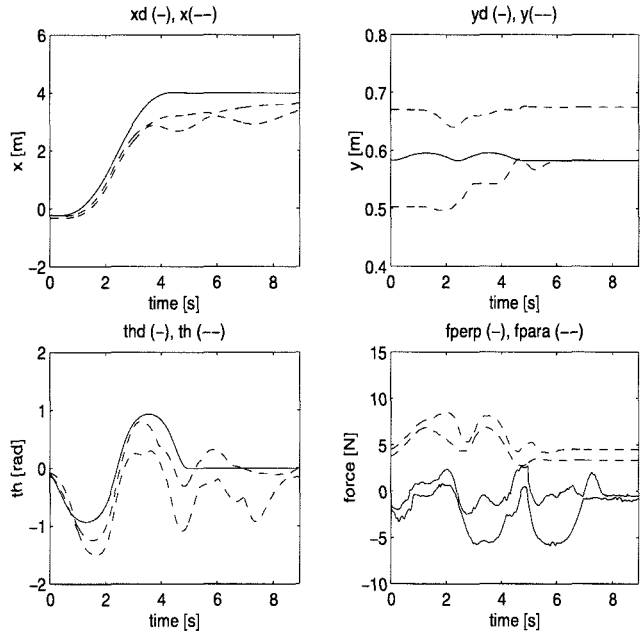


Figure 5.17 Experiment: horizontal step, two degree of freedom. Upper and lower bound over 10 runs.

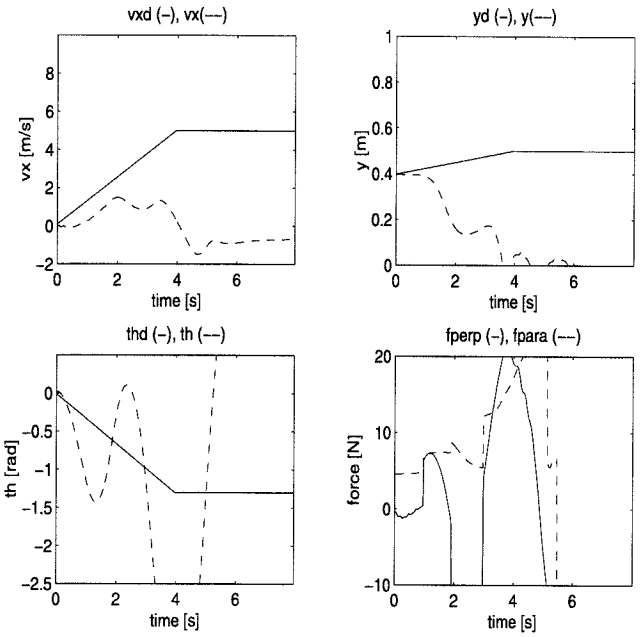


Figure 5.18 Experiment: hover to forward flight, one degree of freedom (unstable).

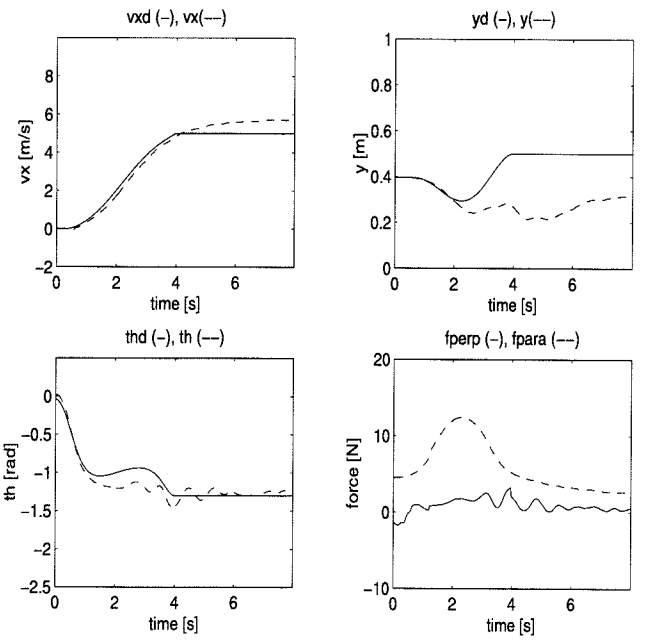


Figure 5.19 Experiment: hover to forward flight, two degree of freedom.

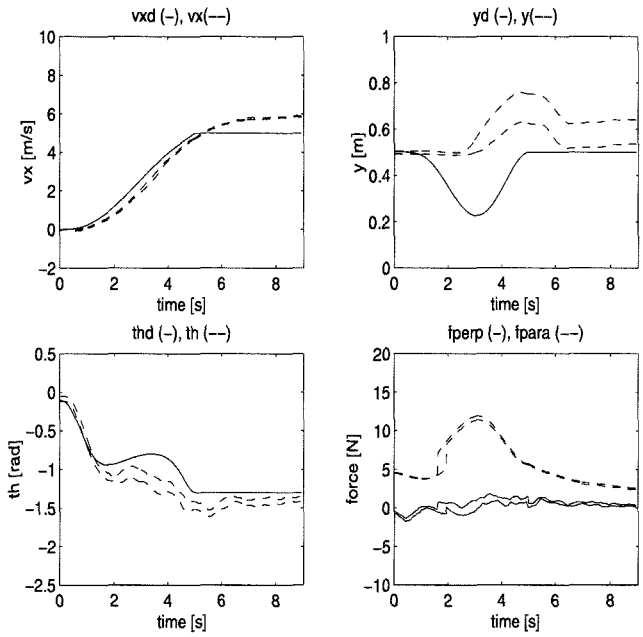


Figure 5.20 Experiment: hover to forward flight, two degree of freedom. Upper and lower bound over 10 runs.

5.10 Conclusions

Maneuvers involving transition to forward flight benefit from the use of a feedforward trajectory based on a flat approximation, provided this feedforward trajectory steers to the right trim for the nonflat system. Appropriate feedforward terms allow a smooth transition between hover and forward flight without much change in altitude, over a regime where no steady state exists.

Through a Lyapunoff stability argument we showed that absorbing the error between the flat system and the flat approximation in the input may adversely affect tracking. This is because the error may actually make the Lyapunoff function decrease faster. If we take the beneficial effect of the error into account, and only compensate when its effect is adverse, we obtain a small improvement in tracking.

It was shown in simulation and experiment that two degree of freedom design improves tracking for hover to forward flight transitions.

Much depends on the interplay between controller and feedforward, but in general we found that an appropriate feedforward term imposes lower bandwidth and gain requirements on the actuators. More research is needed to reach a conclusive answer about the benefits of two degree of freedom design in fast mode switching, but our preliminary results are promising.

Chapter 6

Outer Flatness: The Helicopter

6.1 Introduction

This chapter discusses a different way to use the tools of differential flatness on a system that is not strictly speaking flat. We look at the case where a system can be split in 2 subsytems. First, there is an outer system, in whose control we are really interested, but which we cannot control directly. The outer system is driven by an inner system, which we can control directly, but for which we do not care about the outputs *per se*. For mechanical systems, this split can be thought of as a split in actuator dynamics and rigid body dynamics. We do not care what the internal state of the actuator is, only that the actuators exert the right forces and torques on the rigid body. Many models of mechanical systems ignore actuator dynamics and only look at the outer system. To this extent the split in inner and outer dynamics is a natural one: it corresponds to leaving out part of the low level dynamics.

6.2 Theory

Suppose we have an input affine system of the form

$$\begin{aligned} \dot{x}_1 &= f_1(x_1) + g_1(x_1)y_2 \\ y_1 &= h_1(x_1) \end{aligned} \tag{6.1}$$

$$\begin{aligned} \dot{x}_2 &= f_2(x_2) + g_2(x_2)u \\ y_2 &= h_2(x_2), \end{aligned} \tag{6.2}$$

i.e. the system splits in an outer system with states x_1 and input y_2 and an inner system with states x_2 and input u. We assume the system is input affine for ease of notation, but it is not crucial to the following development. The vector y_2 is not really an input, in the sense that we have no direct control over y_2, but it serves as an intermediate input to the outer system. The output of the inner system y_2 is the input to the outer system, and the outer system depends on the inner system only through the output y_2.

In Section ?? we will give a geometric definition of outer flatness. For this section a formulation based on Equations (??) and (??) suffices.

Definition 6.1 (Outer flatness) We call the system (??), (??) *outer flat* if the system (??) is flat with respect to input y_2.

Remark 6.2 We can allow f_2 and g_2 in the inner system (??) to depend on the outer state x_1, at the expense of a more cumbersome notation. The complication is that the inner error system no longer depends linearly on the outer error. We will not include this dependency here, but indicate in Remark ?? on what modifications have to be made to include x_1 in the inner system.

If the system is outer flat, we can determine (x_1, y_2) from y_1 and its derivatives. We do not assume the total system is flat. If we can control the inner system tightly enough, we will follow the desired trajectory for y_2 closely, and therefore follow the desired output trajectory for y_1 closely. What tight enough means will be made more precise below. See Figure ?? for a pictorial representation of this configuration.

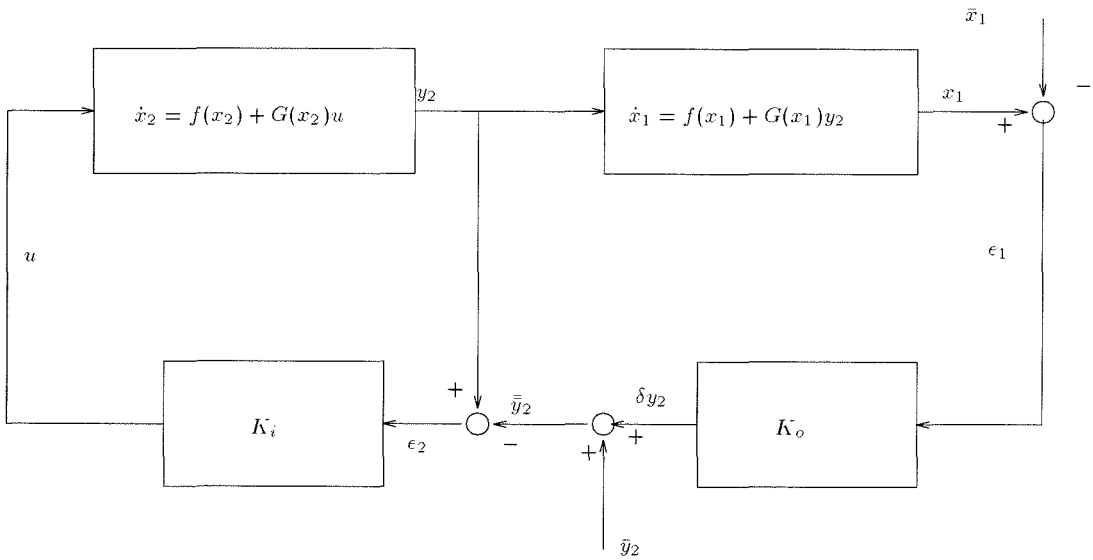


Figure 6.1 System structure for outer flatness.

Outer flatness is reminiscent of the backstepping techniques advocated in [?, ?, ?]. Note that in backstepping the inner system has to be minimum phase, and has to have relative degree equal to 1. We do not a priori require the inner system to be minimum phase, but it will turn out that in order to make general statements we will need that condition. This is only natural in view of the results in [?] about asymptotic tracking. The relative degree requirement in backstepping can be relaxed if one considers repeated backstepping, i.e. there is a chain of inner systems each serving as an outer system to its predecessor. For each level the relative degree requirement must hold. In backstepping, dependence of the inner system on the

outer state causes no extra complications, and is routinely included in the formalism. The structures of outer flat systems and backstepping system seem to be identical. However, the focus in this dissertation is on trajectory generation, and we will show in the sequel how outer flatness can be used for this purpose. The two degree of freedom paradigm advocates decoupled controller design and trajectory generation. In backstepping, the inner-outer structure determines the control design. We want to leave the controller design methodology free, and use trajectory generation as a performance enhancement. We use the inner-outer structure for trajectory generation, and allow any controller that seems best fit to do the job. Backstepping is a top down technique: one starts with the outer most system and ends up with the actuators. Outer flatness is a bottom up approach: we feel that for practical implementation it is imperative that the lower levels work before a controller for the higher level is designed. Backstepping is analogous to designing an autopilot before the aircraft is stabilized. The author of this thesis is not aware of any experimental implementation of backstepping.

Outer flatness also parallels the dynamic inversion advocated by the researchers at Honeywell [?, ?]. In dynamic inversion one considers a subsystem of the form

$$\dot{x}_2 = f_2(x) + g_2(x)u \tag{6.3}$$

where $g_2(x)$ is invertible. This implies that $\dim x_2 = \dim u$, i.e. we have as many control variables as commanded states. Given a commanded x_{2d}, one then inverts these dynamics by setting

$$u = g_2^{-1}(x)(\dot{x}_{2d} - f_2(x)). \tag{6.4}$$

In outer flatness we do not require that $\dim x_2 = \dim u$, since if this were the case for the inner system, the combined inner and outer systems would be flat. We only require some sort of tracking for the inner system. Dynamic inversion has been studied mainly in the context of aircraft control, where the pilot controls the attitude of the aircraft to obtain translational motion. In this case the inner system is also the total system. In the context of outer flatness we explicitly add trajectory generation for the outer system. Although dynamic inversion in principle does allow tracking of arbitrary commands, it only seems to have been applied to set point changes.

We want to know under what assumptions on the inner and outer system we can achieve asymptotic or bounded tracking for the outer system, while maintaining internal stability for the total system. The following development tries to answer that question.

Suppose we find a nominal trajectory $(\bar{x}_1, \bar{y}_2)$ corresponding to an output trajectory $\bar{y}_1$ for the outer system. Since the inner system is not flat, we cannot in general determine a full state space and input trajectory $(\bar{x}_2, \bar{u})$ from $\bar{y}_2$ for the inner system. Suppose also that we find an exponentially stabilizing control law $\bar{\bar{y}}_2 = \bar{y}_2 + K_o(x_1 - \bar{x}_1)$ for the nominal outer system with y_2 as an input, (i.e. with $e_2 \equiv 0$). Here K_o is an operator, not a gain matrix. Then the desired output for the inner system is $\bar{\bar{y}}_2$. Suppose we find an exponentially stabilizing controller

$u = K_i(x_2, \bar{y}_2)$ for the nominal inner system, i.e. with $K_o \equiv 0$, where again K_i is an operator. To simplify notation we assume that y_2 is a subset of the states x_2, i.e., $x_2 = (y_2, \xi_2)$ for some ξ_2, and $h(x_2) = y_2$ picks off the first dim y_2 entries of the state vector. If $\frac{\partial h}{\partial x_2} \neq 0$ we can achieve this through a coordinate transformation. Next we set $\bar{\xi}_2 \equiv 0$, so that $\bar{x}_2 = (\bar{y}_2, 0)$ and $\bar{\bar{x}}_2 = (\bar{\bar{y}}_2, 0)$. This is not necessarily a feasible trajectory, but corresponds to a one degree of freedom design. Writing

$$\begin{aligned} e_1 &= x_1 - \bar{x}_1 \\ e_2 &= x_2 - \bar{\bar{x}}_2, \end{aligned} \tag{6.5}$$

using $x_2 = \bar{x}_2 + K_o(e_1) + e_2$, and assuming that $u = K_i(x_2, \bar{\bar{y}}_2) = K_i(x_2, \bar{y}_2 + K_o(e_1))$ is linear in e_1, we can write the closed loop error system as

$$\begin{aligned} \dot{e}_1 &= f_1(x_1) + g_1(x_1)y_2 - f_1(\bar{x}_1) - g_1(\bar{x}_1)\bar{y}_2 = F_1(t, e_1) + G_1(t, e_1)e_2 \\ \dot{e}_2 &= f_2(x_2) + g_2(x_2)u - \dot{\bar{x}}_2 = F_2(t, e_2) + G_2(t, e_2)e_1. \end{aligned} \tag{6.6}$$

Both the inner system with $e_1 \equiv 0$ and the outer system with $e_2 \equiv 0$ are exponentially stable by design of K_i and K_o.

