Trajectory Bundle Estimation For Perception-Driven Planning

by

Abraham Galton Bachrach

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2013

Author . . . . . .
Department of Electrical Engineering and Computer Science
December 31, 2012

Certified by . . . . . .
Nicholas Roy
Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by . . . . . .
Leslie Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Trajectory Bundle Estimation For Perception-Driven Planning

by

Abraham Galton Bachrach

Submitted to the Department of Electrical Engineering and Computer Science
on December 31, 2012, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

When operating in unknown environments, autonomous vehicles must perceive and understand the environment ahead in order to make effective navigation decisions. Long range perception can enable a vehicle to choose actions that take it directly toward its goal, avoiding dead ends. In addition, the perception range is critically important for ensuring the safety of vehicles with constrained dynamics. In general, the faster a vehicle moves, the more constrained its dynamics become due to acceleration limits imposed by its actuators. This means that the speed at which an autonomous agent can safely travel is often governed by its ability to perceive and understand the environment ahead. Overall, perception range is one of the most important factors that determines the performance of an autonomous vehicle.

Today, autonomous vehicles tend to rely exclusively on metric representations built using range sensors to plan paths. However, such sensors are limited by their maximum range, field of view, and occluding obstacles in the foreground. Together, these limitations make up what we call the metric sensing horizon of the vehicle. The first two limitations are generally determined by the weight, size, power, and cost budget allocated to sensing. However, range sensors will always be limited by occlusions.

If we wish to develop autonomous vehicles that are able to navigate directly toward a goal at high speeds through unknown environments, then we must move beyond the simple range-sensor based techniques. We must develop algorithms that enable autonomous agents to harness knowledge about the structure of the world to interpret additional sensor information (such as appearance information provided by cameras), and make inferences about parts of the world that cannot be directly observed.

We develop a new representation based around trajectory bundles, that makes this challenging task more tractable. Rather than attempt to explicitly model the geometry of the world in front of the vehicle (which can be incredibly complex), we reason about the world in terms of what the vehicle can and cannot do. Trajectory bundles are designed to capture an abstract concept such as the command "go straight and then turn towards the right" in a concrete and actionable manner. We employ a library of trajectory bundles to reason about the layout of obstacles in the environment based on which bundles in the library are predicted to be feasible. Trajectory bundles provide a lens through which we can look at

perception tasks, allowing us to leverage machine learning tools in much more effective ways for navigation.

In this thesis we introduce trajectory bundles, and develop algorithms that use them to enable perception-driven planning. We develop a trajectory clustering algorithm that enables us to construct a set of trajectory bundles. We then develop a Bayesian filtering framework that enables us to estimate a belief over which trajectory bundles are feasible based on the history of actions and observations of the vehicle. We test our algorithms by using them to navigate a simulated fixed wing air vehicle at high speeds through an unknown environment using a monocular camera sensor.

Thesis Supervisor: Nicholas Roy
Title: Professor of Aeronautics and Astronautics

Acknowledgments

I have learned so much, and had so much fun these last several years. I am extremely lucky to have had such an amazing opportunity. I would like to thank the many people who have helped me get here, and who have made the experience what it was. Without your patient support, guidance, friendship and love I would not be where I am today.

In particular, I would like to thank my advisor, professor Nicholas Roy for his support and guidance. He has been a great sounding board for ideas, and instrumental in molding my loose ramblings into coherent research. He has in all ways exemplified what it means to be a mentor.

My thesis committee consisting of Seth Teller, and Tomas Lozano-Perez who have been invaluable resources, and helped refine both the thesis problem, as well as the technical approach. In addition, I would like to thank professors Russ Tedrake, John Leonard, Bill Freeman, and Antonio Torralba for always being happy to discuss my research, and give me their take on things.

I would also like to thank the many friends and collaborators who I have worked with. Samuel Prentice and Ruijie He, my conspirators in everything quadrotor related. Adam Bry for bringing a host of new challenges and ideas to make our research more dynamic and exciting. John Roberts for always being willing to debate anything and everything. Our many dart games were important relaxation, yet also fostered many important research discussions. Albert Huang for the useful guidance, instruction, and always being willing to show me how to do things "the right way". Thanks to Charles Richter, Will Vega-Brown, and Johnathan Kelly for continuing on with the air vehicle research. I cannot wait to see where things progress.

I also benefited tremendously from all the discussions about anything and everything with everyone else in 33x and elsewhere in CSAIL. In particular, I'd like to thank (in alphabetical order) Alex Bahr, Marec Doniec, Maurice Fallon, Hordur Johannsson, Josh Joseph, Michael Kaess, Sertac Karaman, Ross Knepper, Olivier Koch, Tom Kollar, Stefanie Tellex, Finale Doshi-Velez, Javier Velez, and Matt Walter. Thank you for being so generous with your time.

In addition to the people more directly involved in my academic pursuits, I would like to thank my friends in Boston, California, and elsewhere who helped distract me from my work when needed to keep me sane. In particular Katharine Wolf, for your constant care, affection, support, and also for always helping me see that the glass was half full (and this thesis was half done ;-)

Finally, I am grateful to my sisters Rashmi, Devra, Lela, and Bimla for always encouraging me, and being such great role models for me to try to live up to. Last, but certainly not least, I would like to thank my parents for always supporting me, while at the same time pushing me to excel.

Thank you all so much!

Contents

List of Figures

Chapter 1

Introduction

In order to safely move through the environment, an autonomous vehicle must have awareness of the obstacles ahead of it so that it can plan collision free paths. Many autonomous systems assume that such information will be known ahead of time, however there are many situations where accurate prior maps of the environment will not be available. If we wish to develop truly general autonomous vehicles, we must imbue them with the capability to navigate through unknown environments.

