An Efficient Environment Encoding for Trajectory Feasibility and Cost Predictions

EE 80 Senior Thesis Report

Sarah Zou
Advisor: Prof. Yisong Yue
Mentor: Benjamin Riviere

June 3, 2022

Contents

1 Abstract

An efficient method to encoding a robot's surrounding is important for the safety and scalability of path planning problems. I present a method for encoding occupied spaces in an environment via parametric geometry (e.g. circles). This encoding is utilized to train a cost and feasibility predicting neural network for path planning between two states. The circle encoding method uses either bounding circles or set cover to provide a lower dimension input of environments for neural network. The lower input dimension of the models allows for a smaller and faster model of similar performance. Feasibility and cost prediction neural networks enable lazy planning for trajectory calculation. While there are no safety-guarantees, the model I present can act as an heuristic to identify the best paths, amortizing a robot's onboard computation. To train a cost and feasibility predicting model, I first generate multiple environments and calculate encoding circles for obstacle-occupied spaces. For each environment, I use planners to calculate multiple trajectories and store their feasibility and cost. I then train the feasibility and cost prediction models from this data . The feasibility model has 90% accuracy on train and test environments. The circle encoding method was compared with occupancy grid encoding method. When encoding methods were compared to predict occupied space, models with circle encoding generalized to unseen environments better than occupancy grid input models. The method of circle encoding I introduce can be utilized in other learning problems that use environment as input. The feasibility and cost predictions done in this work can be applied to bias tree growth or to other global planning methods.

2 Introduction

Efficient path planning around obstacles is important for many applications of autonomous robots such as urban search and rescue, warehouses, and household robots. Currently approaches for navigating around obstacle involve search-based methods such as A* and D* search or optimization-based approaches such as model predictive control (MPC) and sequential convex programming (SCP). Search-based planning methods are often inefficient in obstacle-dense environments due to expensive collision checking operations. Optimization-based methods also tend to become computationally expensive with more obstacles since they avoid obstacles by optimizing trajectory under constraints. Hence when the number of obstacles increase, the number constraints also increases and the problem becomes computationally expensive to solve. Many times, the most optimal path may require many trajectories to be solved. Hence, it would be advantageous to develop a neural network to predict feasibility and cost of a trajectory to narrow down potential candidate trajectories to only the most promising few.

There's been much work in using learning to address the motion planning problem. In [5] and [6], Riviere et. al use neural networks to learn policy and neural tree expansion, respectively, for multi-robot planning from a perfect information global planner. In [2], Li and Miao et. al use an imitation learning-based MPC-MPNet that consists of a path extension generator and discriminator that choose segments of a larger trajectory that is connected with a Model Predictive Controller (MPC).

Prepossessing input for a neural network model can improve performance while decreasing the computation and space needed. For environments, the standard encoding for 2D environments is occupancy grids and 3D environments voxel grids. However these large rasterized environment encodings require training large convolutional networks (CNN) to encode them to lower dimensions. In VectorNet [1], Gao and Sun showed that using vectorized representation of moving agents and

road context information saved them over 70% of model parameters and with an order of magnitude reduction in FLOPs. For encoding of 3D point cloud data, Qi et.al PointNet [3] and PointNet++ [4] have seen success in applying permutation invariant operations for feature learning in classification and segmentation tasks.

In this work, I encode occupied spaces in the environment as an arbitrary number of circles. The arbitrary number of circles are processed through a DeepSet architecture to obtain a fixed size encoding of the environment [8]. Using this method of environment encoding, I train a feasibility and cost predicting model and test its robustness on both seen and unseen environments. I also conduct baseline performance comparisons between circle encoding and occupancy grid encoding. The main contributions of my work are 1) creating a low-dimensional encoding of 2D obstacle space using circles, and 2) presenting a neural network structure that uses a circle encoding to accurately predict feasibility and cost.

3 Problem Formulation

Notation: We denote vectors with boldface lowercase, function with an italics lowercase letter, scalar parameters with plain lowercase, and subspaces/sets with calligraphic upper-case.

Given a robot system with the feasible state space $\mathcal{X}$, let $\mathbf{s} \in \mathcal{X}$ be the start state and $\mathbf{g} \in \mathcal{X}$ be the goal state in the environment. Let $\mathcal{X}_\mathcal{O} \subseteq \mathcal{X}$ be the occupied state space taken up by environment obstacles. Let $\mathcal{U}$ be the feasible control space. Let the system be subject to dynamics in which

$$x_{t+1} = f(x_t, u_t) \text{ subject to } \mathbf{x_t}, \mathbf{x_{t+1}} \in X \text{ and } \mathbf{u_t} \in \mathcal{U} \tag{1}$$

Given a set time step to solve n, let p be the planner function. $p(\mathbf{s}, \mathbf{g}, f, \mathcal{X}/\mathcal{X}_\mathcal{O})$ returns the trajectory if feasible which means:

$$\exists[\mathbf{x_0}, ..., \mathbf{x_n}] \in (\mathcal{X}/\mathcal{X}_\mathcal{O})^{n+1}, [\mathbf{u_0}, ..., \mathbf{u_{n-1}}] \in \mathcal{U}^n \text{ s.t } |\mathbf{x_n} - \mathbf{g}| \leq \epsilon$$
$$\text{where } x_{t+1} = f(x_t, u_t) \text{ and } x_0 = \mathbf{s} \tag{2}$$

where ϵ is a predetermined vector. Else, p will return nonfeasible. We would like to have a neural network learn p, so for a set f and any $(\mathbf{s}, \mathbf{g}, \mathcal{X}/\mathcal{X}_\mathcal{O})$ it can return a feasibility. We then also want a neural network that can approximate the cost function c that is calculated on a feasible trajectory.