Without further conditions we cannot conclude stability of the combined system as is clear from the following example.

Example 6.3 Suppose both e_1 and e_2 are scalars, and F_1, F_2, G_1 and G_2 are scalar constants. Then the error system (**??**) becomes

$$\begin{aligned} \dot{e}_1 &= F_1 e_1 + G_1 e_2 \\ \dot{e}_2 &= F_2 e_2 + G_2 e_1 \end{aligned} \tag{6.7}$$

and it is clear that $F_1, F_2 < 0$ does not imply stability of the combined system.

To proceed with our stability analysis, we invoke a Lyapunoff argument. We will use the converse Lyapunoff theorem (Theorem 4.5 in [**?**]) which we repeat here for completeness. See [**?**] for a comprehensive treatment of stability.

Theorem 6.4 (Converse Lyapunoff Theorem) *Suppose $x = 0$ is an exponentially stable equilibrium of the system*

$$\dot{x} = f(t, x) \tag{6.8}$$

with f continuously differentiable on $x \in B_R(0)$ and with Jacobian matrix $\frac{\partial f}{\partial x}$ uniformly bounded in time on $B_R(0)$. Then there exist a $R_0 > 0$ and a Lyapunoff function $V(t, x) : [0, \infty[\times B_{R_0}(0) \to \mathbb{R}$ satisfying

$$\begin{aligned} c_1\|x\|_2^2 \leq V(t, x) &\leq c_2\|x\|_2^2 \\ \frac{\partial V}{\partial t} + \frac{\partial V}{\partial x} f(t, x) &\leq -c_3\|x\|_2^2 \\ \|\frac{\partial V}{\partial x}\|_2 &\leq c_4\|x\|_2 \end{aligned} \tag{6.9}$$

for some positive constants c_i. Moreover, if the system is globally exponentially stable and $R = \infty$, then $R_0 = \infty$. If the system is autonomous, V can be chosen independent of t.

Assuming the Jacobians of F_1 and F_2 exist and are uniformly bounded around the origin, we can find Lyapunoff functions $V_i(e_2), V_o(e_1)$ for the nominal inner and outer systems, with constants c_{ji}, c_{jo}, $j = 1, \ldots, 4$ respectively. We will try to construct a Lyapunoff function for the combined system using the Lyapunoff functions for the inner and outer system. In the following, the subscript i refers to the inner system, the subscript o to the outer system. The candidate Lyapunoff function $V = V_i + \lambda V_o$, with $\lambda > 0$ for the combined system still satisfies the requirements of (**??**). The derivative satisfies

$$\begin{aligned}\dot{V}(t, e_1, e_2) &\leq -\lambda c_{o3}\|e_1\|^2 - c_{i3}\|e_2\|^2 + \lambda\|\frac{\partial V_o}{\partial e_1}G_1(t, e_1)e_2\| + \|\frac{\partial V_i}{\partial e_2}G_2(t, e_2)e_1\| \\ &\leq -\lambda c_{o3}\|e_1\|^2 - c_{i3}\|e_2\|^2 + \lambda c_{o4}\|e_1\|\|G_1(t, e_1)\|\|e_2\| + c_{i4}\|e_2\|\|G_2(t, e_2)\|\|e_1\|.\end{aligned} \tag{6.10}$$

Hence,

$$\begin{aligned}\dot{V}(t, e_1, e_2) &\leq -\begin{pmatrix} \|e_1\| & \|e_2\| \end{pmatrix}\begin{pmatrix} \lambda c_{o3} & -\lambda c_{o4}\|G_1(t, e_1)\| \\ -c_{i4}\|G_2(t, e_2)\| & c_{i3} \end{pmatrix}\begin{pmatrix} \|e_1\| \\ \|e_2\| \end{pmatrix} \\ &=: -\begin{pmatrix} \|e_1\| & \|e_2\| \end{pmatrix} P \begin{pmatrix} \|e_1\| \\ \|e_2\| \end{pmatrix}.\end{aligned} \tag{6.11}$$

Writing p_{ij} for the element in the i-th row and j-th column of the matrix P we arrive at the following:

Theorem 6.5 (Exponential tracking for outer flatness) *Suppose K_i and K_o are exponentially stabilizing controllers for the nominal inner system with $e_1 \equiv 0$ and the nominal outer system with $e_2 \equiv 0$ respectively. Suppose the Jacobians of F_1 and F_2 exist and are uniformly bounded in time on some balls around the origin. Suppose u enters the error system (**??**) linearly. Suppose for some $\lambda > 0$, we have $p_{11}p_{22} - \frac{1}{4}(p_{12} + p_{21})^2 > 0$ for all $e_1 \in B_{R_o}$, $e_2 \in B_{R_i}$, where $R_o, R_i > 0$ and for all $t > 0$. Then we have exponential stability of the combined inner-outer system (**??**), (**??**).*

Proof: The symmetric part of P is

$$P_s = \begin{pmatrix} p_{11} & \frac{p_{12}+p_{21}}{2} \\ \frac{p_{12}+p_{21}}{2} & p_{22} \end{pmatrix}. \tag{6.12}$$

Since $p_{11} > 0$, $\det P_s = p_{11}p_{22} - \frac{1}{4}(p_{12}+p_{21})^2 > 0$ implies that P is positive definite. Then V is a Lyapunoff function for the combined system in a ball around the origin with radius $\min(R_i, R_o)$, and the combined system achieves asymptotic tracking. ■

Remark 6.6 The condition that the system (??) is linear in e_1 does not restrict us to static feedback since we can modify the outer system. If we design a dynamic controller K_o, we can add its states to the outer system. The outer system then might lose flatness, but we do not care about the nominal trajectory for the controller states, and can set them to zero. Of course this will slow down the response, since the nominal trajectory is no longer feasible. If we want to include nonlinear controllers K_o, Theorem ?? essentially still holds. F_2 in Equation (??) will now depend on e_1, and we need to expand F_2 in a Taylor series and bound the sum of terms in the derivative of the combined Lyapunoff function in Equation (??). By restricting e_1 to be small, this perturbation can be made arbitrarily small. The notation becomes cumbersome and we do not elaborate this issue.

Remark 6.7 Theorem ?? is rather academic in the sense that its conditions are at best cumbersome to check. It is in general hard to find Lyapunoff functions for systems. The theorem merely serves to show that there are no theoretical obstacles for the outer flatness approach to work.

Remark 6.8 The perceptive reader will realize that in view of [?], in order to design an exponentially stabilizing controller for the inner system that works for arbitrary trajectories, it must be minimum phase. If we are only interested in particular trajectories we do not need this requirement. In this sense the conditions for backstepping are also required for outer flatness.

Remark 6.9 The constant λ can be used to try to make the derivative of V negative.

Next we investigate what happens if we give up exponential tracking for the inner system, and only require bounded tracking. Suppose there exists a function $W_i(t, e_2)$ with

$$\begin{aligned} c_{i1}\|e_2\|_2^2 \leq W_i(t,e_2) &\leq c_{i2}\|e_2\|_2^2 \\ \frac{\partial W_i}{\partial t} + \frac{\partial W_i}{\partial e_2}F_2(t,e_2) &\leq -c_{i3}\|e_2\|_2^2 \text{ for } \|e_2\| > R_i \\ \|\frac{\partial W_i}{\partial e_2}\|_2 &\leq c_{i4}\|e_2\|_2 \end{aligned} \tag{6.13}$$

that is, W_i is a Lyapunoff function with the difference that the derivative is only negative for large e_2. It follows from Theorem 4.10 in [?] that this guarantees bounded tracking. We call W a *bounding function*. Now assume that K_o still achieves exponential tracking for the outer system with $e_2 \equiv 0$, and let V_o be a Lyapunoff function for this nominal outer system. For the function $W = \lambda V_o + W_i$

we have:

$$\begin{aligned}\dot{W}(t,e_1,e_2) &\leq -\lambda c_{o3}\|e_1\|^2 - c_{i3}\|e_2\|^2 + \lambda\|\frac{\partial V_o}{\partial e_1}G_1(t,e_1)e_2\| + \|\frac{\partial W_i}{\partial e_2}G_2(t,e_2)e_1\| \\ &\leq -\lambda c_{o3}\|e_1\|^2 - c_{i3}\|e_2\|^2 + \lambda c_{o4}\|e_1\|\|G_1(t,e_1)\|\|e_2\| + \\ &c_{i4}\|e_2\|\|G_2(t,e_2)\|\|e_1\|,\end{aligned} \tag{6.14}$$

for $\|e_2\| > R_i$. Hence,

$$\begin{aligned}\dot{W}(t,e_1,e_2) &\leq -\begin{pmatrix}\|e_1\| & \|e_2\|\end{pmatrix}\begin{pmatrix}\lambda c_{o3} & -\lambda c_{o4}\|G_1(t,e_1)\| \\ -c_{i4}\|G_2(t,e_2)\| & c_{i3}\end{pmatrix}\begin{pmatrix}\|e_1\| \\ \|e_2\|\end{pmatrix} \\ &=: -\begin{pmatrix}\|e_1\| & \|e_2\|\end{pmatrix}P\begin{pmatrix}\|e_1\| \\ \|e_2\|\end{pmatrix}\end{aligned} \tag{6.15}$$

for $\|e_2\| > R_i$.

Hence we arrive at the following theorem.

Theorem 6.10 (Bounded tracking for outer flatness) *Suppose K_0 achieves exponential tracking for the nominal outer system with $e_2 \equiv 0$, and that W_i is a bounding function for the nominal inner system with $e_1 \equiv 0$ satisfying (??). Suppose the Jacobians of F_1 and F_2 exist and are uniformly bounded in time on some balls around the origin. Suppose u enters the error system (??) linearly. Suppose there is a $\lambda > 0$ and an annulus $R_i \leq \gamma_1 \leq \|(e_1,e_2)\| \leq \gamma_2$, where $p_{11}p_{22} - \frac{1}{4}(p_{12}+p_{21})^2 > 0$, for all $t > 0$. Then we achieve bounded tracking for the combined inner-outer system.*

Proof: Analogous to Theorem **??** we have that since $p_{11} > 0$, $\det P_s > 0$ implies that P is positive definite on the annulus. Then W is a bounding function for the combined system. ■

Remark 6.11 Again, this theorem is quite academic, and merely serves to show there are no theoretical obstacles to the outer flatness approach.

6.3 A Geometric Interpretation of Outer Flatness

To remain true to our definition of flatness in terms of differential geometry in Chapter 2, we will make the Definition **??** of the previous section more precise by a formulation in a differential geometric setting.

Definition 6.12 Let I be a control system, (i.e. $\{I, dt\}$ is integrable), on a manifold $M \times U \times T$. Let I_i be a submodule $I_i \subset I$ on $M_i \times U \times T$ generating a control system. Let the complement I_o (i.e. $I = I_i \oplus I_o$) on $M_o \times Y \times T$ also be a control system, and let the manifolds satisfy $M = M_i \times M_o$ and $Y \subset M_i$. If I_o is differentially flat, then we call I *outer flat* with inner system I_i and outer system I_o.

Remark 6.13 The requirement that $Y \subset M_i$, translates to the requirement that the inputs of the outer system are functions defined on the inner manifold.

Remark 6.14 In particular, every flat system is outer flat with $I_i = \{0\}$.

Example 6.15 The submodule I_i is not unique, as is illustrated in this example. Let

$$\begin{aligned} I &= \{\omega_1, \omega_2, \omega_3, \omega_4\} \\ &= \{dx_1 - x_2 dt, dx_2 - x_3 dt, dx_3 - (x_4 + u)dt, dx_4 - (-x_3)dt\} \end{aligned} \tag{6.16}$$

which is outer flat with inner system $I_i = \{\omega_2, \omega_3, \omega_4\}$ or with inner system $I_i = \{\omega_3, \omega_4\}$

Remark 6.16 If we want to track functions $h_o(x)$, we need $dh_o \in I_o \cup \{dt\}$ to make sure that the functions h_o are determined by the solution curves of I_o.

6.4 The Model Helicopter

The Caltech model helicopter experiment is depicted in Figures **??** and **??**. It is a Kyosho EP Concept electric model helicopter with a 30 inch diameter main rotor, mounted on a 6 degree of freedom stand. For a detailed description of the experiment, see [?, ?, ?]. We have 5 independent controls: left-right cyclic (aileron), fore-aft cyclic (elevator), collective pitch (heave), directional (rudder) and throttle (motor current). The throttle is either ganged to collective, since an increase in collective pitch angle increases the load on the disk and therefore requires an increase in throttle, or is regulated around a constant set point by a low level SISO controller. See [?] for the design of a low level rotor speed governor.

At some level of abstraction we can look at the helicopter as a rigid body actuated by the thrust of the main rotor and the tail rotor. The tail rotor exerts a thrust along the body y axis and a torque along the body z axis. The tail rotor force is small compared to the thrust of the main rotor and we neglect it. The main thrust is roughly aligned with the body z axis. We can measure the XYZ Euler angles (ϕ, θ, ψ) with a Polhemus 6 degree of freedom position sensor described in [?]. Actually, the sensor measures ZYX Euler angles , but from these we can readily calculate the XYZ Euler angles, see [?, ?]. The tail rotor torque τ_b and the main thrust T_b then both act along the body z axis and can be transformed to spatial coordinates by rotations about the y and x axis about angles θ and ϕ respectively. The subscript b indicates that the vector is in body coordinates, the subscript s indicates spatial coordinates. Note that according to aerodynamic convention the z axis is positive pointing down, hence $T_b < 0$ is a thrust upward. Then the thrust in spatial coordinates is

$$T_s = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix} \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ T_b \end{pmatrix}, \tag{6.17}$$

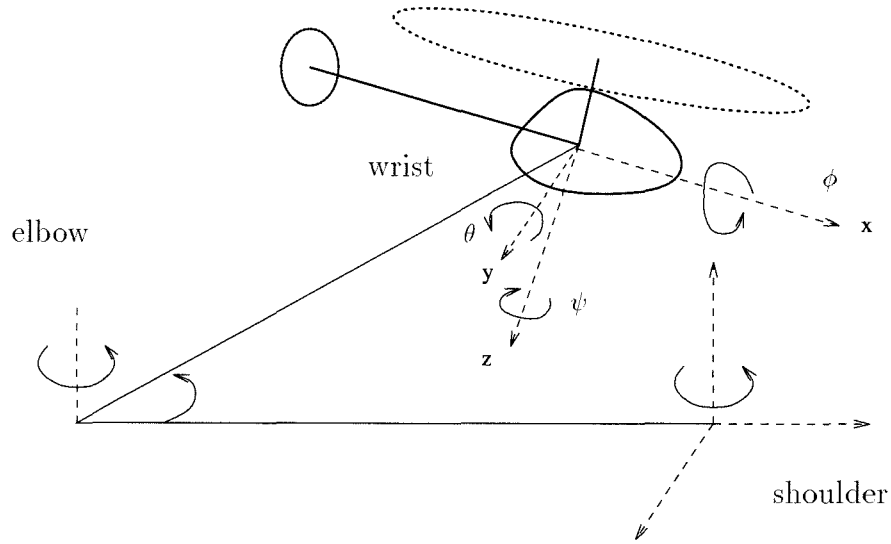


Figure 6.2 Model helicopter experimental setup.

and similarly for the torque. Writing (x, y, z) for the center of mass in spatial coordinates, the rigid body equations for the model helicopter then take the form

$$\begin{pmatrix} m\ddot{x} \\ m\ddot{y} \\ m\ddot{z} \\ J\ddot{\psi} \end{pmatrix} = \begin{pmatrix} T_b \sin\theta \\ -T_b \cos\theta \sin\phi \\ T_b \cos\phi \cos\theta + mg \\ \tau_b \cos\phi \cos\theta \end{pmatrix}, \tag{6.18}$$

where m is the mass of the helicopter, J is the moment of inertia about the z axis, and g is the gravitational acceleration. Note that we have no direct control over roll (ϕ) and pitch (θ) but only through left-right (aileron) and fore-aft (elevator) cyclic control respectively. However, if we can make the control of roll and pitch tight, we are exactly in the situation of Equation (**??**). The thrust T_b and the torque τ_b are real control inputs, the pitch angle θ and the roll angle ϕ are pseudo inputs. We will present the dynamics for pitch and roll below.