Over the years, there has been a large amount of research on algorithms for combining sensor data into maps that can be used for planning. These algorithms aggregate, and interpret the history of measurements taken by the vehicle's sensors as it moves through the environment. Unfortunately, most of the area covered by such observations will be behind the vehicle, and therefore not particularly useful unless the vehicle needs to backtrack and return through an area.

Indeed, much of the work in robotic perception and mapping focuses on understanding where the vehicle has *been* rather than where it is *going*. If we wish to enable autonomous agents to move faster, and more directly through environments for which they do not have access to prior knowledge of the geometric extent and layout of obstacles, we will need to give them algorithms that enable better awareness and understanding of the environment through which they will travel. Today, this information is generally measured by range sensors (such as lidar, or stereo cameras) that directly inform the vehicle about the distance to obstacles. However, the limitations of such sensors mean that in practice, the vehicle

Figure 1-1: A picture of the robust robotics group's autonomous fixed wing aircraft flying in a parking garage area. During the flight depicted the vehicle had access to a prior map of the environment. This thesis seeks to develop algorithms that could enable the vehicle to not require a map ahead of time.

will not have sufficient lookahead range to make good navigation decisions. To overcome these limitations, new algorithms for *inferring* information about the areas that is not directly measurable by range sensors are needed. These algorithms will allow vehicles to make better decisions, and choose better actions than is currently possible when relying exclusively on information that is directly measurable by its range sensors.

Such issues are especially pertinent for a fast moving fixed-wing air vehicle navigating among obstacles, such as the one developed by our group [14], shown in in figure 1-1. The vehicle has highly constrained dynamics since it cannot slow down below the minimum speed required to maintain lift for flight, and cannot change direction instantaneously. As a result it must be able to perceive and reason about the environment sufficiently far ahead to plan safe paths.

Fixed wing air vehicles are just one example of a vehicle configuration that has highly constrained dynamics. Many wheeled robots, such as car-like vehicles have a minimum turning radius. In general, any vehicle with actuation limits will not be able to stop or change its direction of travel instantaneously. As a result, such vehicles must make naviga-

tion decisions well in advance of when they arrive at constraining obstacles.

As the dynamics of the vehicle become more constrained, it is increasingly important for the vehicle to reason about what it will do further into the future. An intuitive analogy is to think of driving a car. At low speeds, the car can stop in a few feet, or swerve to avoid a collision. However, as the car goes faster, the stopping distance increases dramatically, as does the minimum turning radius. When driving on the highway, it is very important for the driver to look far ahead for oncoming hazards.

Even for an idealized holonomic vehicle with direct control over its velocity, extending the planning lookahead is very important. For such vehicles, lookahead is needed to take direct paths through the environment. Without sufficient lookahead, the vehicle will behave myopically. It will follow walls and enter cul-de-sacs that require backtracking to exit. This results in unnecessarily long routes.

At present, autonomous vehicles tend to rely exclusively on metric range sensors such as laser range-finders or stereo cameras to sense obstacles, and inform navigation. However, range measurements will always be limited in a number of ways. All sensors have a maximum usable range at which they can sense obstacles. In figure 1-2(a), we depict the measurements provided by a 5m lidar sensor. The limited range means that the vehicle will have very little information about the surrounding environment through which it will travel. Even if the vehicle has access to a significantly longer range sensor, such as the 30m lidar sensor depicted in figure 1-2(b), obstacles in the foreground will occlude the view of the sensor, and the vehicle will not have any information beyond the occlusion points. We use lidar sensors for reference since stereo and other range sensors are similar but typically with greater range uncertainty. Taken together, these limitations form what we refer to as the metric sensing horizon. When creating a map from the range measurements, the measurements usually partition in the world into three classes. Obstacles are marked where the range measurements occur. Free space is marked between the location of the vehicle and the measured obstacle. Finally, areas beyond the metric sensing horizon are labeled as unknown. This leaves the question of how a planner should treat the unknown regions that lie beyond the metric sensor horizon, as depicted by the gray regions in the figure 1-2.

Often, roboticists treat unknown region as obstacles, and enforce a motion constraint