4 Algorithm Description and Analysis

To learn p, the continuous set $\mathcal{X}/\mathcal{X}_\mathcal{O}$ needs to be encoded into a vector to provide as input to a neural network. I propose an encoding function g in that $g(\mathcal{X}_\mathcal{O})$ returns a set of circles. Each circle is defined by $c_i = (x_i, y_i, r_i)$ where x_i, y_i are the coordinates of the circle center and r_i is the radius of the circle. Let $\mathcal{C} = g(\mathcal{X}_\mathcal{O})$ be the continuous set covered by the circles output by g. The function g should be chosen to maximize coverage of occupied space $(\mathcal{X}_\mathcal{O} \bigcup \mathcal{C})$ and minimize coverage of feasible space $(\mathcal{X}/\mathcal{X}_\mathcal{O} \bigcap \mathcal{C})$. In this section, I will present two algorithms to find $g(\mathcal{X}_\mathcal{O})$.

4.1 Circle Encoding of Obstacles

4.1.1 Bounding Circles

Suppose the obstacles in the environment are given by k polygons defined by their vertices. Using Welzl's algorithm W defined in [7] on each polygon's vertices, the smallest enclosing circle for a polygon P_i can be found and represented as (x_i, y_i, r_i). Define:

$$g_B(\mathcal{X}_\mathcal{O}) = \mathcal{C} = \{W(P_i) = (x_i, y_i, r_i) \text{ for } i = 1, ..., k\} \tag{3}$$

The enclosing function g_B results in $\mathcal{X}_\mathcal{O} \bigcup \mathcal{C}$ being maximized:

$$\mathcal{X}_\mathcal{O} \bigcup \mathcal{C} = \mathcal{X}_\mathcal{O}$$

An example of this type of circle encoding is shown in Figure 1.

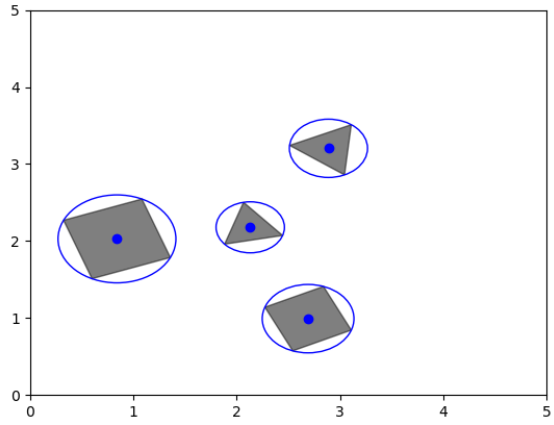
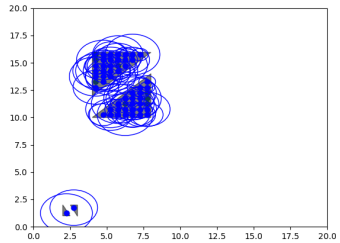


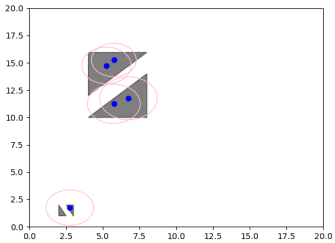
Figure 1: Polygon environment encoded as minimum bounding circles

4.1.2 Set Cover

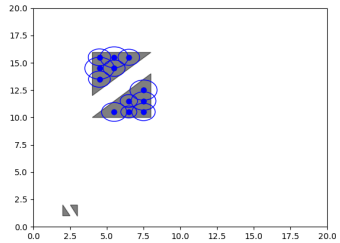
Suppose the vertices are not available but there exists a function h where $h(x, y) = 1$ if $(x, y) \in \mathcal{X}_\mathcal{O}$ and $h(x, y) = 0$ otherwise. In this case, I present an linear set cover algorithm $g_C(\mathcal{X}_\mathcal{O})$ to calculate $\mathcal{C}$ in Algorithm 1. Examples of running Algorithm 1 using different hyperparameters are shown in Figure 2.



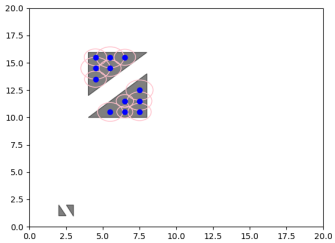
(a) Candidate Circles



(b) Chosen circles from candidate circles in (a)



(c) Candidate Circles



(d) Chosen circles from candidate circles in (b)

Figure 2: Two different runs of described set covering algorithm with different candidate circle productions and grid sizes. Notice the coarser grid in (c) and (d) results in missing the smaller obstacles.