Note that the outer system is flat since from (x, y, z, ψ) we can recover the inputs and pseudo-inputs through:

$$\begin{aligned} |T_b|^2 &= m^2(\ddot{x}^2 + \ddot{y}^2 + (\ddot{z} - g)^2) \\ \theta &= \arcsin \frac{m\ddot{x}}{T_b} \\ \phi &= -\arcsin \frac{m\ddot{y}}{T_b \cos\theta} \\ |\tau_b| &= \frac{\ddot{\psi}}{J \cos\phi \cos\theta}. \end{aligned} \tag{6.19}$$

We cannot determine the sign of T_b, since flying right side up with positive thrust

cannot be distinguished from flying upside down with negative thrust. We will assume that the the helicopter always flies right side up.

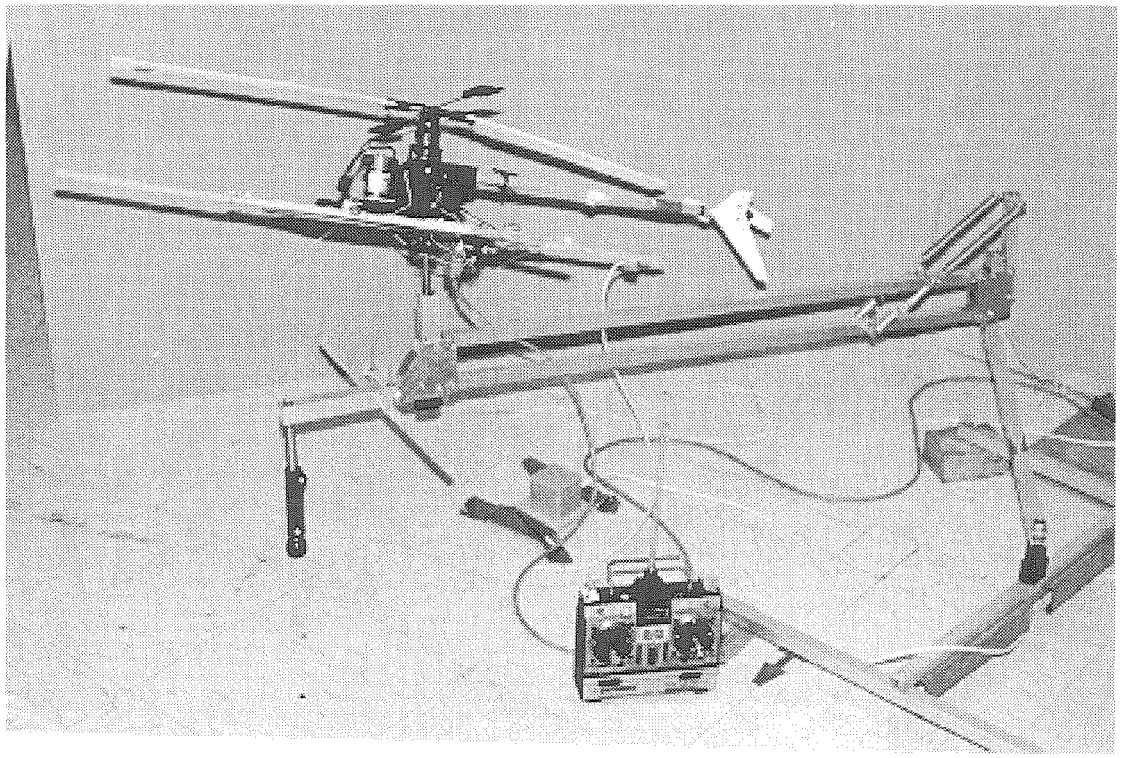


Figure 6.3 Kyosho Concept EP30 model helicopter on stand.

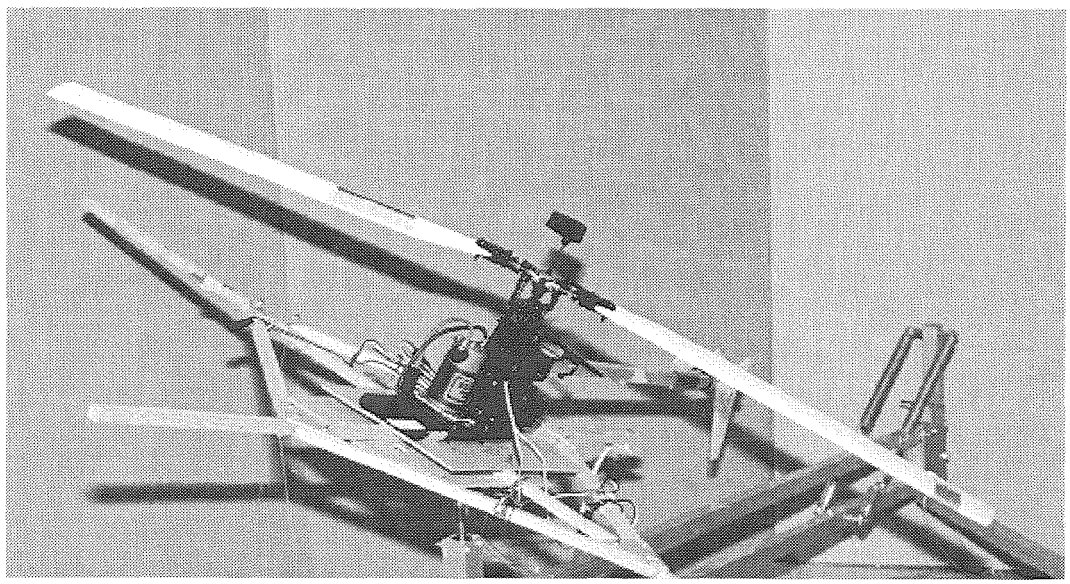


Figure 6.4 Kyosho Concept EP30 model helicopter.

6.5 Identification of Hover Dynamics

The model for the roll and pitch dynamics cannot be obtained analytically, and we resort to linear identification with experimental data around hover. See [?, ?, ?] for a treatment of linear ID, and [?] for a more detailed treatment of the ID of the helicopter. The equation for yaw (ψ) in Equation (??) is not very accurate, since we do not command the yaw torque directly, but rather the pitch angle of the tail rotor blades. Therefore we include the yaw dynamics in the linear identification. Note that substituting the yaw acceleration in Equation (??) by

$$\ddot{\psi} = \cos\phi\cos\theta(A_6[\phi, \theta, \psi, p, q, r] + B_6[\delta_a, \delta_e, \delta_r]) \tag{6.20}$$

does not disturb flatness of the system (??). Here $\delta_{a,e,r}$ are the aileron, elevator and rudder servo input respectively, and A_6 and B_6 are constant vectors of appropriate dimensions, obtained from the linear identification.

Since the sampled data is discrete time, we undertake to identify the dynamics for roll, pitch and yaw in discrete time. If we write p , q and r for the roll, pitch and yaw rates respectively, and δ_a, δ_e, δ_r for the aileron, elevator and rudder respectively, then the state becomes $x = [\phi, \theta, \psi, p, q, r]^T$, the input becomes $u = [\delta_a, \delta_e, \delta_r]^T$ and we need to identify a system of the form:

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k). \end{aligned} \tag{6.21}$$

If needed the discrete time model can be transformed to continuous time.

We impose the following model structure on the angular dynamics:

$$\begin{aligned} A &= \begin{bmatrix} 1 & 0 & 0 & T_s & 0 & 0 \\ 0 & 1 & 0 & 0 & T_s & 0 \\ 0 & 0 & 1 & 0 & 0 & T_s \\ a_{41} & a_{42} & 0 & a_{44} & 0 & a_{46} \\ 0 & a_{52} & 0 & 0 & a_{55} & 0 \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} \end{bmatrix}, \\ B &= \begin{bmatrix} b_1 & 0 & 0 \\ 0 & b_2 & 0 \\ 0 & 0 & b_3 \end{bmatrix}, \\ C &= [I_3\ 0_3], \end{aligned} \tag{6.22}$$

where T_s is the sampling period. The particular structure of the A matrix was obtained by examination of the standard deviation and repeatability of the various coefficients over different data sets.

Running the prediction error method (PEM) described in [?, ?, ?], we arrived

at the following values for the coefficients:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0.02 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0.02 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0.02 \\ -0.12 & -0.08 & 0 & 0.82 & 0 & 0.02 \\ 0 & 0.01 & 0 & 0 & 0.75 & 0 \\ -0.18 & -0.08 & -0.14 & -0.02 & 0.10 & 0.89 \end{bmatrix},$$

$$B = \begin{bmatrix} 0.37 & 0 & 0 \\ 0 & -0.83 & 0 \\ 0 & 0 & -2.03 \end{bmatrix}, \tag{6.23}$$

$$C = [I_3 \; 0_3].$$

6.6 Simulations

In the forthcoming simulations we use the inner model (roll and pitch dynamics) from the linear identification, Equation (??), and the outer model from Equation (??) with the yaw dynamics replaced by the linear dynamics from (??).

We design LQR controllers for the inner and outer system, and use the structure of Figure ?? to simulate the system. In order to stabilize the system we need the inner system to be faster than the outer system. This does not follow straightforwardly from Theorems ?? and ??, but is intuitively clear. For LQR controllers, this translates into higher weights on the states for the inner system than for the outer system. If we put higher weights on the outer system than on the inner system, the simulations go unstable. Figures ?? and ?? show a step response of 0.4 m in 2 seconds in the x (forward) direction, for a two and one degree of freedom controller respectively. In these and subsequent plots, the nominal trajectory from Equation (??) is plotted as a solid line, the commanded trajectory for the inner system is plotted as a dotted line (this is the trajectory $\bar{y}_2$ from Section ??), and the simulated system response is plotted as a dashed line. We see that the effect of two degree of freedom design is a great improvement of performance. Figures ?? and ?? show simulated step responses of 0.4 m in 2 seconds simultaneously in the x (forward) and y (sideways) directions, for a two and one degree of freedom design respectively. Again, the two degree of freedom design improves performance significantly. We conclude that the simulations validate the outer flatness approach.

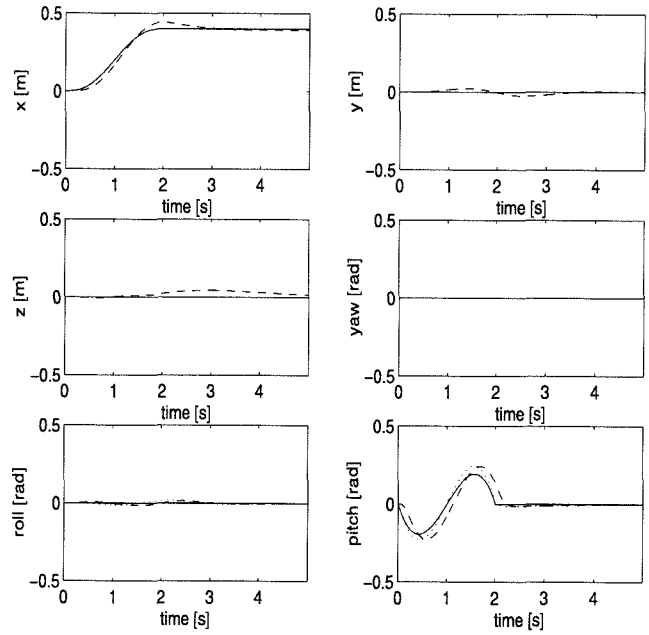


Figure 6.5 Simulation: two degree of freedom controller. Solid: nominal, dotted: commanded, dashed: experimental. Step sideways.

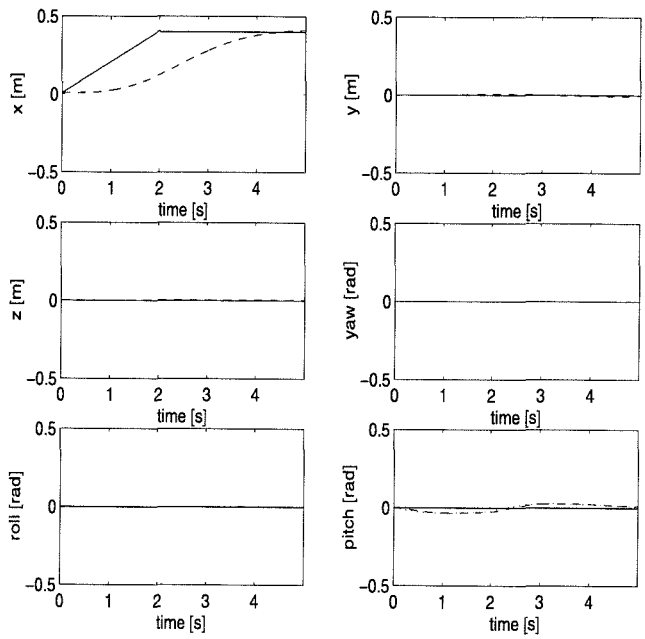


Figure 6.6 Simulation: one degree of freedom controller. Solid: nominal, dotted: commanded, dashed: experimental. Step sideways.

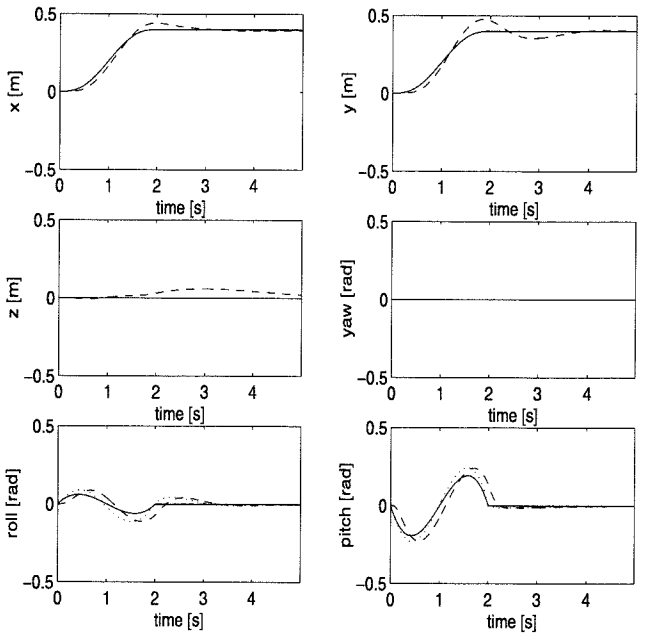


Figure 6.7 Simulation: two degree of freedom controller. Solid: nominal, dotted: commanded, dashed: experimental. Simultaneous step sideways and forward.