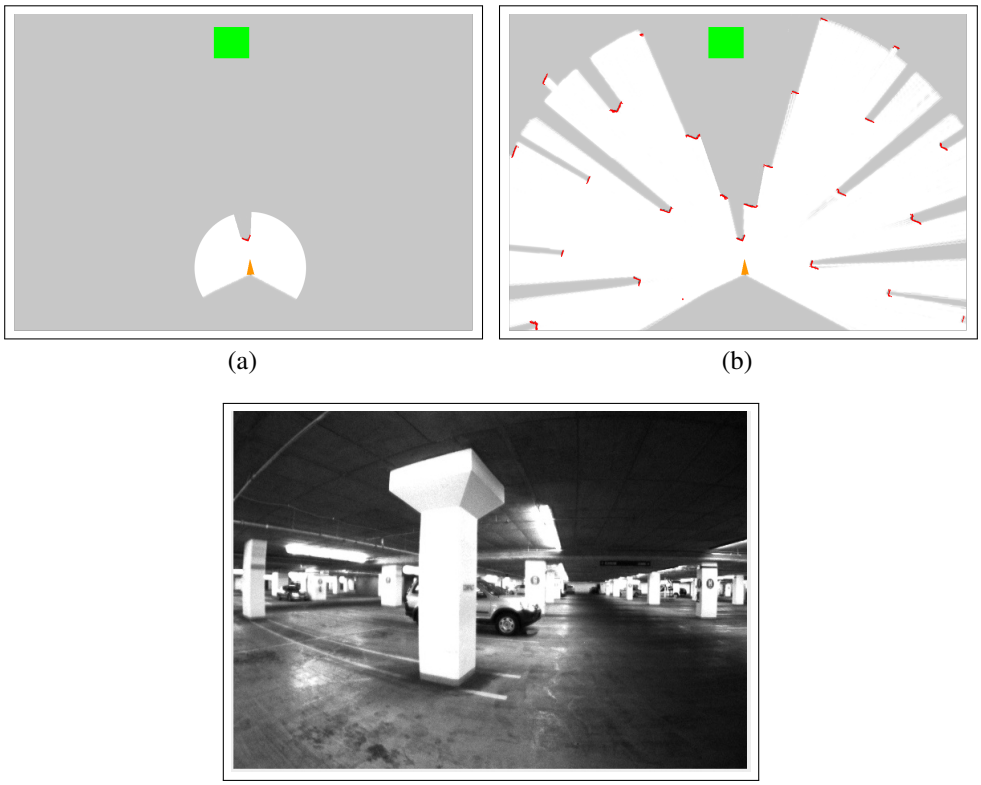


Figure 1-2: An illustration of the limitations of range sensors. Figure (a) shows the limited information provided by a 5m range sensor. The vehicle pose is depicted by the orange triangle, and the goal is the green square. White areas are regions known to be free space, grey are unknown regions, and red are known obstacles. Figure (b) shows the information provided by a 30m range sensor. Even with the significantly greater range, there is still a significant amount of ambiguity caused by the occlusions in the foreground. (c) A camera image taken from the vantage point of the robot. While the image does not directly provide metric range information about the distances to obstacles, it provides a large amount of contextual information that can be leveraged to inform planning and decision making.

that the vehicle never plan a path that enters it. As a result, the metric sensing horizon effectively limits the range at which we can make navigation decisions, limiting the kinds of motion the vehicle can safely execute, and causing myopic behavior. Such motion limitations are often acceptable for slow moving holonomic vehicles, however, if want to develop more agile and fast moving vehicles, we will need to use information beyond simple range measurements to reason about the unknown areas of the environment.

We believe that the best way to extend the planning horizon of autonomous vehicles is to leverage the full range of available information about the environment. Texture and

lighting information from passive monocular camera imagery can provide cues about what the vehicle can do from a very long range, but do not directly provide information about the geometry. The shape of the front side of an obstacle provides clues as to what the backside might look like since many objects have symmetries. However, since such sources of information do not directly provide information about the layout of environmental structures, we must infer the structure from cues contained in the data. Fortunately, we can rely on past experience to distinguish between the likely and unlikely explanations. For example, as a person, we can rely on our knowledge of pillars, cars, and the general layout of similar parking garage structures to interpret the image in figure 1-2(c).

Autonomous vehicles can leverage past experiences from similar environments to interpret its sensors, and make inferences about the areas beyond the metric sensing horizon. This inference in effect extends the perception horizon significantly beyond the metric sensing horizon, thereby improving the navigation performance of the vehicle.

1.1 Perception-Driven Planning

Let the state of the vehicle at time t be denoted s_t. Motion planning algorithms seek to find a sequence of actions that will take the vehicle along a collision free path

$$\tau = \{s_{\text{init}}, \ldots, s_i, \ldots, s_{\text{goal}}\} \tag{1.1}$$

between an initial state s_{init}, and a goal state s_{goal}. The trajectory must obey the dynamics of the vehicle given as

$$s_{t+1} = f(s_t, a) \tag{1.2}$$

where $a \in A$ is a specified action. The action space A may be discrete or continuous, the exact form is not important for our purposes.

Motion planning algorithms often assume that the geometry of the environment (map) is known ahead of time. The map provides a function that indicates whether each position $p \in \mathbb{R}^d$ is in collision.

$$M(p) \rightarrow \{0, 1\} \tag{1.3}$$

The dimensionality d is either 2 or 3 depending on whether the vehicle is constrained to operate on a plane (usually ground vehicles, but sometimes a useful simplification for air vehicles). Often the function M takes the form of an occupancy grid, or a set of polygonal obstacles.

A collision checking function $C(s, M)$ uses the map to determine whether the vehicle extends into occupied space from state s.

$$C(s, M) \rightarrow \{0, 1\} \tag{1.4}$$

It returns the value 1 if the vehicle is in collision. The collision checking function provides access to the configuration space of the robot [79]. Sample based motion planners generally leave the configuration space implicit, however, for some problems it is preferable to precompute a map of the configuration space.

Even with complete information about the layout of obstacles in the environment, finding a collision free path can be a very challenging problem. The planner must search over the set of all possible paths to find one that is feasible. Some algorithms seek to go further and identify the minimum cost path from the set of all feasible paths. Here cost can be any metric on the trajectory, but is often distance, or time.

If the action space of the vehicle is continuous then there are an infinite number of possible trajectories that the vehicle could take. Even if A is discrete, the number of possible trajectories is exponential in the number of time steps. Planning algorithms must therefore be smart about how they search over possible paths and performs collision checking to gain computational tractability.

Often, the search is broken up into many smaller pieces that are stitched together to form a complete trajectory. If the state and action spaces of the vehicle can be discretized, then dynamic programming can be used to efficiently search over the space of paths. For continuous planning problems, sample based motion planning algorithms such as the RRT [74] incrementally grow a tree of trajectories that sparsely explore the collision free space. At any given time, the planner solves for the path between two points ignoring obstacles, and then collision checks the resulting path segment.

When the environment is not known ahead of time, the vehicle must use its sensors to develop an internal representation that enables it to perform collision checking. At each time step, the vehicle will receive new information about the environment, and should therefore compute a new plan that accounts for its improved understanding of the world. Such approaches are generally called receding horizon planning [103]. Receding horizon planning approaches are also employed in situations where the dynamics of the vehicle are complicated enough that computing a complete path to the goal with an accurate dynamics model takes too much time.