Algorithm 1 Circle Set Covering

1: Create a grid G of points spaced w apart.
2: For $(x, y) \in G$ if $h(x, y) = 1$ append (x, y) to list P.
3: Set m to be the length of P.
4: Create n number of circles with random radius centered at randomly chosen points (without replacement) in P.
5: Iterate through circles and make matrix A of size (m, n) in which $A(i, j) = 1$ if circle j covers point i.
6: Create vector c where $c(i) = \frac{\text{number of points covered by circle i}}{\text{radius of circle i}}$
7: Let $\tilde{A}$ be vertical stack of A and identity matrix of (n, n). b is column vector of m 1's and n 0's.
8: The optimization problem is max $\mathbf{c}^\intercal x$ with constraints $\tilde{A}x \geq b$. Written in primal linear program form:

$$\max \mathbf{c}^\intercal x \text{ s.t. } -\tilde{A}x \leq -b \tag{4}$$

4.2 Neural Network

The data D was partitioned into columns of non-encoding elements(start/end states) and encoding elements (bounding circle or set cover circles of environment obstacles). The non-encoding elements are first processed through a feed forward network (FF1) given in Eq. 5. The encoding elements are processed by permutation-invariant Deep Set block given by Eq. 6. The output of the deep set and FF1 are then fed into a feed forward FF2 that predicts feasibility or cost. The overall

structure is shown in the schematic of Figure 3.

$$FF(x) = W^l \sigma(...W^1 \sigma(x)) \tag{5}$$

$$f(c_1, ..., c_l) = \rho(\sum_{m=1}^{l} \phi(x_m)) \tag{6}$$

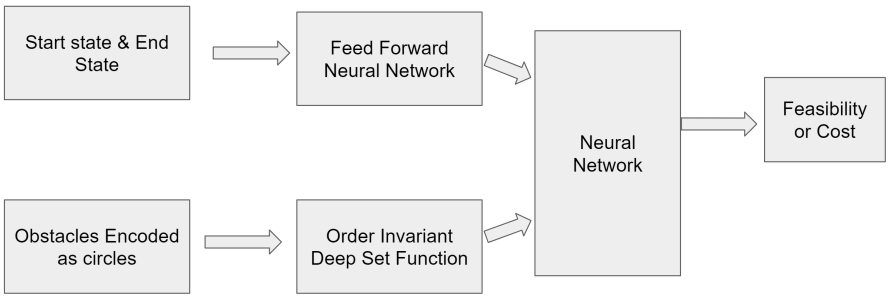


Figure 3: Schematic of Network Architecture

5 Experimental Results

For experiments in this project I used Dubin's car as dynamic system (Equation 7) and sequential quadratic program as solver to plan trajectories that avoided obstacles. However the algorithm and method described is agnostic to choice of system and planner.

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ v \\ \theta \end{bmatrix}, \mathbf{u} = \begin{bmatrix} \dot{v} \\ \dot{\theta} \end{bmatrix} \tag{7}$$

$$\mathbf{x}_{t+1} = f(x_t, u_t) = \mathbf{x}_t + \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \dot{v} \\ \dot{\theta} \end{bmatrix} \Delta t \tag{8}$$

5.1 One Environment Learning

To see if using neural network to predict feasibility is possible, I trained the model to predict feasibility for Dubin Car of state $[x_s, y_s, 0, 0]$ to go to $[x_g, y_g, 0, 0]$. The model was a feed forward model with input layer size of 4 (x_s, y_s, x_g, y_g) with an output of size 1.

The output was between 0 and 1 with 0 being infeasible and 1 being feasible. Trained on 2400 randomly chosen start and end points SQP trajectories and validated on 600 points. Model was trained for 200 epochs with final training accuracy of 99.3% and validation accuracy of 95%. The trained network results are seen in Figure 4. Later the model was also trained over the entire Dubin car state space with training loss of 99.4% and validation accuracy of 86%. These results indicates it is possible for a neural network for learn the feasible trajectories of a system.

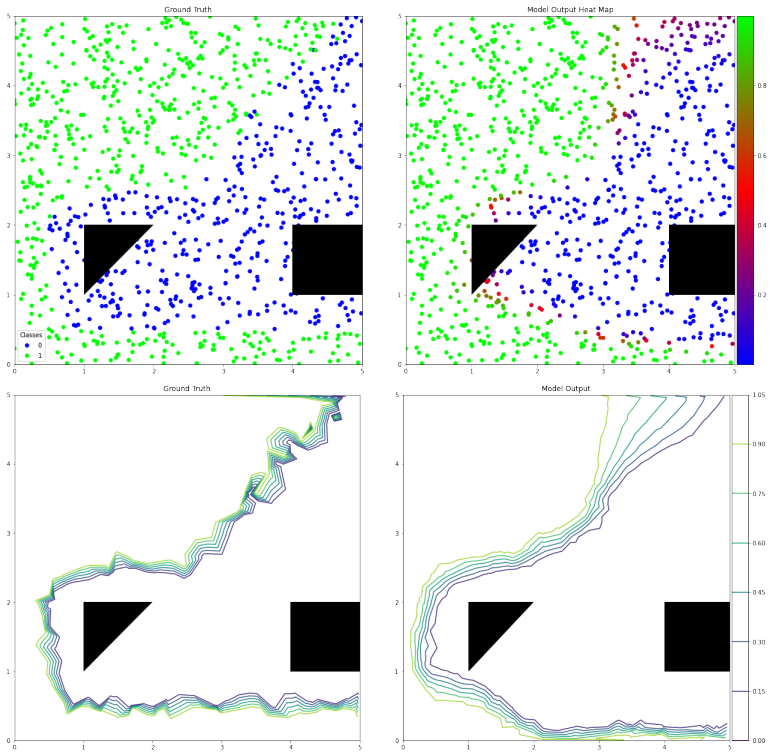


Figure 4: Ground truth vs. One Environment Model of feasible start points (x,y) with end state of [0,0,0,0] for Dubin's car. The green points in the scatter plot indicate the trajectory from that point to the origin is feasible and blue means the described trajectory is infeasible. The bottom two plots show the contour plot of the above scatter plots.