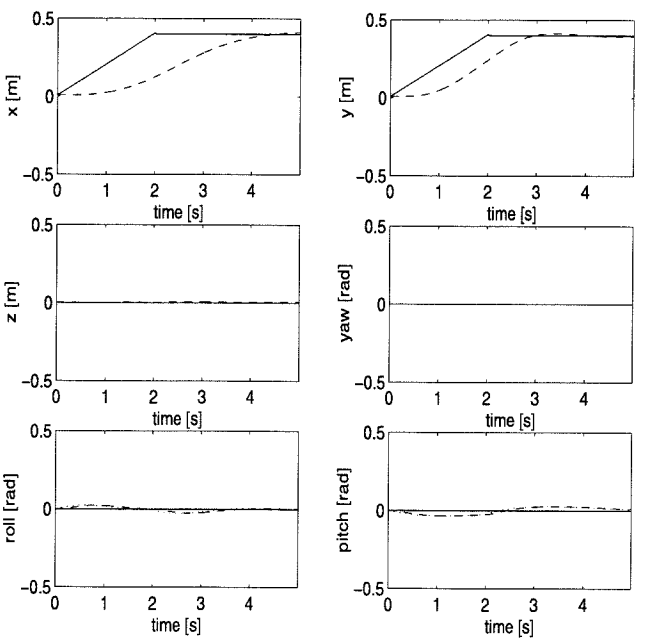


Figure 6.8 Simulation: one degree of freedom controller. Simultaneous step sideways and forward. Solid: nominal, dotted: commanded, dashed: experimental.

6.7 Hover Control Design

We transformed the model (??) to continuous time with a Tustin a method, and we designed an $\mathcal{H}^\infty$ controller for roll, pitch and yaw. The structure of the $\mathcal{H}^\infty$ controller is depicted in Figure ??.

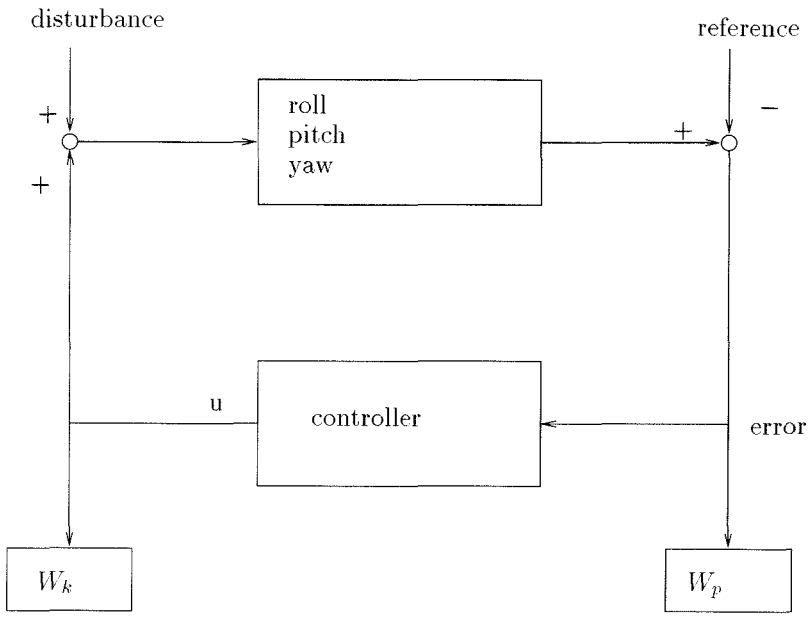


Figure 6.9 Structure of the $\mathcal{H}^\infty$ controller for the attitude dynamics of the helicopter.

The weighting functions W_k and W_p are 3×3 diagonal transfer functions that penalize control effort at high frequencies and tracking error at low frequencies respectively. The diagonal entries of W_k and W_p have the form

$$\begin{aligned} W_{ki} &= K_{ki}\frac{s+f_{ki}}{s+100f_{ki}} \\ W_{pi} &= K_{pi}\frac{s+100f_{pi}}{s+f_{pi}} \end{aligned} \tag{6.24}$$

where the values of the coefficients are tabulated in Table ??.

Our $\mathcal{H}^\infty$ design was not able to eliminate an unstable coupling between roll and yaw without sacrificing yaw performance. We therefore conducted a separate loop shaping design for the SISO system with rudder as input and yaw as output. This eliminates the feedback from roll to yaw. This system is obtained by taking the appropriate entries from model (??).

We tested the disturbance rejection properties of this controller by tapping on the frame. The results are shown in Figure ??, where the times at which the disturbances were applied are marked with vertical lines. The step responses for this controller are shown in Figure ??. It can be seen that the speed of the response

angle	K_{ki}	f_{ki}	K_{pi}	f_{pi}
roll	200	6	0.1	4
pitch	200	6	0.05	4
yaw	200	4	0.01	4

Table 6.1 Parameter values for the weighting functions in (??)

is not as tight as might be hoped for.

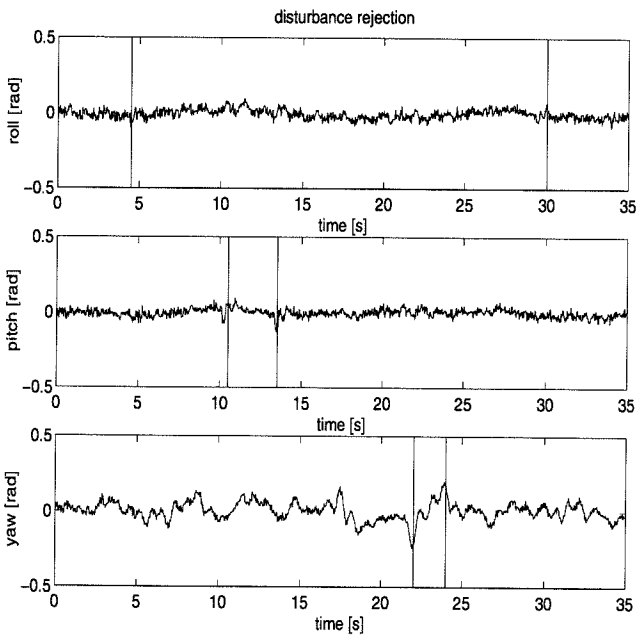


Figure 6.10 Experiment: disturbance rejection with LQR/LS controller. Disturbances indicated with vertical lines.

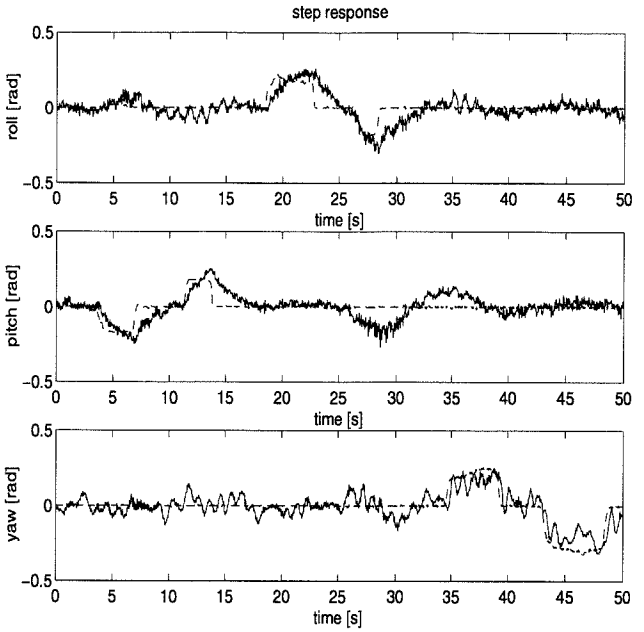


Figure 6.11 Experiment: step responses with LQR/LS controller. Reference signal is dashed, measured response is solid.

6.8 Experiments

For safety reasons, the helicopter is mounted on the stand depicted in Figure **??**, and the range of roll, pitch and yaw rotation and lateral translation is restricted by strings. The rotor spins at about 1200 rpm, and when it hits something, the helicopter crashes, usually damaging the blades and the tail boom. Unfortunately, the stand dynamics are not negligible. The lower arm link adds to the weight, exceeding the maximum payload of the helicopter. We mount springs at the elbow joint to compensate for some of this added weight. Since the springs contract as the helicopter takes off, the buoyancy compensation decreases with altitude, resulting in an increased weight at higher altitudes. The friction in the elbow and shoulder joints turned out to be bigger than the drag for low lateral velocities. The helicopter then wants to rotate about the elbow joint only, since rotation about the shoulder joint constitutes a pure addition of friction. This translates into a preferred direction of lateral motion on the circle prescribed by the lower arm with the elbow joint pinned. This can be remedied by restricting the helicopter to level flight with the wrist resting on the floor. By extending the springs we can increase the effective weight of the helicopter beyond its maximum payload. The friction of the wrist on the floor will then dominate the friction of the elbow and shoulder joint, so that there is no longer a preferred direction of lateral motion.

We found that the vertical and yaw dynamics can be controlled independently from the horizontal and pitch-roll dynamics. This is since we have 2 independent

controls for heave and yaw: namely the collective pitch and rudder respectively.

We therefore restrict attention to (x, y) translation induced by (pitch,roll) rotation in this section, effectively resulting in 5 degree of freedom motion: 3 angular degrees of freedom and x and y translation.

We compare the two degree of freedom design with the one degree of freedom design for a step in the y (sideways) direction in Figures ?? and ?? respectively. The nominal trajectory computed from the flat system (??) is plotted as a solid line. The nominal trajectory for the inner system (the roll and pitch dynamics) plus the controller correction from the outer system is plotted as a dotted line (this is the desired trajectory $\bar{\bar{y}}_2$ in Section ??). The measured position is plotted as a dashed line.

We see that the response for the two degree of freedom design is slightly less oscillatory, but the improvement is not substantial. Both designs leave a large steady-state error, due to the stiction of the wrist on the floor. We think the poor performance is due to the slowness of the roll and pitch dynamics, as could be seen from Figure ??. The helicopter cannot follow the commanded pitch and roll commands. Also the friction in the joints and the stiction from the floor represent unmodelled dynamics. One might suggest to increase the time interval over which the step is commanded, to decrease the frequency content. This has the undesired effect of decreasing the amplitude of the nominal roll and pitch command, making the two degree of freedom design indistinguishable from the one degree of freedom design. Another possibility to maintain amplitude with decreasing frequency content, is to use larger values for the inertial masses than for the gravitational mass. Just as for the ducted fan, it is easy to see that this does not disturb flatness of Equation (??). This solution is totally ad hoc and will not be pursued here.

Figures ?? and ?? show the two and one degree of freedom respectively for a step in the x (forward) direction. The same conclusions hold as for the sideways step.

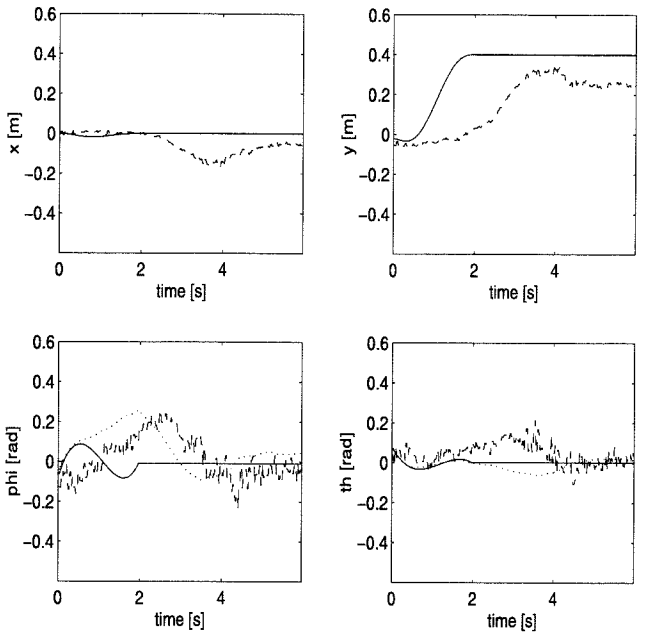


Figure 6.12 Experiment: two degree of freedom controller. Step sideways. Solid: nominal, dotted: commanded, dashed: experimental.

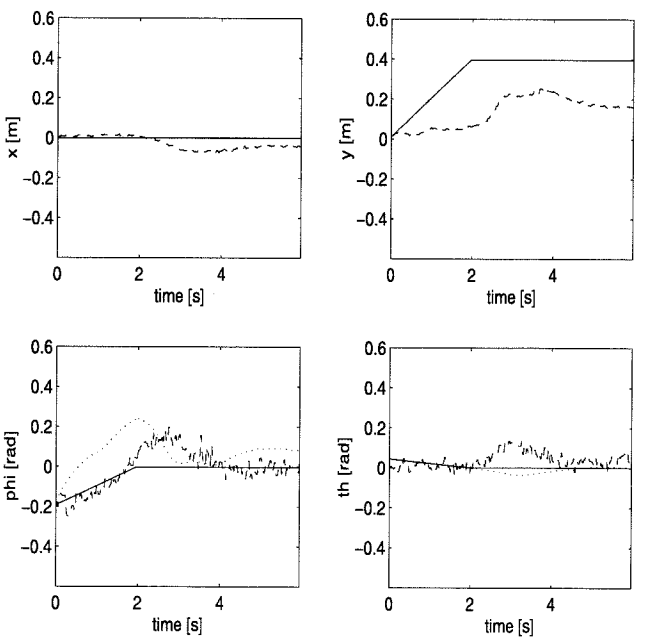


Figure 6.13 Experiment: one degree of freedom controller. Step sideways. Solid: nominal, dotted: commanded, dashed: experimental.

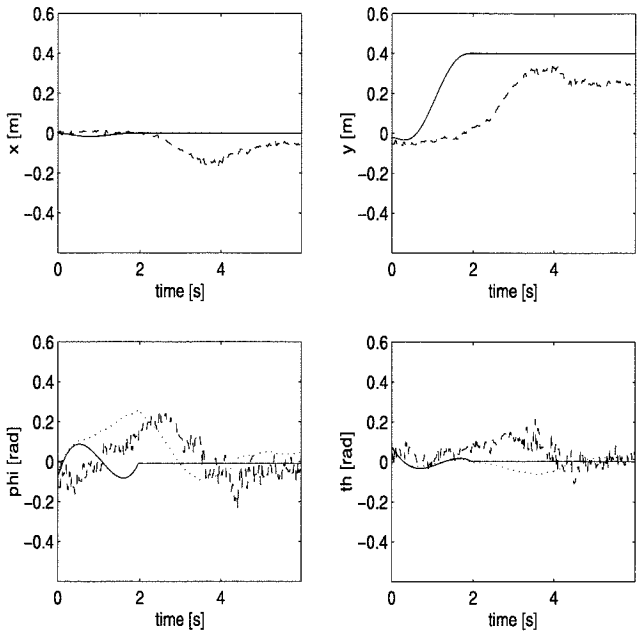


Figure 6.14 Experiment: two degree of freedom controller. Solid: nominal, dotted: commanded, dashed: experimental. Step forward.

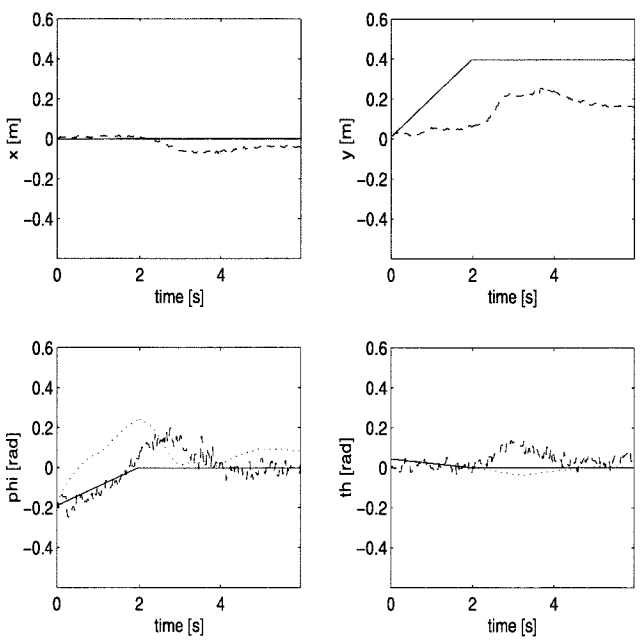


Figure 6.15 Experiment: one degree of freedom controller. Solid: nominal, dotted: commanded, dashed: experimental. Step forward.