In these situations, rather than compute a path all the way to the goal, it is useful to focus more computation effort locally, and then use a heuristic (or approximate planner) to connect the local plan with the goal. The local planner creates a plan out to the planning horizon, and then the heuristic is used to estimate the "cost-to-go" from the end of the plan to the goal. The distance to the planning horizon dictates how far ahead the vehicle will be able to detect and react to obstacles or other decision points.

To accelerate local planning, rather than searching over all possible local trajectories, it can be useful to simply test a representative set of trajectories. The computational effort of generating trajectories can be performed offline by precomputing a library of trajectories L_0 that take the vehicle from the origin out to the planning horizon. At each time step, the trajectory library L_t can be quickly generated by simply applying a rigid body transform to L_0 such that it originates from the current state s_t. The motion planning problem is then reduced to simply picking which of the trajectories in L_t to follow. Normally, the planner checks each trajectory for collisions, and then uses a cost-to-go metric $H(\tau, s_{\text{goal}})$ to choose between the feasible trajectories.

$$\tau_t = \underset{\tau \in L_t}{\operatorname{argmin}} F(\tau, M) H(\tau, s_{\text{goal}}) \tag{1.5}$$

Here the $F(\tau, M)$ is a function that determines whether trajectory τ is feasible in the current environment.

$$F(\tau, M) = \begin{cases} 1, & \text{if } C(s, M) = 0 \quad \forall s \in \tau \\ 0, & \text{otherwise} \end{cases} \tag{1.6}$$

The vehicle executes one or more steps along τ_t before replanning to select τ_{t+1}.

When the environment is unknown, the approach above breaks down since M is not available to use as an input to $F(\tau, M)$. In these situations, a very common approach is to aggregate the sensor measurements Z into a map of the environment.

$$M_t = m(Z_1, \ldots, Z_t) \tag{1.7}$$

Here $m()$ is an arbitrary function that builds a map from sensor data. This allows the same collision checking function to be used, and planning can proceed as before.

$$\tau_t = \underset{\tau \in L_t}{\operatorname{argmin}} F(\tau, M_t) H(\tau, s_{\text{goal}}) \tag{1.8}$$

The map can be thought of as a sufficient statistic of the vehicle's knowledge of the environment. The function $m()$ and the form of M have a significant influence on the accuracy of collision checking.

In the face of noisy and ambiguous sensor data, the intermediate map representation may actually be a hindrance to effective planning rather than a useful organizational paradigm. To incorporate information from such data, it may be more effective to learn models that predict whether each trajectory is feasible directly from the sensor measurements.

$$F(\tau, Z_1, \ldots, Z_t,) \rightarrow \{0, 1\} \tag{1.9}$$

This function can then be used to choose the trajectory to follow.

$$\tau_t = \underset{\tau \in L_t}{\operatorname{argmin}} F(\tau, Z_1, \ldots, Z_t,) H(\tau, s_{\text{goal}}) \tag{1.10}$$

By directly predicting the feasibility of trajectories from data, we frame the collision checking task in terms of a set of binary classification problems. This is beneficial because it makes it easy for the prediction model to leverage large amounts of historical training data to help it interpret the currently observed raw sensor data. As such, the approach makes it easier to use relevant information in the data that does not directly provide information

(a) (b)

Figure 1-3: Two corridors with very different geometry, but similar feasible trajectories. For planning purposes, there is no reason to expend effort modeling all of the intricacies of the tree trunks and branches.
Photos courtesy of Flickr users *raulc* and *residae* via Creative Commons, respectively.

about the environmental geometry.

Another benefit of directly predicting the feasibility of trajectories is that it transforms the problem from reasoning about the geometry of the world to reasoning about what the vehicle can and cannot do. Reasoning about the world in terms of trajectory feasibility, summarizes the relevant information about the environment at the appropriate level of fidelity for planning. There are many environmental structures that have very different geometries, yet result in the same set of feasible trajectories. If we only reason about the trajectory feasibility, we abstract away all of the complexity of the environment geometry that is not relevant to determining whether a trajectory is feasible.

For example, consider the two "corridors" in figure 1-3, one formed by a flat wall on either side, and the other from rows of trees. In both situations a vehicle such as the one shown in figure 1-1 will only be able to fly straight ahead. However, while the configuration space representation of the two scenes may be the same,the true underlying geometry is wildly different. Modeling the flat walls as planes can be done easily, but the geometry of the trees is more complex and not trivially modeled in the same way. While it may be obvious to a person looking at the figures that there is no reason to model all the details of the branches, determining the proper threshold on fidelity a priori is quite challenging. By reasoning about the environment in terms of feasible trajectories, we gain invariance

to details of the geometry that do not influence the actions of the vehicle. We must only reason about factors that are significant enough to influence navigation decisions. In this way, we gain access to the configuration space of the vehicle without first requiring us to model the geometry of the obstacles.

The abstraction provided by reasoning in terms of trajectories is particularly relevant for air vehicles that should never make contact with the environment. For a ground vehicle, modeling each surface in the environment is often very important, such as whether the surface is covered with grass or tall brush. However, for an air vehicle, the concept of interest is purely the 3D geometry of the environment and the details of the surface property are generally irrelevant.

While directly predicting trajectory feasibility provides a number of nice properties, it is still an extremely challenging classification problem. The function relating the input sensor data and the output label is extremely complicated. In addition, the concept that we are asking the classifier to learn is sensitive to small changes in the environment. If a trajectory passes close to an obstacle, then a small shift that is imperceptible in the input features may be the difference between a sample being labeled as feasible or not. As such, the function relating the input to output can be unstable in many regions, which makes learning particularly challenging. To mitigate these issues, we reframe the prediction problem in terms of predicting trajectory bundles, which provides a more abstract concept to learn.