0.750375	0.0515
0.038875	0.15925

Table 1: Confusion matrix of feasibility model

5.2 Multi-Environment Feasibility and Cost Learning

5.2.1 Data Generation

Data was gathered on different random environments with variable number of obstacles and obstacle shapes. For each environment: 1) enclosing circles were calculated according to bounding circles (section 4.1.1), 2) random start and end states were generated, 3) SQP (with set parameters) was run between start and end state and feasibility and cost was recorded. The resulting data columns were [start state, end state, circle encoding of environment, cost, feasibility]. The data was then further preprocessed by subtracting the start position from the end state and circles in the encoding to make a relative coordinate frame.

5.2.2 Network Architecture

Given the success of learning feasiblity in one environment, I then proceeded to train an network for any environment of Dubin Car. The network was trained on 150k points from 30 environments with 5k points from each environment. The environments had up to 10 obstacles. Obstacles were squares and triangles of up to 1 width in a 5x5 environment.

FF1 has input layer of 6 neurons, one hidden layer of 50 neurons, and an output layer of 20 neurons. ϕ has input layer of 3, hidden layers of 10,20, and 50 neurons, and an output layer of 20. ρ has input layer of 20 neurons (the same size as ϕ output layer), hidden layers of 40, 100, 200 nuerons, and an output layer of 20 neurons. The output of FF1 and ρ were combined and feed into FF2 (feed-forward network) of input size 40 with hidden layers of 50, 100, 50, 10, and output of size 1. The components of the model were trained together end-to-end. All networks used fully connected feed forward structure with ReLU activation. Batch size was 1k and trained for 200 epochs. Learning rate used was 7×10^{-3}. Loss functions were chosen depending on learning goal. For feasibility, I used PyTorch BCEWithLogitsLoss as loss function. For cost, the loss function was changed to PyTorch MSE.

5.2.3 Model Performance

The feasibility model was trained on 40k data points which were 5k points in 8 different environments (1-4 obstacles). At the end of training, model was 91.9% accurate on training data and 91% accurate on test data from the same environments.

Table 1 is confusion matrix on test data. There is a 5% false negative rate and a 4% false positive rate. In one unseen environment (4 obstacles) the model yields 89% accuracy. In one unseen environment (1 obstacles) the model yields 90.6% accuracy (Figure 5).

The cost model was trained on 23,431 feasible trajectories in 22 different environments (1-5 obstacles). The results are also quite satisfactory with average MSE loss of 671. The cost had a standard deviation of 1458 and ranged from 9 to 8,738. The mean cost is 1,926. So a MSE loss of 671 is quite good. Figure 6 shows cost prediction versus ground truth. Using circle set cover algorithm (Algorithm 1) to replace bounding circles encoding in data generation section 5.2.1 results in similar performance.