6.9 Conclusions

In this chapter we introduced the notion of *outer flatness* as a two-layered system structure with a flat outer system and a non-flat inner system. We compared outer flatness with the existing methodologies of backstepping and dynamics inversion. We proved two theorems expressing the conditions for exponential tracking and bounded tracking of the total system, assuming exponential tracking for the outer system n both cases, and exponential and bounded tracking for the inner system, respectively. The conditions of the theorems are cumbersome to check, but they show that there are no theoretical obstructions to the application of outer flatness.

We applied outer flatness to the Caltech model helicopter experiment, for which we performed linear ID and hover controller design. Simulations showed that outer flatness is a valid approach, although the experiments showed that there is not much gain over a one degree of freedom approach, due to slowness of the inner dynamics.

Chapter 7

Conclusions

7.1 Summary

The theoretical foundation for differential flatness in terms of exterior differential systems is given in Chapter 2. This chapter also proves some theorems on flat systems in the geometric framework. In particular, it is shown that flatness is equivalent to feedback linearization in an open and dense set. A complete characterization of flatness for single input systems is given. It is shown that for single input time independent systems we can always take the flat inputs independent of time. The chapter also gives examples of flat systems.

Chapter 3 presents some important trajectory generation problems for differentially flat systems and algorithms to solve them. It presents the point-to-point steering problem, the least squares approximation problem, and the cost minimization problem. It also presents the software library that implements these algorithms, and analyzes the computational complexity of the algorithms and indicates typical computation times. The software is demonstrated through simulations and experiments.

Chapter 4 presents the real time trajectory generation problem, and two algorithms to solve it for differentially flat systems. One is based on repeated point-to-point steering, with a receding horizon destination, the other one is based on additional cost minimization. Again, simulations and experiments validate the algorithms.

Chapter 5 presents some extensions to deal with perturbations to flatness, and validates these in experiment and simulation. The extensions are illustrated with the problem of mode switching for a thrust vectored aircraft. Mode switching involves a strongly nonlinear transition between substantially different flight regimes. It is concluded that steering to the right trim condition for the aircraft is crucial, but compensating for the perturbation by absorption in the input is not. The results are explained with a Lyapunoff argument.

In Chapter 6 we define outer flatness as a two layered structure where the outer layer is flat, and the inner is not. Outer flatness is compared with the similar concepts of backstepping and dynamic inversion. We present two theorems on the type of tracking (bounded or exponential) achievable based on tracking properties

of the inner and outer system and additional conditions. We present the helicopter experiment as a test case for outer flatness.

We reiterate some of the philosophical points of this dissertation. A recurring theme in this work is its emphasis on software tools and experimental validation. New theory in engineering disciplines needs software tools to be accessible to the engineering community. Without software tools, much potentially useful theory just dies on the shelves. For linear control, much research is directed toward the development of good software. For nonlinear control, this effort is largely absent. In this thesis we undertook the development of a software library for trajectory generation problems for differentially flat nonlinear systems. Much work remains to be done, mainly to extend this library to nonflat systems. By the same token, new theory in engineering disciplines needs to be validated in experiments. Actual implementation of theory is an objective test of its usefulness, and implementation problems are important indications for rewarding directions of research. This dissertation systematically tested new theory on experiments.

7.2 Future Research

Most of this dissertation focuses on differentially flat nonlinear systems or approximations thereof. As of yet, no general test for differential flatness exists. Therefore, an obvious direction for future research is the formulation of such a test. This is no easy task, since it was shown in Chapter 2 that differential flatness is equivalent to dynamic feedback linearizability in an open and dense set, and many excellent researchers have devoted much effort to finding a test for dynamic feedback linearizability without much success. However, see [?] for a special case. It is the belief of this author that a more promising approach is to find approximating flat systems and use the techniques described in Chapters 5 and 6.

Another approach to trajectory generation is the development of fast code that solves the two point boundary value problem resulting from optimal control. Some promising work has been done by [?, ?, ?, ?], using sparse matrix algebra and collocation (direct) methods. This allows solving the trajectory generation problem in its full generality. It remains to be seen to what extent this is amenable to real time implementation. The iterative character of the collocation method seems to lend itself well to real time implementation. The grid of collocation points can be made fine for the immediate future, and coarser for the more remote future, constantly being updated as time moves along and new trajectory input becomes available. We believe that this offers the most potential for two degree of freedom design.

An issue not explicitly addressed here is the incorporation of uncertainty in the trajectory generation stage. Some researchers, [?, ?, ?], have studied the robustness analysis of nonlinear systems along a given trajectory, but the synthesis problem remains untouched.

Appendix A

Basics of Nonlinear Geometric Control Theory

Arguably the most popular approach to the control of nonlinear systems is provided by feedback linearization [?, ?]. The theory of feedback linearization is well known, but will be repeated here for completeness and future reference. The system under consideration is

$$\begin{aligned} \dot{x} &= f(x) + g(x)u \\ y &= h(x) \end{aligned} \tag{A.1}$$

where $x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$, $y \in \mathbb{R}^m$. The function f and g are assumed to be C^∞ and the matrix g is full rank. Note that the system is *affine*, i.e. the right-hand side is linear in the input. Non-affine systems can be transformed into this form by adding integrators to all inputs. The geometric properties of the extended system are identical to the original system. Note also that the above system is *square*, i.e. the number of inputs equals the number of outputs.

Nonlinear control theory borrows many tools from differential geometry. The *Lie derivative* of a function h with respect to a vector field f is the function

$$L_f h = \frac{\partial h}{\partial x}^T f. \tag{A.2}$$

The Lie derivative of a vector field g with respect to a vector field f is the vector field

$$L_f g = \frac{\partial g}{\partial x} f. \tag{A.3}$$

The k-th Lie derivative of a function h with respect to a vector field f is defined recursively as the function

$$L_f^k h = L_f L_f^{k-1} h, \tag{A.4}$$

and similarly for 2 vector fields. The *Lie Bracket* of two vector fields f and g is the vector field

$$[f, g] = \frac{\partial g}{\partial x} f - \frac{\partial f}{\partial x} g = L_g f - L_f g. \tag{A.5}$$

The system (**??**) is said to have well defined *vector relative degree* γ at a point x_0 if there exists a vector of integers $\gamma = (\gamma_1, \ldots \gamma_m) \in \mathbb{N}^m$ such that the *decoupling matrix*

$$B_{ij} = L_{g_j} L_f^{\gamma_i - 1} h_i(x) \tag{A.6}$$

has full rank m at x_0, and $L_{g_j} L_f^k h_i(x) = 0$ for $i = 1, \ldots, m$, $j = 1, \ldots, m$, $k = 0, \ldots, \gamma_i - 2$ for all x in a neighborhood of x_0. If the system has well defined vector relative degree it takes the form

$$\begin{aligned} \begin{pmatrix} y_1^{(\gamma_1)} \\ \vdots \\ y_m^{(\gamma_m)} \end{pmatrix} &= \begin{pmatrix} L_f^{\gamma_1} h_1 \\ \vdots \\ L_f^{\gamma_m} h_m \end{pmatrix} + B(x)u =: a(x) + B(x)u \\ \dot{\eta} &= p(x) + q(x)u. \end{aligned} \tag{A.7}$$

Applying the feedback transformation

$$u = B^{-1}(x)\big(-a(x) + v\big), \tag{A.8}$$

and state coordinate transformation

$$(\xi, \eta) = \psi(x), \tag{A.9}$$

where $\xi = (y_1, \ldots, y_1^{(\gamma_1 - 1)}, \ldots, y_m, \ldots, y_m^{(\gamma_m - 1)})$, puts part of the system in linear form:

$$\begin{aligned} \dot{\xi}_1^1 &= \xi_1^2 \\ \dot{\xi}_1^2 &= \xi_1^3 \\ &\vdots \\ \dot{\xi}_1^{\gamma_1} &= v_1 \\ &\vdots \\ \dot{\xi}_m^1 &= \xi_m^2 \\ &\vdots \\ \dot{\xi}_m^{\gamma_m} &= v_m \\ \dot{\eta} &= p(\eta, \xi) + q(\eta, \xi)v \\ y_i &= \xi_i^1 \end{aligned} \tag{A.10}$$

where $\dim(\eta) = n - \sum_i \gamma_i$. If $\sum \gamma_i = n$ the system (**??**) is *feedback linearizable* by static state feedback. This means that we can transform the system into a linear system by a coordinate transformation and a feedback transformation. If $\sum \gamma_i < n$ then the remaining dynamics for η are called the *zero dynamics*, or the *internal dynamics* . We call a system *minimum phase* if the zero dynamics are

asymptotically stable. This term originates in linear systems theory, where stability of the zero dynamics is equivalent to all zeros being in the left half plane, resulting in the minimum phase for a given magnitude of the transfer function. If the zero dynamics are not asymptotically stable, the system is called *non-minimum phase*. Sometimes the term *maximum phase* is used for systems whose zero dynamics are minimum phase in reverse time, i.e. the linearization has all zeros in the right half plane.

Let $Y_{dN}(t) = \{y_d(t), \ldots, y_d^{(N)}(t)\}$ be a class of desired trajectories and their derivatives up to some order N. Let Q be a compensator of the form

$$\begin{aligned} \dot{z} &= a(x, z, Y_{dN}) \\ u &= \alpha(x, z, Y_{dN}). \end{aligned} \tag{A.11}$$

We say that a control law Q achieves *exact tracking* of trajectories in Y_{dN} for the system **??** if $y(t) - y_d(t) = 0$ for all t for some initialization of the controller state z. The control law achieves *asymptotic tracking* for the system if $\lim_{t\to\infty}(y(t) - y_d(t)) = 0$ for all $y_d \in Y_{dN}$ and the equilibrium $(0, 0)$ of the unforced system

$$\begin{aligned} \dot{x} &= f(x) + g(x)\alpha(x, z, 0) \\ \dot{z} &= a(x, z, 0) \end{aligned} \tag{A.12}$$

is asymptotically stable. These definitions for exact tracking and asymptotic tracking are taken from [**?**]. We say an algorithm achieves *asymptotic trajectory generation* for a class of signals Y if the algorithm generates from $y_d \in Y$ a feasible full state and input trajectory (x_d, u_d) such that $\lim_{t\to\infty} h(x_d(t)) - y_d(t) = 0$ for all $y_d \in Y$.

We can transform a desired trajectory to the new ξ coordinates, and implement a linear tracking control that guarantees asymptotic tracking of the desired trajectory:

$$v_i = \xi_{id}^{\gamma_i} + \sum_{j=1}^{\gamma_i - 1} c_j(\xi_i^j - \xi_{id}^j) \tag{A.13}$$

with the polynomials $\lambda^{\gamma_i} - \sum c_j \lambda^j$ Hurwitz for all i. If the zero dynamics are stable, internal stability of the total system is guaranteed. In that case, the error system is asymptotically stable around the origin, hence we achieve asymptotic tracking for all trajectories.

Grizzle et al. [**?**] present a fundamental result on the necessity of stability of the zero dynamics for asymptotic tracking. We repeat this result here for completeness.

Theorem A.1 (Grizzle et al., [?]) *Given a system of the form* **??** *satisfying the following conditions:*

1. *The system is analytic.*
2. *The system possesses a zero dynamics manifold.*
3. *The system is left invertible.*

4. The system has a controllable linearization.

and let $Y(\epsilon, N) = \{y(t) \mid \|y(t)\| \leq \epsilon, \ldots, \|y^{(N)}(t)\| \leq \epsilon, \forall t\}$. *Then a necessary condition for asymptotic tracking of signals in* $Y(\epsilon, N)$ *for any* N *and* ϵ *is that the system have asymptotically stable zero dynamics.*

Note that this theorem shows that we cannot achieve asymptotic tracking even by decreasing the magnitude of the desired outputs and their derivatives. To achieve asymptotic tracking we need to relax either the analyticity requirement, or reduce the set of desired trajectories, or resort to some approximate scheme.

Recall (see [?]) that a left inverse of a system Σ is a system Σ_L (the *left inverse*) that reconstructs the unique input that leads to a given output of Σ, given that output and the initial state. The dual problem of finding a (possibly non-unique) input to Σ that produces a desired output of Σ given that output and the initial state is called *right inversion*. A system Σ_R that performs this inversion is called a *right inverse*.

Recall (see [?]) that a *zero dynamics manifold* for the system ?? is a C^1 manifold Z^* of $\mathbb{R}^n$ containing the origin and satisfying

1. $Z^* \subset h^{-1}(0)$.
2. $f|_{Z^*} \subset TZ^* + span\{g\}$.
3. Z^* is locally maximal with respect to 1) and 2).

The system has a *zero dynamics* if in addition there exists a *unique* function α^* such that $f^* := f + g\alpha^*|_Z$ is tangent to Z, α^* is C^1 and $\alpha^*(0) = 0$.

If the zero dynamics are not asymptotically stable, we can find a trajectory in an arbitrarily small ball around the origin of the state space that we cannot track asymptotically. Intuitively this can be understood by realizing that this trajectory excites the zero dynamics, and the system will have to give up tracking to maintain internal stability. If we are balancing a broom stick on our hand, it is clear that we cannot follow arbitrary prescribed motions with our hand. Sometimes we have to maneuver to keep the broom stick balanced, resulting in loss of tracking.

The relevance of Theorem ?? for trajectory generation is born out by the fact that trajectory generation combined with a linear controller K based on the Jacobi linearization of the plant will achieve asymptotic tracking of signals in $Y(\epsilon, N)$ for N large enough and ϵ small enough. This follows from Lemma 4.5 in [?] and the fact that the higher order terms in the error system for $x - x_d$ are uniformly Lipschitz in time for desired signals in $Y(\epsilon, N)$. Hence asymptotically stable zero dynamics are also necessary for real time trajectory generation, unless we relax the conditions of Theorem ?? somehow.

Appendix B

LIBTG: C-routines for Trajectory Generation for Flat and Approximately Flat systems. Version 1.0

B.1 Introduction

This appendix describes a set of libraries in C for the generation of trajectories for flat and approximately flat systems. The code is available as a gzipped tarfile through anonymous ftp from `avalon:/pub/vannieuw/software/trajgen.tar.gz`. There are three main libraries. The first one, `libnr2.a`, is a modified subset of routines from the Numerical Recipes in C. . They are modified for speed and pseudo real-time execution. The second one, `libmatrix2.a` is a set of matrix routines. The third one, `libtg.a` contains the proper trajectory generation routines. The application of the software in trajectory generation is explained in the paper [?], and in Chapter 3 and 4.

The functions in `libtg.a` have the format `TG_nn_mm()` where `nn` is a problem class, and `mm` is the particular function within that problem class. Example: `TG_cost_compcoeff()` would compute the coefficients in a cost minimization problem.

Each data structure `dd` has associated with it the following functions:

```
TG_dd_create(dd, pars)
TG_dd_free(dd)
TG_dd_print(dd, text)
```

which create, release and print the data structure respectively.