1.2 Trajectory Bundles

Reasoning about the world in terms of the feasibility of all possible trajectories provides some invariance to the structure of the environment, but such exhaustive reasoning remains sensitive to the exact locations of the constraining obstacles. Such fine details can normally be handled by replanning once the vehicle gets closer to the obstacle. From longer ranges, the vehicle only needs coarse guidance for the general direction that it should follow.

In this thesis, we develop a new representation based around a library of trajectory bundles. A trajectory bundle provides a compact, high level concept that we can use to

abstractly reason about what the vehicle can and cannot do in the environment surrounding it. This abstraction frames the learning problem in a manner that is less sensitive to minor changes in the perceptual input and environment layout, thereby making the learning problem easier, resulting in more effective predictions. For decision making purposes, the predictions about the feasibility of the trajectory bundles are mapped back onto the underlying library of trajectories.

A trajectory bundle β is composed of a set of similar trajectories[1] taken from the precomputed library of trajectories L_0.

$$\beta = \{\tau^1, \tau^2, \ldots, \tau^m\}, \qquad \beta \subset L_0 \tag{1.11}$$

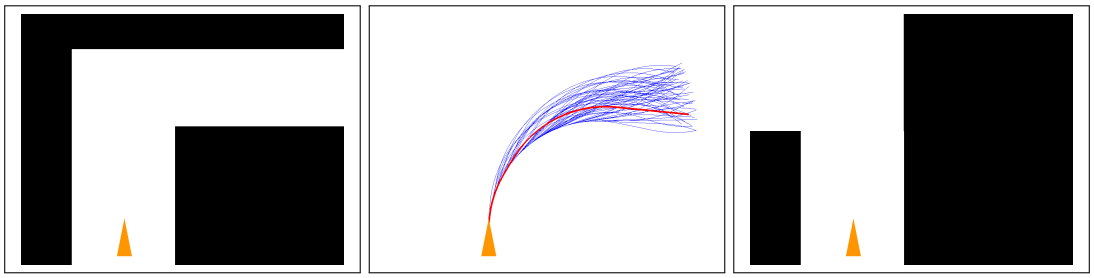
A trajectory bundle can be thought of as consisting of the set of trajectories corresponding to a high level description of a sequence of actions such as: "go straight for a while, and then turn towards the right". A key insight of our approach is that it often much easier to reason about high level abstract concepts, especially in the face of noisy and ambiguous data.

A trajectory bundle is considered to be feasible if *any* of the component trajectories are feasible in the current environment.

$$F(\beta) = \begin{cases} 1, & \text{if } \exists \tau \in \beta \colon F(\tau) = 1 \\ 0, & \text{otherwise} \end{cases} \tag{1.12}$$

For clarity, we drop the input data that collision checking depends on, and simply refer to collision checking a trajectory as $F(\tau)$. By defining feasibility in this manner, the trajectory bundle captures a homotopy-like notion similar in spirit to the one developed in Knepper et al. [67]. Homotopy [8] is a mathematical concept from algebraic topology defining the set of paths between two points that can be continuously deformed between one another without colliding with an obstacle. While the mathematical notion of homotopy is used for inspiration, without a single start and goal for all paths, it is not directly applicable.

[1] Trajectory shape similarity is an inherently ambiguous notion. We will discuss similarity metrics for trajectories in chapter 4.



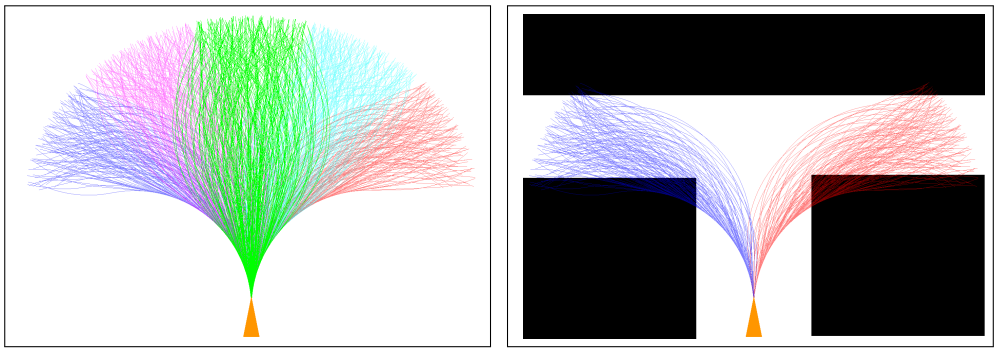
(a) (b) (c)

Figure 1-4: Illustration of the difficulty of determining the feasibility of a trajectory as compared to trajectory bundle. It is quite difficult to determine whether the red trajectory in (b) is feasible in the environment depicted in (a). However, one can be quite confident that a trajectory *similar* to the red one is feasible. In this case, the red trajectory does clip the inside corner, however a number of the blue trajectories are collision free. In contrast, it is obvious that none of the trajectories in (b) are feasible in the environment depicted in (c).