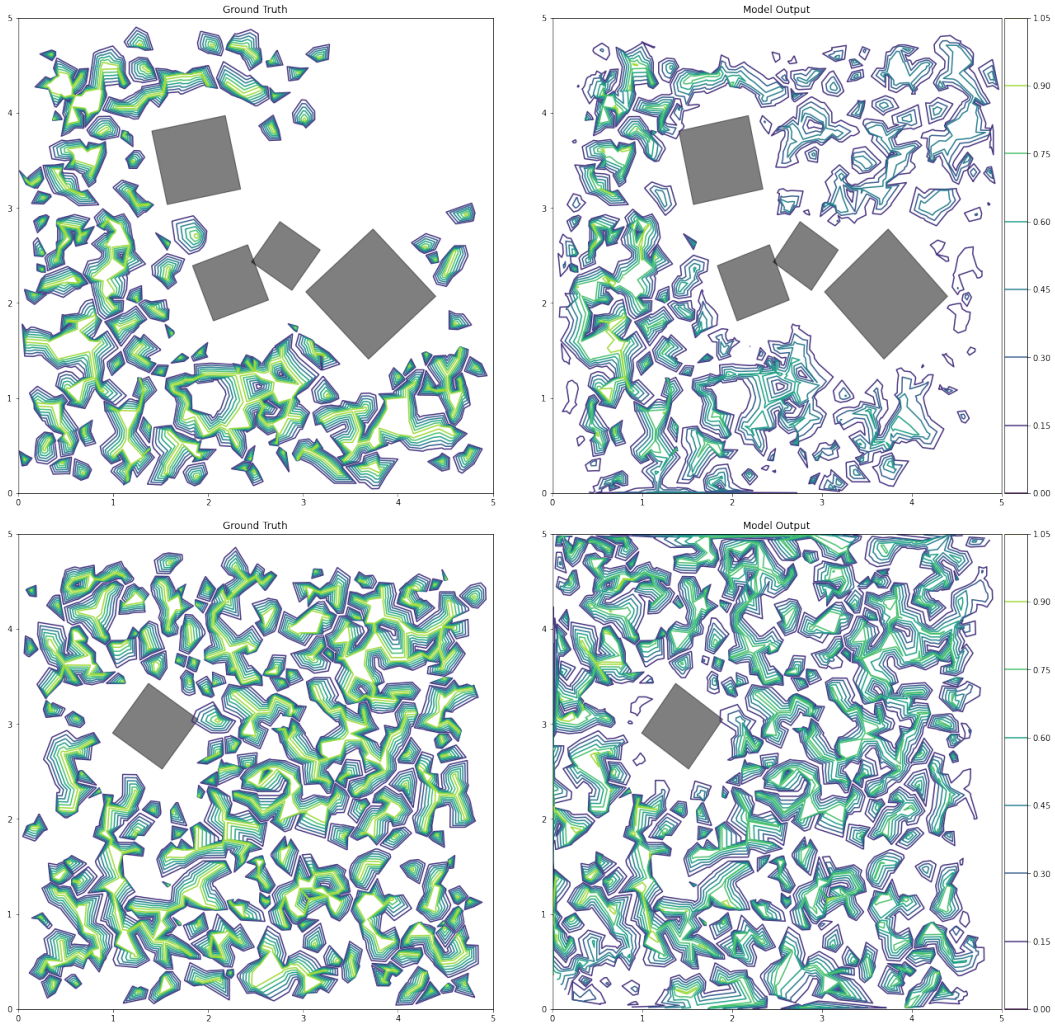


Figure 5: Feasibility Model Prediction contours compared with ground truth in two different environments in which end state is $[0, 0, 0, 0]$ for Dubin's car

5.3 Baselines: Environment Encoding

I then compare the performance of circle encoding to occupancy grid encoding. The first experiment was with 3x3 occupancy grid. In each environment, there was only one square obstacle with set width centered at one of the occupancy grid points. The occupancy grid was then put into a feed-forward neural network to act as baseline model. Since the obstacles were quite big compared to environment space, instead of predicting feasibility, the models were trained to predict whether the input point was in an obstacle or not. The resulting outputs are shown in Figure 8. The loss and accuracy over epochs are shown in Figure 7. Notice that the circle encoding model outperforms the occupancy grid encoding model. The reasoning for baseline testing models to predict whether an end goal from the start in in an obstacle or not is that predicting whether goal is in obstacle is the bare minimum task a feasibility model needs to do: a goal being in an obstacle makes a trajectory infeasible. For these experiments, the starting point was changed to the origin of the $[-1, 1]^2$ environment and the endpoint was randomly generated.

I then scaled the experiment to higher resolution and random fixed obstacle placement. I then compare the encoding performances with a 20x20 occupancy grid. I trained a CNN and a