The directory structure is as follows:

```
/trajgen
   /include          : header files
   /lib              : all  libraries
   /src
      /nr            : source code for Numerical Recipes
      /matrix        : source code for matrices
      /tg            : source code for trajectory generation
```

```
/examples          : examples of physical systems
   /trajdata       : trajectory data files
   /util           : utilities for testing functions in the examples
/matlab            : m-files for displaying data
/doc               : documentation
```

The `examples` directory contains examples of a kinematic car, a ducted fan engine with full position information, and the ducted fan engine where we ignore the horizontal position coordinate. The subdirectory `util` contains routines to test some of the subroutines related to the physical modelling of the above examples. It makes a table with the lift and drag coefficients for the ducted fan for instance. The directory `matlab` contains some plotting and analysis routines to display the data in `examples/trajdata`.

You need to enter a lot of data particular to your problem to use these routines. Rather than entering everything through endless parameter lists, I kept some important variables global. It will be clear from the examples which variables you need to update to tailor the routines to your problem. You are encouraged to look at the source code and add features where you need them. Consider the code offered here as a framework you need to fill in, and can modify, much like the Numerical Recipes. Re-compilation will be trivial with the provided makefiles.

The code is infested with `if(verbose>n)printf(....);` to print interesting statistics using the matrix print functions. These are left in on purpose, for your convenience. It is true that the `if()` statements entail a performance penalty, but this penalty is small and can be completely avoided by using `#ifdef VERBOSE ...#endif` preprocessor statements around the appropriate code fragments.

B.1.1 Notational Conventions

File names, function names and variable names are in `typewriter font`. The notation `1..n` indicates a range of integers from `1` to `n`. We indicate the dimensions in the order (number of rows, number of columns) of a matrix A by `size(A) = (nr, nc)`.

B.1.2 Acknowledgments

Funding from NASA and AFOSR is gratefully acknowledged. Thanks to Michael Kantner for writing the trajectory data structure and trajectory functions code.

B.2 Numerical Recipes in C$^\flat$

This section describes a modification to a subset of the Numerical Recipes in C [?]. Please buy a legal copy of the Numerical Recipes to use these modified routines. The following subsections describe the modifications.

B.2.1 Caveats

The routines still use array indexing from `1..n`. I hate this as much as the next guy, but it is too much of a pain to chase through all the changes.

B.2.2 Floats Replaced by Doubles

All floats are replaced by doubles.

B.2.3 Array Allocation

All memory allocations for arrays are done at the first call of the routine, or in subsequent calls when the parameter indicating array size has increased. This is done by declaring a `static` integer indicating the size which is initialized to 0. All arrays are also declared as `static`. Most of the Numerical Recipes routines are called many times in a row with the same size parameter. The above procedure therefore saves a considerable amount of time. It also enables the routines to be run in real-time. The disadvantage is that at any one given time the memory requirements are bigger: all arrays coexist simultaneously in memory. There is no way around this tradeoff between memory and computation time. For our application, speed is more important. The user has to make sure all memory is allocated in a background process before the routine is called real-time. This is achieved easily by calling the routine once with appropriate size parameter.

We thought about writing for each routine a companion routine that would just allocate the necessary memory, or passing a flag argument to all routines that would make the routine allocate memory and then return. This means the user would have to track down the calling hierarchy of all Numerical Recipes routines. After some testing we found that this was tedious. The current setup allows the user to call the routine with the exact same arguments as in real use, without the headache of tracking down the internal working of the routine. We judged this preferable to the first approach described above.

B.2.4 Maximum Number of Iterations

The Numerical Recipes define a constant `MAXITER` at the beginning of all routines of iterative nature. It would indiscriminately abort the entire program to the operating system if `MAXITER` was exceeded. The modified routines pass `MAXITER` as a parameter. If `MAXITER` is exceeded, the routine will always return the best solution found so far, and call the function `nrsofterror()`. This routine is by default one that prints an error message to `stderr`, but can be redefined by the user to an appropriate alarm call. It should be redefined if the code is to be called in real-time routines, since `printf()` statements are too slow to be executed in an interrupt loop.

B.3 Matrices and Trajectories

B.3.1 Introduction

There are several data structures defined to store information. The more complex ones pertaining specifically to trajectory generation will be introduced in later sections. In this section we introduce utilities to deal with matrices and trajectories.

B.3.2 Matrices

A matrix is simply a pointer to a column of pointers. Each pointer in the column points to a row of the matrix. A more sophisticated matrix data structure would have the dimensions as members and a flag for complex or real data. For compatibility with Numerical Recipes, we abstain from this.

Many matrix libraries have been written, and they all do a standard set of operations, plus more or less extra fancy ones. It seems ridiculous to have yet another set of matrix routines for this library, but we need to to maintain compatibility with the Numerical Recipes. The matrix library uses the Numerical Recipes library described above for inversion and singular value decomposition. Indexing is therefore still from `1..n`, for compatibility with this library. The standard routines include matrix and vector allocation, addition, subtraction, multiplication, scaling, transposition, dot products. The more fancy routines are described below. See `matrix2.h` for a complete listing of the functions.

Pseudo Inverse

We compute pseudo inverses with the routine

```
pseudo_inv(double **Adest, double **A, int nr, int nc)
```

which will dump the pseudo inverse of `A` into `Adest`. `size(A) = (nr, nc)`. It is smart enough to decide between $Adest = A^*(AA^*)^{-1}$ if `nr<nc`, and $Adest = (A^*A)^{-1}A^*$ if `ncr>nc`. It returns $Adest = A^{-1}$ if `nc = nr`. Allocates memory for storage arrays.

Projection Matrices

We can compute a projection matrix with the routine

```
int project_mat(double **P, double **A, int nr, int nc)
```

which computes the matrix $P = (I - A^*(AA^*)^{-1}A)$ if `nr<nc`, and returns with exit code 1. If `nr>nc` it sets $P = 0$, and returns with exit code -1. Projection matrices are useful in constrained optimization: in that case we need to project the gradient onto our constraint manifold at every step.

Projection of Vectors

The function

```
int project_vec(double *y, double *x, double **A,
    int nr, int nc, double *b)
```

computes $y = Px + A^{\dagger}b$, where $P = (I - A^{*}(AA^{*})^{-1}A)$ is the projection matrix associated with `A`. This is the projection of x onto the plane `Ax = b`. This functionality is useful for constrained optimization. `size(A) = (nr, nc)`. If `nr<= 0` there are no constraints, `y = x`, and the return code is 1. If `nr>nc`, there are more constraints than free variables, `y = 0` and the return code is -1. A faster version of this routine is

```
int project_vec_f(double *y, double **P, double *x,
    int nra, int nca, double **Aps, double *b, int nrb, int ncb)
```

with the projection matrix `P` and the pseudo inverse `Aps` already precomputed. `size(P) = (nra, nca)`, `size(Aps) = (nrb, ncb)`. It effectively computes `y = P.x + Aps.b`. This is useful since we usually need to compute the projection and pseudo inverse for other purposes too, so we might just as well use them here.

Nullspace

The nullspace of a matrix can be computed with

```
int nullspace(double **Fnull, double **F, int nr, int nc,
   int nulldim, double svdeps)
```

which writes the nullspace of `F` in `Fnull`. `size(F) = (nr, nc)`. It does this with a singular value decomposition. If `svdeps > 0.0` it will compute the eigenvectors corresponding to singular values less than `svdeps`, and return 1. If `svdeps < 0.0` and `nulldim > 0` it will compute the eigenvectors corresponding to the `nulldim` smallest singular values, and return 2. If both `svdeps < 0.0` and `nulldim <= 0` it will return -1.

Copying Matrices

Basic matrix copying is done with

```
int copy_dmatrix(double **matdest, double **matsrc, int nr, int nc)
```

which will copy matrix `matsrc` into `matdest`. `size(matsrc) = (nr, nc)`.

It is often convenient to compute a small matrix and copy it to some location in a bigger matrix. This can be done with

```
int copy_dmatrix2(double **matdest, double **matsrc, int r0, int r1,
     int c0, int c1)
```

which will copy matrix `matsrc` to location `(r0, c0)` in matrix `matdest`. `size(matsrc) = (r1-r0+1, c1-c0+1)`. Returns `(r1-r0+1)(c1-c0+1)`, the number of elements copied.

Basic vector copying is done with

```
int copy_dvector(double *v1, double *v2, int n)
```

which copies vector `v2` to vector `v1`. Returns `n`, the number of elements copied. `size(v1) = (n)`.

Copying to an arbitrary location can be achieved with

```
int copy_dvector2(double *v1, double *v2, int r0, int r1)
```

which copies vector `v2` to location `(r0)` in vector `v1`. `size(v2) = (r1-r0+1)`. Returns `r1-r0+1`, the number of elements copied.

B.3.3 Trajectories

The trajectory code has been written by Michael Kantner, with some minor additions by me. A trajectory is basically an array, where time indexes the rows, and states and inputs index the columns.

```
typedef struct trajectory_data {
  double *time;
  double **data;
  int current;
  int nrows;
  int ncols;
  int flag;
} TRAJ_DATA;
```

Here, `time` contains the time point, `data` contains the trajectory data, `nrows` and `ncols` indicate the dimension of `data` (NOTE: `time` is not included in the number of columns count). `current` is a pointer to the last evaluated trajectory point. See `traj.h` for a list of all the utilities operating on trajectories.

B.4 Basis Functions

We expand trajectories in a basis, and perform calculations on the coefficients of those basis functions. A `BASIS` is a structure

```
struct basis_struc{
    double (*func)(int order, int der, double arg,
                   struct basis_struc *);
    int numbasfun;
    double t0, t1;
    double **mat;
    void *custom;
};
```

where `func` return the value of the `der`-th derivative of basis function number `order`, at time `arg`, and has access to the parameters of the basis through the last argument. `numbasfun` is the number of basis functions in the basis, `t0, t1` are the initial and finite time of the interval on which the basis is defined, `mat` is a matrix of coefficients, the meaning of which depends on the basis functions, `custom` is an arbitrary pointer, the meaning of which depends on the basis.

A *regular basis* is a set of polynomials with zeros equally spaced over the basis interval. To initialize a basis of regular polynomials, use

```
BASIS *TG_bas_initregular(int numbasfun)
```

which will fill the matrix in the `BASIS` with the coefficients of the polynomials, each row representing a polynomial. Currently, only the regular basis is implemented. Feel free to add more bases and send me the code.

The function

```
int TG_bas_free(BASIS *basis)
```

releases the memory taken by the `BASIS basis`. The function

```
int TG_bas_print(BASIS *basis, char *text)
```

prints some statistics on a `BASIS`. The function

```
int TG_bas_setinterval(BASIS *basis, double t0, double t1)
```

sets the initial and final time on the interval, i.e. `basis->t0` and `basis->t1`.

Given a basis, we want to evaluate several function values, or vectors of function values. The function

```
double TG_bas_evalpoint(BASIS *basis, double x, int nder,
        double *coef)
```

evaluates the `nder`-th derivative of a function written with respect to a basis at a point. The coefficients of the function are stored in `coef`. So it returns $\sum_i a_i \phi_i^{\text{nder}}(x)$.

If we want to evaluate a function value and its derivatives at a point, we use the function

```
double TG_bas_evalflag(BASIS *basis, double x, int maxder,
        double *coef, double *flag, int *numbasfun)
```

which evaluates the 0-th through the `maxder`-th (inclusive) derivative of a function expanded in basis `BASIS` at a point `x` and puts the result in `flag`: `flag[i]` = $\sum_j a_j \phi_j^{(i)}(x)$.

To get the values of all separate basis functions and their derivatives at a point, we can use the function

```
int TG_bas_evalmat(BASIS *basis, double x, int maxder,
        double **mat)
```

which evaluates a matrix of derivatives of basis functions in the basis `BASIS` at a point `x` and puts the result in `mat`. `mat[i][j-1]` = $\phi_j^{(i)}(x)$, where `i = 0..maxder`, `j = 1..numbasfun`. Note the reverse order of the indices: a row corresponds to a derivative, a column to a basis function. Note also that the matrix `mat` is indexed as `mat[0..maxder][0..numbasfun-1]`.

B.5 Flat Systems

A system is *flat* if we can find output functions $z(x)$ (equal in number to the number of inputs) such that all states and inputs can be written as a function of z and its derivatives:

$$(x, u) = \phi(z, \dot{z}, ..., z^{(l)}).$$

This means there is a unique correspondence between trajectories in output space and state space. Trajectory generation is greatly facilitated by planning trajectories in the lower dimensional output space, and then lifting them to state space.

B.5.1 Flat Structures

Trajectory generation is easy for flat systems. For a flat system we need to know certain structure constants which we assemble in the structure `FLATSTRUC`.

```
typedef struct{
    int dimoutput,
        maxnumbasfun,
        maxnumders,
        totnumders,
        totnumbasfun;
    int *numbasfun,
        *numders;
    double **zflag;
} FLATSTRUC;
```

Here `dimoutput` is the dimension of the output, `numders` is an array of length `dimoutput` specifying how many derivatives we need to take of each of the outputs to recover the states and inputs, `maxnumders` is the maximum number in this array, `totnumders` is the sum of the numbers in this array. `numbasfun` is an array of length `dimoutput` specifying how many basis functions we want to expand each flat output, `maxnumbasfun` is the maximum number in this array, `totnumbasfun` is the sum of the numbers in this array. `zflag` is an array storing the values of the flat outputs and their derivatives. `size(zflag) = (dimoutput, maxnumders)`.

The function

```
FLATSTRUC *TG_flat_create(int dimoutput, int *numbasfun,
          int *vecreldeg)
```

creates a flat structure with indicated parameters. The entries of `vecreldeg` are copied to `FLATSTRUC.numders`, and `numbasfun` is copied to `FLATSTRUC.numbasfun`. `zflag` is filled with zeros.

We release and print some statistics about a flat structure with

```
int TG_flat_free(FLATSTRUC *flatstruc)
int TG_flat_print(FLATSTRUC *flatstruc, char *text)
```

respectively.

The function

```
int TG_flat_compflag(BASIS *basis, double **coef, double x,
   FLATSTRUC *flatstruc)
```

computes the values for the flat flag in `FLATSTRUC.zflag` at point `x` if the outputs have coefficients `bcoef` with respect to basis `basis`.

The function

```
int TG_flat_setflag(FLATSTRUC *flatstruc, double *flag)
```

sets the flat flag `FLATSTRUC.zflag` to the stacked values in the vector `flag`. It figures out where to split the vector over the different outputs by the information in `flatstruc`. The function

```
int TG_flat_extractflag(FLATSTRUC *flatstruc, double *flag)
```

does the reverse: it pulls out the values in `flatstruc.zflag` and writes them to `flag`, stacked one output after the other. This is useful since we want to do matrix multiplication on the vector of flat outputs.

It is often necessary to string out the values in a flat flag in to a single vector for matrix operations, and vice versa, for storage purposes. This can be done with the function

```
int TG_flat_flatten(FLATSTRUC *pflatstruc, double *coef1,
    double **coef2)
```

takes the values in `coef2` and flattens them out into `coef1` according to the structure `pflatstruc`.

The function

```
int TG_flat_stack(FLATSTRUC *pflatstruc, double *coef1, double **coef2)
```

does the reverse. It stacks the values in `coef1` into the array `coef2` according to the information in `flatstruc`

B.5.2 Flat Systems

A flat system is a structure

```
typedef struct{
    int diminput,
        dimoutput,
        dimstate,
        dimxstate,
        complevel;
    void (*xu2zfun)(int n, double *xu, double *z, double *pars);
    void (*z2xufun)(int n, double *xu, double *z, double *pars);
    void (*flatpert)(int nx, int nu, double *xflat, double *u, int fcomp);
} FLATSYS;
```

where `diminput, dimoutput, dimstate, dimxstate` are the dimension of the input, output, state and extended state (with dynamic feedback compensator that makes the system flat) respectively. The transformations from (state input) to flat flag, and vice versa are given by `xu2zfun()` and `z2xufun()` respectively. If the system is only an approximation to a flat system, the function `flatpert()` gives the perturbation to flatness, i.e. the term $h(x)$ in $\dot{x} = f(x) + G(x)u + h(x)$ where the nominal system $\dot{x} = f(x) + G(x)u$ is flat. `complevel` is the compensation level for nonflatness. `complevel= 0` means we don't compensate for perturbations to flatness, `complevel = 1` means we project the perturbation $h(x)$ onto the range of $G(x)$ and add the result to the nominal input.