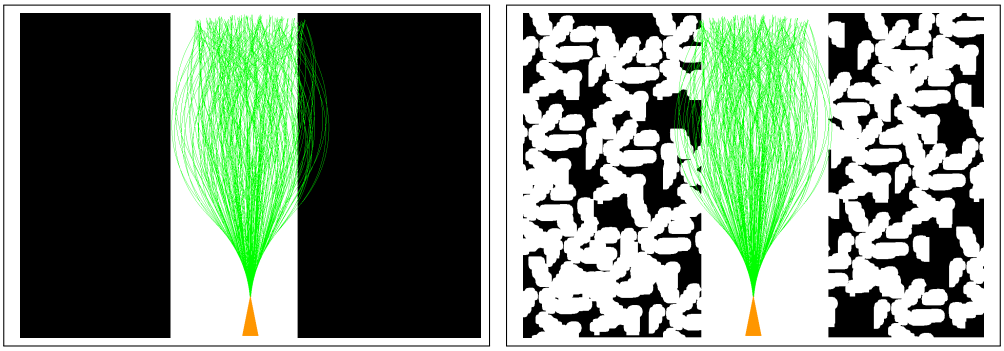
The invariance to minor changes in the environment layout provided by trajectory bundles translates into a concept that is easier to reason about since predictions need not be as specific. In the face of noisy or ambiguous sensor data, it can be easier to make an approximate decision about feasibility, rather than need to determine the exact metric geometry of the environment.

For example, if we ask whether the red trajectory in figure 1-4(b) is feasible in the environment depicted in figure 1-4(a), one would be hard pressed to answer without explicitly measuring or overlaying the trajectory on top of the map. The red trajectory is not in fact feasible in figure 1-4(b) since it brushes against the corner in the turn. While the red trajectory may not be feasible, it is clear that a trajectory similar to the red one should be able to make the right turn without clipping the corner. Therefore, one should have a much easier time answering whether *any* of the blue trajectories in figure 1-4(b) are feasible. In contrast, it should be clear that none of the trajectories are feasible in the environment depicted in figure 1-4(c). The exact details of the specific trajectory are less important than the overall shape of the route. A trajectory bundle is designed to capture this notion.

We employ a library of trajectory bundles to represent the set of options available to the vehicle. The state of the environment is captured by a bitvector indicating which of the



(a) $X = [11111]$ (b) $X = [10001]$



(c) $X = [00100]$ (d) $X = [00100]$

Figure 1-5: An example trajectory bundle library with five bundles. Each bundle consists of trajectories of the same color. The images show the mapping between a potential environment and which trajectory bundles are feasible. In environment (a), all five are feasible, whereas in (b), two are feasible. In (c) and (d), only one bundle is feasible.

bundles in the library are feasible.

$$X = [F(\beta^1), F(\beta^2), \ldots, F(\beta^n)] \tag{1.13}$$

For example, consider the simplistic trajectory bundle library depicted in figure 1-5(a). The library has five bundles roughly corresponding to the actions left, right, straight, left-straight, and right-straight. If we are facing down a hallway, then only the bundle pointing down the hallway is feasible, and the rest would be infeasible, as shown in figures 1-5(c) and 1-5(d). The corresponding trajectory bundle library state is $X = [00100]$. The open space environment depicted in figure 1-5(a) corresponds to the bitvector $X = [11111]$.

The exact composition of the trajectory bundle library, both in terms of the number and composition of the trajectory bundles, as well as the underlying trajectories is up to the

designer. In chapter 4 we will describe the process that we used to develop the trajectory bundle library used in this thesis.

1.3 Estimation of Trajectory Bundle Feasibility

To use trajectory bundles for planning, we must develop algorithms for estimating which bundles are feasible from noisy sensor data. To accomplish this, we frame the problem as a state estimation problem where we estimate the state of the environment in terms of trajectory bundles. Due to the uncertainty in the sensor data, we cannot know which bundles are feasible with certainty. As a result, we will instead estimate a *belief* over which trajectory bundles are feasible in the current local environment. Bayes filtering has been shown to be an effective approach for such estimation problems [113].

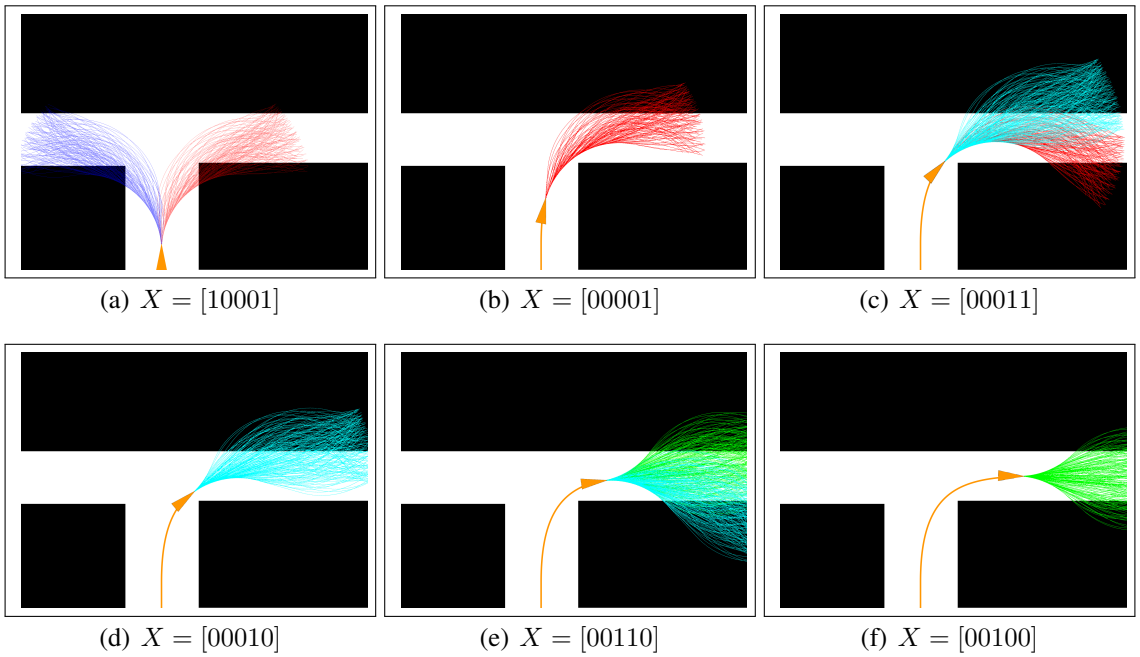
While we assume a static environment, if we estimate the environmental layout from the perspective of the vehicle, the motion of the vehicle translates into an apparent motion of the environment relative to the vehicles coordinate system. This causes the set of feasible trajectory bundles to vary in time in accordance with the surrounding environment. Figure 1-6 shows the progression of which trajectory bundles are feasible as the vehicle rounds the corner in a "T" shaped environment.