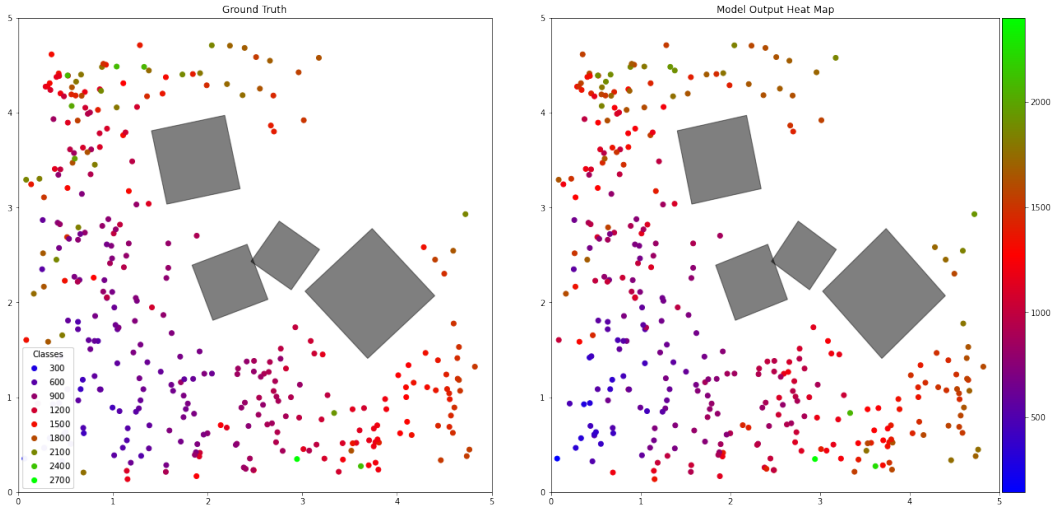


Figure 6: Cost Prediction on Unseen environment with origin as desired end state

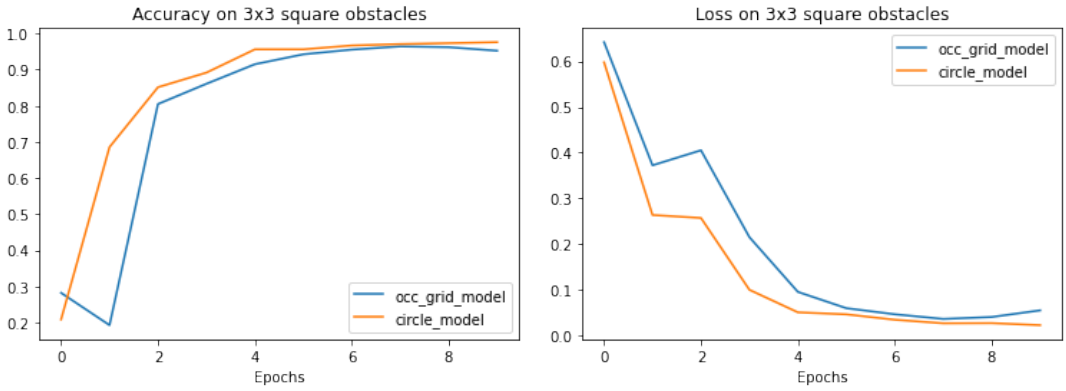


Figure 7: 3x3 Circle Encoding versus Occupancy Grid Model Accuracy and Loss

Multi-layer Preceptron Network (also known as Feed Forward Network) with the end point and (20×20) occupancy grid encoding of the environment. I then also trained the same feasibility network described in the previous section with the bounded circle encoding and the set covering encoding. The models trained over 1k points in 1k different environments with 1 to 10 obstacles for a total training set of 1 million data points. All models were trained until accuracy and losses between epochs stabilized.

The results show on seen environments, the CNN model tend to overfit the obstacles (Figure 9). The overfit hypothesis is supported by the poor performance on unseen environments in Figure 10 in which the CNN predicts feasible over an entire obstacle. The MLP model has the best performance but I predict at higher dimensions or higher resolutions the MLP will start to fail. On unseen environments, the circle encoding models perform better than CNN and on-par with MLP. The performance difference between circle and MLP might arise from the non-optimal covering of obstacles by bounding circle and set cover. The input circles are shown in pink in Figure 9 and Figure 10. Notice the pink circles do not provide a clear boundary between feasible and non-feasible.

6 Conclusions

In this project, I introduce a novel method for environment encoding and use this encoding to successfully train neural networks to predict feasibility and cost. This method is general and can be applied to any combination of dynamics, planner, and environment. Circle encoding can be applied to any learning problem that take environment as input. The feasibility and cost predicting models has the potential to improve any global planning strategies that are burdened with computationally-intensive local planning operations. Although, I cannot show that my encoding method consistently beats baseline occupancy grid models, I have provided a lightweight alternative without compromising accuracy by much. For future work, the project will benefit with better tuning of hyperparameters, refinement of circle set covering algorithm or using ellipses over circles for encoding, and further exploration of situations where circular encoding is superior to other encoding types.

References

[1] Jiyang Gao et al. "Vectornet: Encoding hd maps and agent dynamics from vectorized representation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 11525–11533.

[2] Linjun Li et al. "Mpc-mpnet: Model-predictive motion planning networks for fast, near-optimal planning under kinodynamic constraints". In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 4496–4503.

[3] Charles R Qi et al. "Pointnet: Deep learning on point sets for 3d classification and segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 652–660.

[4] Charles Ruizhongtai Qi et al. "Pointnet++: Deep hierarchical feature learning on point sets in a metric space". In: *Advances in neural information processing systems* 30 (2017).

[5] Benjamin Riviere et al. "Glas: Global-to-local safe autonomy synthesis for multi-robot motion planning with end-to-end learning". In: *IEEE Robotics and Automation Letters* 5.3 (2020), pp. 4249–4256.

[6] Benjamin Riviere et al. "Neural Tree Expansion for Multi-Robot Planning in Non-Cooperative Environments". In: *IEEE Robotics and Automation Letters* 6.4 (2021), pp. 6868–6875.

[7] Emo Welzl. "Smallest enclosing disks (balls and ellipsoids)". In: *New results and new trends in computer science*. Springer, 1991, pp. 359–370.

[8] Manzil Zaheer et al. "Deep sets". In: *Advances in neural information processing systems* 30 (2017).