A flat structure is created with

```
FLATSYS *TG_flat_createsys(int diminput, int dimoutput, int dimstate,
        int dimxstate, int complevel, void (*xu2zfun)(),
        void (*z2xufun)(), void (*flatpert)())
```

It can be released with

```
int TG_flat_freesys(FLATSYS *flatsys)
```

If we are given a flat structure, a basis and a set of coefficients, we can compute the time history of the flat outputs over the time interval indicated by the basis. The function

```
int TG_flat_comptraj(BASIS *pbasis,  double **coef2,
    int numtabs, FLATSTRUC *pflatstruc, FLATSYS *pflatsys,
    TRAJ_DATA *ptraj)
```

computes the values of the states and inputs corresponding to the flat outputs at `numtabs` time points equally spaced between `pbasis->t0` and `pbasis->t1`. It uses the mappings between (states inputs) and flat flag given in `pflatsys`.

The mappings between flat flag and state and input can be accessed with

```
int TG_flat_xu2z(FLATSYS *pflatsys, double *xu, double *z,
    double *pars);
int TG_flat_z2xu(FLATSYS *pflatsys, double *xu, double *z,
    double *pars);
```

These functions just call the corresponding members in the structure `pflatsys`.

B.6 Trajectory Generation Routines

We have routines for the following trajectory generation problems:

- Steering from a point to a point.
- Following a trajectory in a least squares sense.
- Following a trajectory while minimizing a cost function in the states and/or inputs.

- Taking real-time pilot input and generating with some delay a full state and input space trajectory.

These 3 problems have a `TG_pp_compcoeff()` routine where `pp = (p2p, lsq, cost)` for each of the problems.

B.6.1 Point-to-Point Trajectories

An important class of trajectory generation problems consists of steering from one point to another. The routines have prefix `TG_p2p`.

The coefficients are computed with

```
int TG_p2p_compcoeff(BASIS *basis, FLATSTRUC *flatstruc0,
    FLATSTRUC *flatstruc1, double **coef)
```

The flat structures `flatstruc0` and `flatstruc1` contain the flat flags at initial and final point indicated by `basis`. The computed coefficients are stored in `coef`. `size(coef) = (dimoutput, maxnumbasfun)`.

Example

Consider the following streamlined code fragment from the file `p2p.c` in the `examples` directory:

```
TG_p2p_init();
for(i=1; i<NUMGOALPTS; i++){

   /* compute zflag(t0) and zflag(t1) : */
   TG_flat_xu2z(pflatsys0, waypoints[i-1], zderinit, pars);

   TG_flat_xu2z(pflatsys0, waypoints[i], zderfinal, pars);

   TG_bas_setinterval(pbasis, waypoints[i-1][0], waypoints[i][0]);

   TG_flat_setflag(pflatstruc0, zderinit+1);
   TG_flat_setflag(pflatstruc1, zderfinal+1);

   TG_p2p_compcoeff(pbasis, pflatstruc0, pflatstruc1, coef2);

   TG_flat_comptraj(pbasis, coef2, numtrajpts[i-1], pflatstruc0,
                    pflatsys0, ptrajxu);
}

traj_save(xutrajdatfile, ptrajxu);

TG_p2p_cleanup();
```

Initialization of the necessary data structures is done in `void TG_p2p_init()`, and they are released in `void TG_p2p_cleanup()`. The necessary parameters initialized in `TG_p2p_init()` are listed in the included parameters file, which can be recognized by the suffix `xx_p2p.par`, where `xx` indicates the physical system.

The example does a repeated point-to-point trajectory generation, using state and time data in an array `waypoints`, and for each stretch using `numtrajpoints[i]` time values. For each pair of points it computes the initial and final flat flag, with `TG_flat_xu2z()`, sets the interval on the basis with `TG_flat_setinterval()`, and writes the computed flat flags to flat structures with `TG_flat_setflag()`. It then calls `TG_p2p_compcoeff()` to compute the coefficients `coef2`, and `TG_flat_comptraj()` to compute the trajectory. Note that this last routine appends the segments to the end of the trajectory, so that the result is a concatenation of all segments. The trajectory is saved to file with `traj_save()`.

B.6.2 Least Squares Trajectories

If we are given a trajectory in flat output space, and want to construct a trajectory in state and input space, we can approximate the trajectory in a basis in a least squares sense, and from the coefficients of the flat outputs compute the values of state and input. The routines have prefix `TG_lsq`.

The coefficients are computed with

```
int TG_lsq_compcoeff(BASIS *basis, FLATSTRUC *pflatstruc,
    TRAJ_DATA *traj, double **coef)
```

The desired trajectory is given in `traj`, the flat structure in `pflatstruc`, and the basis in `basis`. The computed coefficients are stored in `coef`. `size(coef) = (dimoutput, maxnumbasfun)`.

Example

Consider the following streamlined code fragment from the file `lsq.c` in the `examples` directory:

```
TG_lsq_init();

TG_lsq_compcoeff(pbasis, pflatstruc0, ptrajz, coef2);

TG_flat_comptraj(pbasis, coef2, ptrajxu->nrows, pflatstruc0,
         pflatsys0, ptrajxu);

traj_save(xutrajdatfile, ptrajxu);

TG_lsq_cleanup();
```

Initialization of the necessary data structures is done in `void TG_lsq_init()`, and they are released in `void TG_lsq_cleanup()`. The necessary parameters ini-

tialized in `TG_lsq_init()` are listed in the included parameters file, which can be recognized by the suffix `xx_lsq.par`, where `xx` indicates the physical system.

The desired trajectory `ptrajz` is read in `TG_lsq_init()`. The least squares error minimizing coefficients `coef2` are computed in `TG_lsq_compcoeff()`. The trajectory is computed in `TG_flat_comptraj()` and is saved to file with `traj_save()`.

B.6.3 Optimization

We collect the data pertaining to an optimization problem in a structure `OPTSTRUC`.

```
struct opt_struc{
    int minmethod,
        numfixpt,
        numconstr,
        totnumpars,
        numfreepars,
        maxiter;
    double (*costfun)(double *),
           *partsol,
           *zflagcon,
           *z0,
           *z1;
    void (*gradcostfun)(double *par, double *grad);
    double **Fproj,
           **F,
           **Fps,
           **Fnull;
};
typedef struct opt_struc OPTSTRUC;
```

The minimization method is indicated by `minmethod`:

1. Simplex algorithm
2. Simulated annealing
3. Powell's method,
4. Fletcher-Reeves-Polak-Ribiere (conjugate gradient)
5. Davidon-Fletcher-Powell method with Broyden-Fletcher-Goldfarb-Shanno update of pseudo Hessian (variable metric)

See [?] for a detailed treatment of these optimization routines. Methods 4 and 5 need derivative information. `numfixpt` takes on values (0,1,2) indicating if we have no constraints (0), initial time constraints only (1) ot both initial and final time constraints. The total number of constraints is given by `numconstr`, the total number of parameters is `totnumpars`. The number of free parameters is `numfreepars =`

totnumpars - numconstr. The maximum number of iterations allowed in the outer loop of the optimization is maxiter. The function to be minimized is costfun which takes a vector of parameters as an argument. The gradient of the cost function is gradcostfun which takes a vector par of parameters as an argument and fills a vector grad with the gradient. We allow linear constraints of the form zflagcon = F.totnumpars. The pointers z0 and z1 point to the initial and final time part of the constraint respectively. They point within the vector zflagcon. While not strictly necessary, this makes coding less cumbersome. Optimization is performed by finding a particular solution partsol to the constraints and optimizing over the nullspace Fnull of F. It will be convenient to have expressions for the pseudo inverse Fps and the projection matrix Fproj of F.

The function

```
int OPTSTRUC *TG_opt_create(int minmethod, int numfixpt,
    int totnumpars, double (*costfun)(), void (*gradcostfun)(),
    int veclength, int maxiter)
```

allocates an optimization structure with the indicated parameters. All vectors and matrices are initialized to zero.

The function

```
int TG_opt_print(OPTSTRUC *poptstruc, char *title)
```

prints some statistics about an optimization structure, preceded by the string title.

The function

```
int TG_opt_free(OPTSTRUC *poptstruc)
```

releases the space taken by an optimization structure.

The function

```
int TG_opt_setflag(double *z0, double *z1, OPTSTRUC *poptstruc)
```

sets the constraints at the initial time to z0, if poptstruc->numfixpt > 0 and sets the constraints at the final time to z1, if poptstruc->numfixpt > 1. It returns numfixpt in all these cases. If poptstruc->numfixpt > 2 it returns -numfixpt.

B.6.4 Cost Minimizing Trajectories

If we are given a trajectory in output space, but the outputs are not the flat outputs, there is some freedom left in the corresponding state and input trajectory which we may wish to exploit to minimize an additional cost criterion. The routines have prefix TG_cost.

The most important computation is performed by

```
int TG_cost_compcoeff(BASIS *basis, FLATSTRUC *pflatstruc,
     OPTSTRUC *poptstruc, double **coef)
```

which computes the coefficients `coef` with respect to basis `basis` of the flat outputs whose structure is given in `pflatstruc`. The optimization parameters are given in `poptstruc`.

Some additional functionality is provided in

```
int TG_project_coef(BASIS *pbasis, FLATSTRUC *pflatstruc0,
    OPTSTRUC *poptstruc, double **coef2)
```

which will find a particular solution to the constraints in the optimization structure `poptstruc` and write it to `coef2`. The flat structure is given by `pflatstruc0`, and the basis in `pbasis`.

The function

```
void TG_project_2_null(int numfixpt, FLATSTRUC *pflatstruc,
     double **Fproj, double **Fnull)
```

will calculate the null space from the projection matrix by an appropriate selection of the columns of the latter. It turns out that this gives better optimization results than a brute force calculation of the nullspace from `F`, since `Fproject` has a block structure to it corresponding to the different flat outputs, and by picking columns of the projection matrix we preserve this structure.

Since the computation time is linear in the number of trajectory points, we recommend subsampling the trajectory first. This can be done with the routine

```
TRAJ_DATA *traj_subsample(TRAJ_DATA *trjsrc, int n)
```

Example

Consider the following streamlined code fragment from the file `cost.c` in the `examples` directory:

```
TG_cost_init();

if(NUMFIXPT == 1){
    TG_flat_xu2z(pflatsys0, xinit, poptstruc0->zflagcon, pars);
}
if(NUMFIXPT == 2){
    TG_flat_xu2z(pflatsys0, xinit, poptstruc0->zflagcon, pars);
    TG_flat_xu2z(pflatsys0, xfinal, poptstruc0->zflagcon +
                 (poptstruc0->numconstr/2), pars);
}

TG_project_coef(pbasis, pflatstruc0, poptstruc0, coef2);

TG_cost_compcoeff(pbasis, pflatstruc0, poptstruc0, coef2);

TG_flat_comptraj(pbasis, coef2, ptrajxu->nrows, pflatstruc0,
                 pflatsys0, ptrajxu);
```

```
traj_save(xutrajdatfile, ptrajxu);

TG_cost_cleanup();
```

We initialize the necessary structures in `void TG_cost_init()`. They are cleaned up by `void TG_cost_cleanup()`. The necessary parameters initialized in `TG_cost_init()` are listed in the included parameters file, which can be recognized by the suffix `xx_cost.par`, where `xx` indicates the physical system.

Depending on the number of constraints, `NUMFIXPT`, we compute the constraints on the flat flag at the initial and final time, with the function `TG_flat_xu2z()`, and write these constraints to `poptstruc0`. We then compute a particular solution `coef2`, to the constraints with `TG_project_coef()`. This function also computes the projection matrix, the pseudo inverse, and the nullspace corresponding to the constraint matrix and stores them in `popstruc0`. Then we perform a minimization over the nullspace of the constraint matrix with `TG_cost_compcoeff()`. The coefficients `coef2` are used to calculate the trajectory with `TG_flat_comptraj()` which is saved to file with `traj_save(xutrajdatfile, ptrajxu)`.

B.6.5 Real-Time Trajectory Generation

These routines perform the computations in [?] for real-time trajectory generation. We collect the data pertaining to an optimization problem in a structure `RTSTRUC`.

```
struct rt_struc{
   int zlength,
       Tdel,
       dimoutput,
       ydesptr,
       rttype;
    double tinit,
           tnext,
           tfinal,
           Ts,
           *xinit,
           *xfinal,
           **ydes,
           **Art,
           **Brt;
};

typedef struct rt_struc RTSTRUC;
```

Here `zlength` is the total length of the flat flag, `Tdel` is the number of samples delay in the trajectory generation, `dimoutput` is the dimension of the output, `ydesprt` points to the most recent sample in the array `ydes` that stores the pilot input and the corresponding time over `Tdel+1` samples. `size(ydes) = (Tdel+1, dimoutput+1)`.

`rttype` indicates the type of real-time problem: `rttype = 1` for real-time point-to-point steering (algorithm 1 in [?]), `rttype = 2` does additional cost minimization (algorithm 2 in [?]), `rttype = 3` does not generate a feasible (state, input) trajectory put feeds the pilot input through directly. The initial time of the interval over which the trajectory is computed is `tinit`, the initial state and input are in `xinit`. At the end of a step in the algorithm, `xinit` will contain the desired nominal state and input for the next sampling time. The final time is `tfinal`, the final state and input are in `xfinal` The time for the next desired state and input is `tnext`, where `tinit< tnext< tfinal`. The sampling time is `Ts`. The matrices `Art` and `Brt` are the state and input matrix respectively for the propagation of the flat flag.

The function

```
RTSTRUC *TG_rt_create(int zlength, int dimoutput, int Tdel,
        double Ts, int rttype);
```

creates a `RTSTRUC`. We can print some of the members of this structure with

```
int TG_rt_free(RTSTRUC *prtstruc);
```

and release the structure with

```
int TG_rt_print(RTSTRUC *prtstruc, char *text).
```

To compute the matrices `Art` and `Brt` one can use

```
int TG_rt_compAB(BASIS *pbasis, FLATSTRUC *pflatstruc, RTSTRUC *prtstruc);
```

To propagate the flat flag with the matrices `Art, Brt` use

```
int TG_rt_updateflatflag(OPTSTRUC *poptstruc, RTSTRUC *prtstruc);
```

which will compute `z1 = Art z1 + Brt zf`, and write the result to `prtstruc->z1`.