Let X_t be a bitvector of length n representing the state of the environment in terms of the feasible trajectory bundles at time t. At each time step, the vehicle will take an action a_t that takes it from its previous position to its current position. We can reason about the effect that this action has on the feasible trajectory bundles by modeling the distribution over the feasibility at time t as being drawn stochastically from a probability distribution $p(X_t|X_{t-1}, a_t)$. After each motion, the vehicle receives a new observation Z_t from its sensors. We can model the observation as being drawn from the distribution $p(Z_t|X_t)$.

We wish to maintain a belief over the state X_t

$$bel(X_t) = p(X_t|Z_{1:t}, a_{1:t}) \tag{1.14}$$

The Bayes filtering algorithm computes this quantity recursively, basing the current esti-



(a) $X = [10001]$ (b) $X = [00001]$ (c) $X = [00011]$

(d) $X = [00010]$ (e) $X = [00110]$ (f) $X = [00100]$

Figure 1-6: An animation of the sequence of feasible trajectory bundles as the vehicle moves through the environment. The vehicle position is indicated by the orange triangle, with the path followed trailing out the back.

mate of the belief on the previous belief $bel(X_{t-1})$. The algorithm splits the computation into two steps: the control update step, which computes

$$\overline{bel}(X_t) = \sum_{X_{t-1}} p(X_t|X_{t-1}, a_t) bel(X_{t-1}) \tag{1.15}$$

and the measurement update step

$$bel(X_t) = \eta p(Z_t|X_t)\overline{bel}(X_t) \tag{1.16}$$

where η is a normalizing constant.

The Bayes filter algorithm as presented here is an extremely general algorithm that can be applied to almost any discrete estimation task. In practice, one must build models for the probability distributions of interest, namely the observation model $p(Z_t|X_t)$, and the transition model $p(X_t|X_{t-1}, a_t)$. These models must be structured such that they are informative, while also ensuring that it is feasible to compute the summations and multiplications re-

quired by the Bayes Filter algorithm.

For our problem, the transition probability distribution $p(X_t|X_{t-1}, a_t)$ subsumes a number of factors, including the geometrical relations between the different trajectory bundles, as well as the distribution over possible environmental layouts. Determining the parameters of this probability distribution either analytically, or manually would be very difficult, so in chapter 6 we present methods for learning approximations to this distribution from data.

Similarly, for very complicated sensors, such as the cameras, or laser rangefinders, it may not be possible to analytically determine a reasonable generative model for the observations of the form $p(Z_t|X_t)$. Instead, in chapter 5 we will seek to develop discriminative models that directly predict the value of X_t from features extracted from the raw sensor measurements. These predictions are then incorporated into the Bayesian filtering framework as pseudo-observations.

1.4 Thesis Goal

In this thesis we focus on developing algorithms that enable an autonomous vehicle to navigate through an unknown environment. We are particularly interested in this problem as it relates to navigation for vehicles with highly constrained dynamics, such as high speed micro air vehicles.

A key technology that is required to enable this capability is the ability to model the layout of the environment far enough ahead of the vehicle in order to avoid obstacles, and make navigation decisions. Rather than building a more conventional occupancy map, or other geometric representation of the environment, we leverage the trajectory bundle representation.

We will determine whether we can create tractable models that accurately estimate a belief over the feasibility of trajectory bundles, and use this belief to make effective navigation decisions.

1.5 Hypothesis

This thesis seeks to address the question of whether the abstraction of the metric environmental geometry layout provided by reasoning about the world in terms of trajectory bundles reduces the data and computational complexity of learning to predict the set of feasible actions available to an autonomous vehicle.

1.6 Contributions

The key contributions of this thesis are:

- Development of the trajectory bundle library representation.
- A trajectory clustering algorithm that can divide a set of trajectories into subsets containing similar trajectories. The trajectory clustering algorithm is used to partition a dense trajectory library into trajectory bundles.
- A data driven prediction algorithm for predicting which trajectory bundles are feasible from monocular camera images.
- A conditional random field model for reasoning about higher level spatial structure in the predicted trajectory bundles.
- A Bayesian filtering algorithm to estimate a belief over the feasibility of trajectory bundles based on the history of action-observation pairs.

1.7 Organization

The chapters of this thesis describe the algorithmic components of our system. Chapter 2 provides a review of previous results in perception-driven mobile robot navigation. The system and world model that we use to develop and evaluate our algorithms is presented in chapter 3. We also describe a baseline trajectory library based planning system that is

used to evaluate the importance of sensing horizon for our problem domain. We then develop an algorithm for clustering vehicle trajectories in chapter 4. The trajectory clustering algorithm is used to construct the trajectory bundle library.

In chapter 5 we focus on learning models to predict which trajectory bundles are feasible from a single monocular camera image. A nearest-neighbor prediction system is developed, and then enhanced by modeling the output of the predictor using a conditional random field model. We then develop a Bayesian filtering algorithm to incorporate the history of observations in the prediction of which trajectory bundles are feasible in chapter 6. The resulting feasibility estimates are shown to significantly improve the navigation performance.

In chapter 7 we present preliminary experiments where we apply some of the developed algorithms on a real world dataset. Finally, we discuss some conclusions, and present directions for future work in chapter 8.