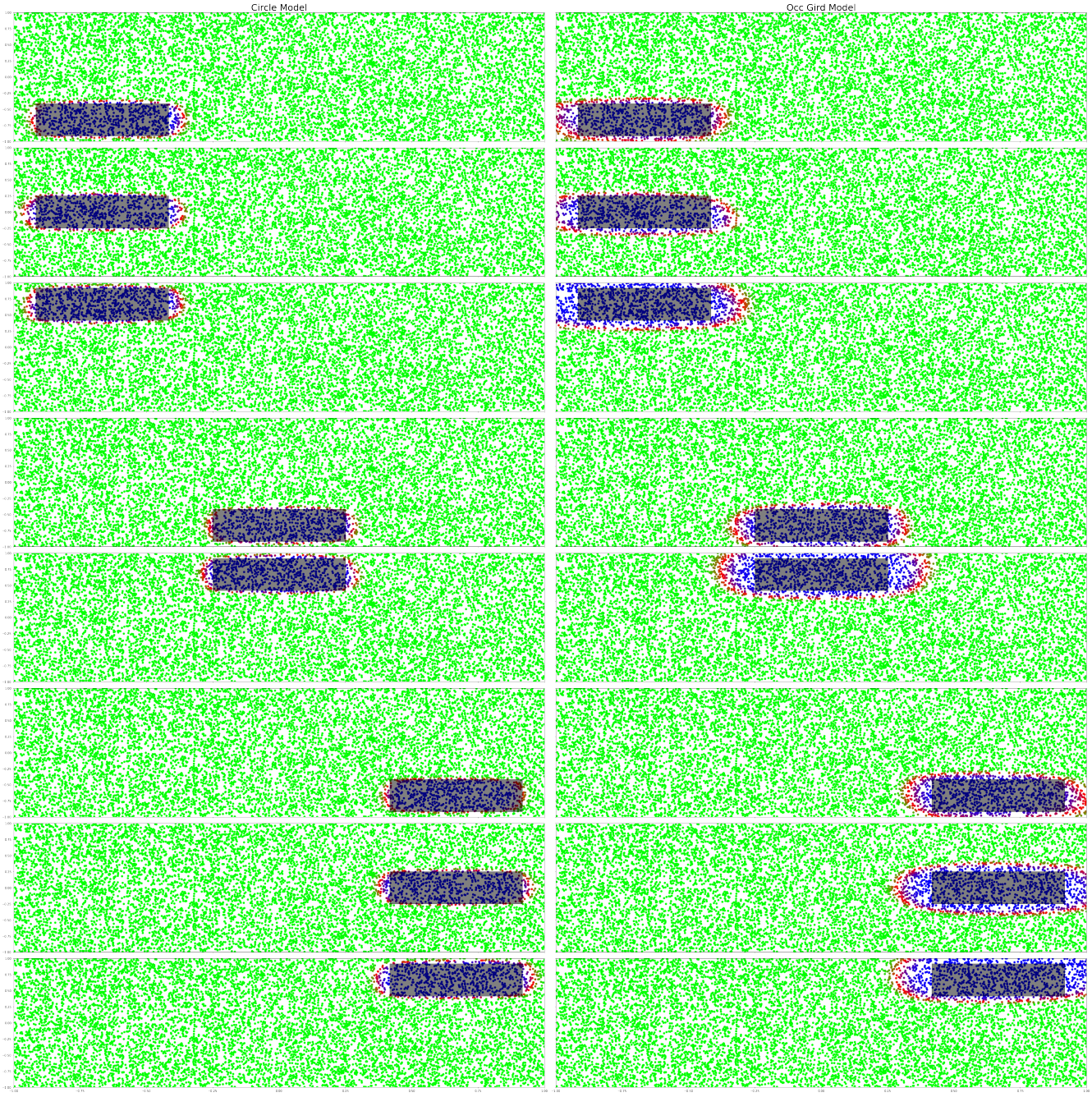


Figure 8: 3x3 Circle Encoding versus Occupancy Grid Models Output Comparisons

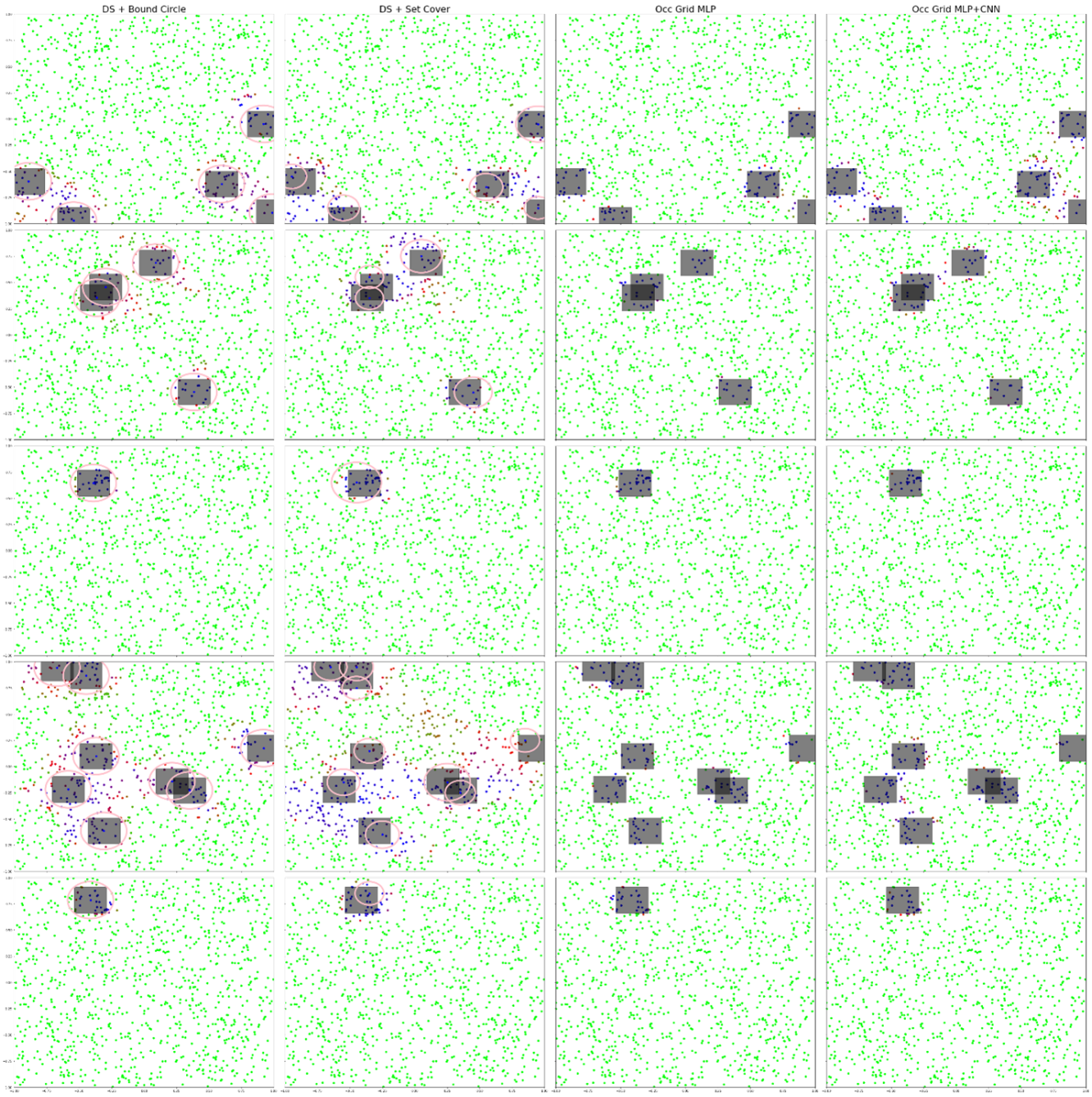