Example

Consider the following streamlined code fragment from the file `cost.c` in the `examples` directory:

```
TG_rt_init();

if(prtstruc0->rttype == 1){
       TG_rt_compAB(pbasis, pflatstruc0, prtstruc0);
};

ptrajxu->current = prtstruc0->Tdel;

while(!stop){

     if(prtstruc0->rttype == 1){
        TG_rt_updateflatflag(poptstruc0, prtstruc0);
```

```
            TG_flat_z2xu(pflatsys0, poptstruc0->z0, prtstruc0->xinit, pars);
        }

        traj_datachange(ptrajxu, prtstruc0->tfinal, prtstruc0->xinit+1,
             ptrajxu->current);
        ptrajxu->current++;

if(TG_rt_readinput(prtstruc0) < DIMOUTPUT+1) stop = 1;

    }

    traj_save(xutrajdatfile, ptrajxu);

    TG_rt_cleanup();
```

We initialize the necessary structures in `void TG_rt_init()`. They are cleaned up by `void TG_rt_cleanup()`. The necessary parameters initialized in `TG_rt_init()` are listed in the included parameters file, which can be recognized by the suffix `xx_rt.par`, where `xx` indicates the physical system. This example reads pilot input from a file in the function `TG_rt_readinput()` and writes to a trajectory `ptrajxu`. First we compute the propagation matrices with `TG_rt_compAB()`, and set the `current` pointer in `ptrajxu` to `prtstruc0->Tdel`, since that is the number of samples we wait. Then we get into a loop, running until there is no more pilot input and `TG_rt_readinput()` returns fewer than `dimoutput` numbers. In the loop we propagate the flat flag with `TG_rt_updateflatflag()`, and compute the state and input from `poptstruc0->z1` with `TG_flat_z2xu()`. We save this state in `ptrajxu`, update its `current` pointer, and read new input. The function `TG_rt_readinput()` is user supplied and updates the data in `prtstruc0->ydes` and `poptstruc0->z1`. When there is no more pilot input, `stop` is set to 1 and we exit the loop and save the trajectory to file. In real real-time applications we would not save to file of course, but feed every nominal state to the controller.

Bibliography

[1] I.H. Abott and A.E. von Doenhoff. *Theory of Wing Sections Including a Summary of Airfoil Data.* Dover Publications, 1959.

[2] A. Banaszuk, J. Hauser, W.M. Sluis, and R.M. Murray. On measures of non-integrability of pfaffian systems. In *IFAC Symposium*, July 1996. To appear.

[3] G. Becker and A. Packard. Robust performance of linear parametrically varying systems using parametrically-dependent linear feedback. *Systems and Control Letters*, 3:205 – 215, 1994.

[4] J.T. Betts. Sparse Jacobian updates in the collocation method for optimal control problems. *Journal of Guidance*, 13(3):409–415, May-June 1990.

[5] J.T. Betts and W.P. Huffman. Application of sparse nonlinear programming to trajectory optimization. *Journal of Guidance*, 15(1):198–206, January-February 1992.

[6] J.T. Betts and W.P. Huffman. Path-constrained trajectory optimization using sparse sequential quadratic programming. *Journal of Guidance, Control, and Dynamics*, 16(1):59–68, January-February 1993.

[7] R.L. Bryant, S.S. Chern, R.B. Gardner, H.L. Goldschmidt, and P.A. Griffiths. *Exterior Differential Systems.* Springer Verlag, 1991.

[8] A.E. Bryson and Y.C. Ho. *Applied Optimal Control.* Ginn and Co., 1969.

[9] F. Bullo and R. M. Murray. Experimental comparison of trajectory trackers for a car with trailers. In *IFAC World Conference*, pages 407–412, 1996.

[10] F.M. Callier and C.A. Desoer. *Linear System Theory.* Springer Verlag, 1991.

[11] E. Cartan. Sur l'équivalence absolue de certains systèmes d'équations différentielles et sur certaines familles de courbes. In *Œuvres Complètes*, volume II, pages 1133–1168. Gauthier-Villars, 1953.

[12] E. Cartan. Sur l'intégration de certains systèmes indéterminés d'équations différentielles. In *Œuvres Complètes*, volume II, pages 1169–1174. Gauthier-Villars, 1953.

[13] Honeywell Technology Center. Multivariable control design guidelines. Technical report, Honeywell Technology Center, October 1995. First Draft.

[14] B. Charlet, J. Lévine, and R. Marino. On dynamic feedback linearization. *Systems and Control Letters*, 13:143 – 151, 1989.

[15] D. Chen. An iterative solution to stable inversion of nonminimum phase systems. In *Proc. American Control Conference*, pages 2960–2964, June 1994.

[16] D. Chen. Output tracking of nonlinear nonmimimun phase systems. In *Proc. IEEE Control and Decision Conference*, pages 2340–2345, December 1994.

[17] H. Choi, P. Sturdza, and R.M. Murray. Design and construction of a small ducted fan engine for nonlinear control experiments. In *Proc. American Control Conference*, pages 2618–2622, June 1994.

[18] J.M. Coron. Global stabilization for controllable systems without drift. *Mathematics of Control, Signals, and Systems*, 5:295–312, 1992.

[19] S. Devasia and B. Paden. Exact output tracking for nonlinear time-varying systems. In *Proc. IEEE Control and Decision Conference*, pages 2346–2355, December 1994.

[20] B. Etkin. *Dynamics of Atmospheric Flight*. John Wiley and Sons, 1985.

[21] A.F. Fillipov. Differential equations with discontinuous right-hand side. *AMS Translations*, 42(2):199–231, 1964.

[22] M. Fliess. Generalized controller canonical forms for linear and nonlinear dynamics. *IEEE Transactions on Automatic Control*, 35:994–1001, 1990.

[23] M. Fliess, J. Lévine, Ph. Martin, and P. Rouchon. On differentially flat nonlinear systems. In *IFAC Symposium on Nonlinear Control Systems Design (NOLCOS)*, pages 408–412, 1992.

[24] M. Fliess, J. Lévine, Ph. Martin, and P. Rouchon. Sur les systèmes non linéaires différentiellement plats. In *C.R. Acad. Sci. Paris, t. 315, Série I*, pages 619–624, 1992.

[25] M. Fliess, J. Lévine, Ph. Martin, and P. Rouchon. Linéarisation par bouclage dynamique et transformations de Lie-Bäcklund. In *C.R. Acad. Sci. Paris, t. 317, Série I*, pages 981–986, 1993.

[26] M. Fliess, J. Lévine, Ph. Martin, and P. Rouchon. Nonlinear control and Lie-Bäcklund transformations: Toward a new differential geometric standpoint. In *Proc. IEEE Control and Decision Conference*, pages 339–344, December 1994.

[27] C. Foias. *$\mathcal{H}^\infty$ control theory*. Springer Verlag, 1991.

[28] R.A. Freeman and P.V. Kokotović. Design of 'softer' robust nonlinear control laws. *Automatica*, 29(6):1425–1437, 1993.

[29] R.A. Freeman and P.V. Kokotović. Inverse optimality in robust stabilization. *SIAM Journal of Control and Optimization*, 1994. to appear.

[30] R.B. Gardner and W.F. Shadwick. Symmetry and the implementation of feedback linearization. *Systems and Control Letters*, 15:25–33, 1991.

[31] R.B. Gardner and W.F. Shadwick. The GS algorithm for exact linearization to Brunovsky normal form. *IEEE Transactions on Automatic Control*, 37(2):224–, February 1992.

[32] N.H. Getz. *Dynamic Inversion of Nonlinear Maps with Applications to Nonlinear Control and Robotics*. Ph.D. thesis, UC Berkeley, Berkeley, California, 1995.

[33] N.H. Getz and K. Hedrick. An internal equilibrium manifold method of tracking for nonlinear nonmimimum phase systems. In *Proc. IEEE Control and Decision Conference*, pages 2241–2245, December 1995.

[34] A. Giaro, A. Kumpera, and C Ruiz. Sur la lecture correcte d'un résultat d'Elie Cartan. *C.R. Acad. Sc*, 287 Série A:241 – 244, 1978.

[35] J.W. Grizzle and M.D. Di Benedetto. Asymptotic model matching for nonlinear systems. *IEEE Transactions on Automatic Control*, 39(8):1539–1550, August 1994.

[36] J.W. Grizzle, M.D. Di Benedetto, and F. Lamnabhi-Lagarrigue. Necessary conditions for asymptotic tracking in nonlinear systems. *IEEE Transactions on Automatic Control*, 39(9):1782–1795, September 1994.

[37] W. Hahn. *Stability of Motion*. Springer Verlag, 1967.

[38] C.R. Hargraves and S.W. Paris. Direct trajectory optimization using nonlinear programming and collocation. *Journal of Guidance*, 10(4):338–342, July-August 1987.

[39] J. Hauser, S. Sastry, and G. Meyer. Nonlinear control design for slightly nonminimum phase systems: Application to V/STOL aircraft. *Automatica*, 28(4):665–679, 1992.

[40] F. Doyle III, Frank Allgöwer, and M. Morari. A normal form approach to approximate input-output linearization for maximum phase nonlinear SISO systems. *IEEE Transactions on Automatic Control*, 41(2):305–309, February 1996.

[41] A. Ilchmann, D.H.Owens, and D. Prätzel-Wolters. Sufficient conditions for stability of linear time-varying systems. *Systems and Control Letters*, 9:157–163, 1994.

[42] A. Isidori. *Nonlinear Control Systems*. Springer Verlag, 1989.

[43] A. Isidori and C.I. Byrnes. Output regulation of nonlinear-systems. *IEEE Transactions on Automatic Control*, 35:131–140, 1990.

[44] M. Kantner, B. Bodenheimer, P. Bendotti, and R.M. Murray. An experimental comparison of controllers for a vectored thrust, ducted fan engine. In *Proc. American Control Conference*, pages 1956–1961, June 1995.

[45] H. Khalil. *Nonlinear Systems*. Macmillan Publishing Company, 1992.

[46] M. Krstić, I. Kanellakopoulos, and P. Kokotović. *Nonlinear and Adaptive Control Design*. John Wiley and Sons, 1995.

[47] A.M. Kuethe and C.Y.Chow. *Foundations of Aerodynamics: Bases of Aerodynamic Design*. John Wiley and Sons, 1976.

[48] H. Kwakernaak and R. Sivan. *Linear Optimal Control Systems*. Wiley and Sons Inc., 1972.

[49] D.A. Lawrence and W.J. Rugh. On a stability theorem for nonlinear systems with slowly varying inputs. *IEEE Transactions on Automatic Control*, 35(7):860–864, July 1990.

[50] E.B. Lee and L.Markus. *Foundations of Optimal Control Theory*. Wiley and Sons Inc., 1967.

[51] L. Ljung. *System Identification, Theory for the User*. Prentice Hall, 1987.

[52] L. Ljung. *System Identification Toolbox*. MathWorks, 1991.

[53] Ph. Martin. *Contribution à l'étude des systèmes différentiellement plats*. Ph.D. thesis, L'Ecole Nationale Supérieure des Mines de Paris, 1993.

[54] Ph. Martin. Endogenous feedbacks and equivalence. In *Mathematical Theory of Networks and Systems*, Regensburg, Germany, August 1993.

[55] Ph. Martin, S. Devasia, and B. Paden. A different look at output tracking: control of a VTOL aircraft. In *Proc. IEEE Control and Decision Conference*, pages 2376–2381, December 1994.

[56] Ph. Martin and P. Rouchon. Feedback linearization and driftless systems. *Mathematics of Control, Signals, and Systems*, 7:235–254, 1994.

[57] G. Meyer, L.R. Hunt, and R. Su. Nonlinear system guidance in the presence of transmission zero dynamics. Technical Report Draft 5, NASA, October 1994.

[58] G. Meyer, L.R. Hunt, and R. Su. Nonlinear system guidance. In *Proc. IEEE Control and Decision Conference*, pages 590–595, December 1995.

[59] M. Morari. Model predictive control: Multivariable control technique of choice in the 1990s? Technical Report CIT/CDS 93-024, California Institute of Technology, 1993.

[60] J. Morris and M. van Nieuwstadt. Experimental platform for real-time control. In *Proceedings of the ASEE Annual Conference*, 1994.

[61] J. Morris, M. van Nieuwstadt, and Pascale Bendotti. Identification and control of a model helicopter in hover. In *Proc. American Control Conference*, pages 1238–1242, 1994.

[62] B. Morton, D. Enns, and B.Y. Zhang. Stability of dynamic inversion control laws applied to nonlinear aircraft pitch-axis models. *International Journal of Control*, 63:1–25, 1996.

[63] R. M. Murray, M. Rathinam, and W.M. Sluis. Differential flatness of mechanical control systems: A catalog of prototype systems. In *ASME International Mechanical Engineering Congress and Exposition*, 1995.

[64] R.M. Murray. Nilpotent bases for a class of non-integrable distributions wih applications to trajectory generation for nonholonomic systems. *Mathematics of Control, Signals, and Systems*, 7:58–75, 1994.

[65] R.M. Murray, Z. Li, and S.S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

[66] M.van Nieuwstadt and J.Morris. Control of rotor speed for a model helicopter: A design cycle. In *Proc. American Control Conference*, pages 688–692, 1995.

[67] M.van Nieuwstadt and R.M. Murray. Approximate trajectory generation for differentially flat systems with zero dynamics. In *Proc. IEEE Control and Decision Conference*, pages 4224–4230, December 1995.

[68] M.van Nieuwstadt and R.M. Murray. Fast mode switching for a thrust vectored aircraft. In *Computational Engineering in Systems Applications*, July 1996. To appear.

[69] M.van Nieuwstadt and R.M. Murray. Real time trajectory generation for differentially flat systems. In *IFAC Symposium*, July 1996.

[70] M.van Nieuwstadt, M. Rathinam, and R.M. Murray. Differential flatness and absolute equivalence. In *Proc. IEEE Control and Decision Conference*, pages 326–333, December 1994.

[71] H. Nijmeijer and A. van der Schaft. *Nonlinear Dynamical Control Systems*. Springer Verlag, 1990.

[72] A. Packard. Gain scheduling via linear fractional transformations. *Systems and Control Letters*, 22:79 – 92, 1994.

[73] W.H. Press, W.T. Vetterling, S.A. Teukolsky, and B.P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 1992.

[74] M. Rathinam. Personal communication, November 1996.

[75] M. Rathinam and R.M. Murray. Configuration flatness of Lagrangian systems underactuated by one control. Technical Report CIT-CDS 96-006, California Institute of Technology, 1996. submitted to CDC 96.

[76] W.J. Rugh. Analytical framework for gain scheduling. In *Proc. American Control Conference*, pages 1688–1694, June 1990.

[77] W.F. Shadwick and W.M. Sluis. On E. Cartan's absolute equivalence of differential systems. In *C.R. Acad. Sci. Paris, t. 313, Série I*, pages 455–459, 1991.

[78] J.J. Slotine and W.L. Li. *Applied Nonlinear Control*. Prentice Hall, 1991.

[79] W.M. Sluis. *Absolute Equivalence and its Applications to Control Theory*. Ph.D. thesis, University of Waterloo, Waterloo, Ontario, 1992.

[80] W.M. Sluis, A. Banaszuk, J. Hauser, and R.M. Murray. A homotopy algorithm for approximating geometric distributions by integrable systems. *Systems and Control Letters*, 1996. To appear.

[81] T. Söderström and P. Stoica. *System Identification*. Prentice Hall, 1989.

[82] J.E. Tierno and R.M. Murray. Robust performance analysis for a class of uncertain nonlinear systems. In *Proc. IEEE Control and Decision Conference*, pages 1684–1689, December 1995.

[83] J.E. Tierno, R.M. Murray, and J.C. Doyle. An efficient algorithm for performance analysis of nonlinear control systems. In *Proc. American Control Conference*, pages 2717–2721, June 1995.

[84] J.E. Tierno, R.M. Murray, J.C. Doyle, and I.M. Gregory. Numerically efficient robustness analysis of trajectory tracking for nonlinear systems. Technical Report CIT-CDS report 95-032, California Institute of Technology, 1995. Submitted to J. Guidance, Controls and Dynamics.

[85] W.F.Shadwick. Absolute equivalence and dynamic feedback linearization. *Systems and Control Letters*, 15:35–39, 1990.

[86] X. Zhu and M. van Nieuwstadt. The Caltech helicopter experiment. Technical Report CIT-CDS 96-009, California Insitute of Technology, June 1996.

Index