Figure 9: Model Outputs on Training Set with different circle encoding of environments

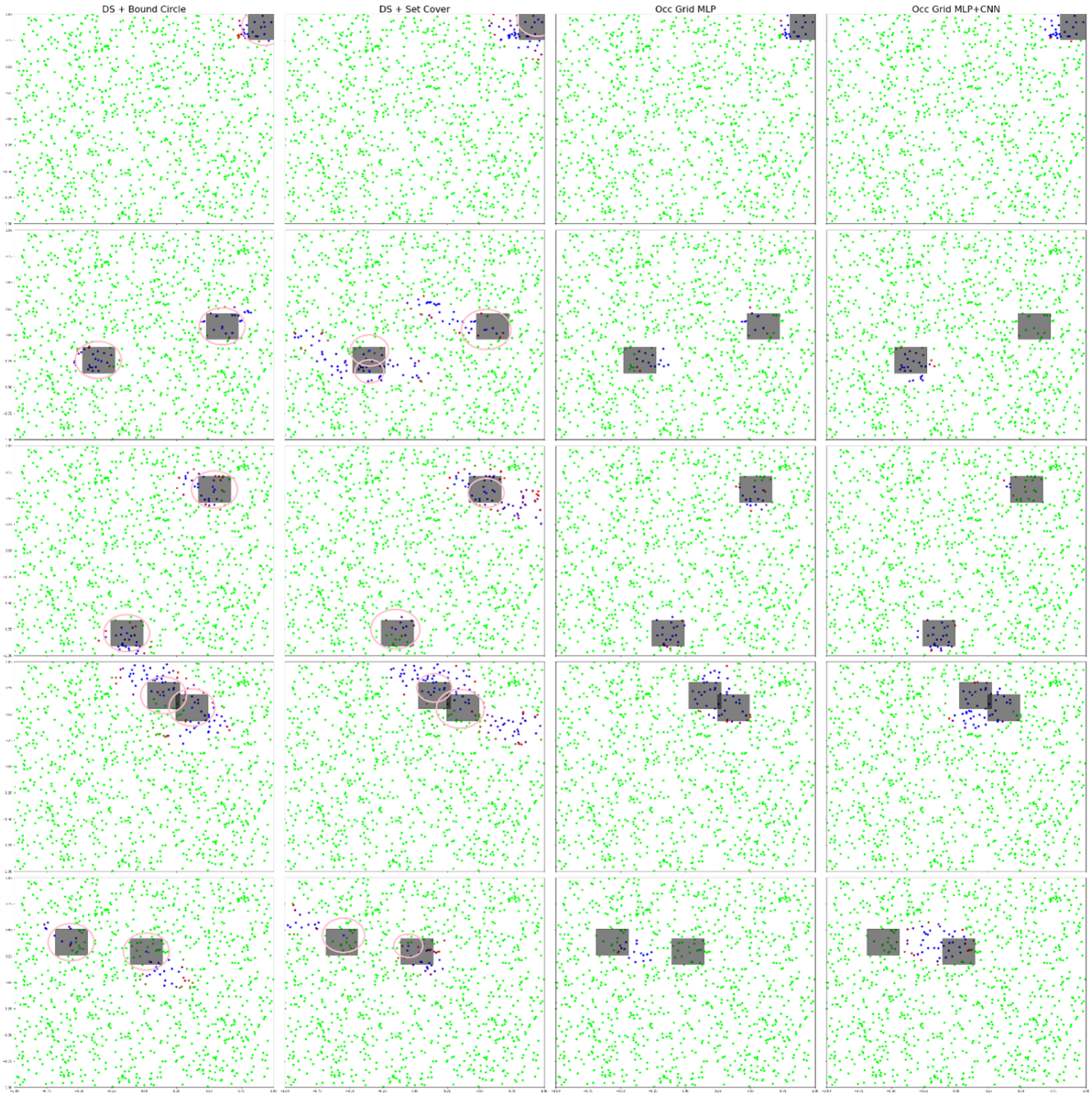


Figure 10: Model Outputs on Unseen Environments