Chapter 2

Related Work

Mobile robot perception and navigation are two of the fundamental challenges in the field of robotics. As such, there has been a considerable amount of research on each of these problems, as well as work that considers them jointly. In this chapter, we will provide some background on the approaches from the literature that have been previously developed. We will first discuss some commonly used representations for robot mapping and navigation, these are the functions $m(.)$ and representation M discussed in section 1.1. We then discus the approaches other authors have taken when solving developing robots that navigate through unknown environments.

2.1 Environment Representations

There are a large number of environmental representations in use in robotic systems today. The environment representation provides a way to predict various aspects of the world as a function of the vehicle location. Each potential representation will entail trade-offs, which may significantly impact the utility of the constructed environmental representation. As a result, it is important to take into account the purpose for building the map when thinking about these trade-offs. For example if the reason for the robot to build a map of the environment is to provide situational awareness to human rescue workers that will follow, then visual appearance may be more important than metric accuracy. For such purposes, a texture mapped planar surface model [97] (such as is often used in rendering

computer graphics) may be a good fit. Metric accuracy of the geometry is not particularly important, and much of the fine details can simply be represented by 2D image patches. Taken to the extreme, simply stitching together the acquired imagery, and projecting it onto a sphere such as is done in Google's Street View [2] can provide situational awareness. Other representations such as a collection of surfels [47] also provide results that are more visually informative, while not necessarily increasing the metric accuracy of the geometry.

On the other hand, if the map is being created to provide a survey of a building for planning and equipment layout purposes, then metric accuracy is of extreme importance. In such situations occupancy maps, or a polygonal representation of obstacles may be most appropriate.

Representations For Planning

For the purposes of this thesis, we are interested in the setting where the representation is being used for mobile robot navigation. This means that the representation will be used to understand where the vehicle can and cannot go, and to make decisions about the actions that the vehicle should take.

As such, we do not necessarily care about visual or metric accuracy in the representation as long as the representation enables a planner to choose a good path for the vehicle. An analogy can be made to the concept of sufficient statistics in that as long as the representation captures the necessary information about the environment for planning decisions, the exact form of the representation does not matter.

In addition to their utility for planning, each of the possible representations present trade-offs in terms of their ease of construction from sensor data. Certain representations, may be very easy to use in a planner, but difficult to create from noisy sensor measurements. On the other hand, representations that are simple to construct may not be easy to use for planning.

Finally, since we are interested in the situation where the vehicle will be creating its environment representation from noisy and ambiguous sensor data collected as it moves through the environment, and it will be desirable to take into account the uncertainty in our

estimates. Encoding this uncertainty is easier in some representations than others.

2.1.1 Point Clouds

Perhaps the simplest representation to construct from range sensor measurements is to simply aggregate the information into a point cloud. The measured ranges are projected from the current pose of the vehicle, to determine the location of the measured point. These points are then simply stored in a list, or other storage data structure.

While simple, point cloud representations do not summarize the raw underlying sensor data, which can result in very high memory usage. In addition, the point cloud only models the occupied regions of the environment, and does not model the free space. As a result, by itself, the map does not allow us to differentiate between regions that we know to be safe, and unknown regions.

In general, point clouds are post-processed to extract planes, or other geometric objects from the data to generate a more complete representation for planning [95]. However, probably the most common post-processing step is to bin the samples into grid cells such as is done for occupancy mapping.

A hybrid solution presented in OCallaghan and Ramos [91] reasons about range measurements as samples in a Gaussian process (GP). This provides the ability to reason about the occupancy of areas, without having to assume independence between all measurements, as is normally done for occupancy maps. Since the GP representation is continuous, it allows the generation of variable resolution maps. In addition, the GP provides a covariance measure that can be used to reason about the uncertainty in different areas. Smith et al. [109] use also use GPs, but instead of modeling occupancy, they reason about the surface measured by the range readings.

Vandapel et al. [118] develop a method for post-processing point clouds to fill free space with overlapping spheres. Rather than accurately estimate the surfaces of obstacles, they leverage the intuition that in many environments it is simpler to map and reason about the free space exclusively. The overlapping free space spheres create a network of tunnels through the environment. This network enables efficient online path planning with a low

memory footprint. However, it is unclear how one could extend the work to operate in an online situation instead of post-processing a complete point cloud.

Point cloud maps are one of the most commonly used representations in feature based simultaneous localization and mapping (SLAM) [31], especially for visual SLAM, which uses camera measurements [27]. In the computer vision literature, such techniques are more commonly referred to as structure-from-motion [117] or bundle adjustment [115]. In general however, SLAM algorithms only model distinctive feature points that can be reliably, and repeatably detected in the sensor data, such as corners. As such the maps are generally quite sparse. In fact, many algorithms are designed around explicitly leveraging sparsity in the problem [120]. Unfortunately, this sparsity means the maps built by SLAM algorithms are often not particularly useful for planning purposes.

Finally, perhaps the biggest issue with using a point cloud representation for our problem, is that points are very low level objects. They do not provide a good way to reason about areas that are not directly measured by either a range sensor, or by triangulating points in camera images. As a result, it would be very difficult to extrapolate beyond the directly observed ranges.

2.1.2 Occupancy Maps

Occupancy maps are one of the most popular, and commonly used representations in robotics [111]. Occupancy maps split the world into small discrete cells, and store the current belief about whether each cell is likely to be free or occupied. To allow for efficient updates and use, each cell in the map is normally reasoned about independently [82, 32]. This independence assumption is required due to the vast over-segmentation of the environment. Any individual cell is such a small piece of the environment, that if one wanted to reason about large scale structure in the environment such as whole objects or buildings, there would simply be too many variables involved for inference to be tractable. On the other hand, simply making the cells in an occupancy map larger, would create a representation that is too coarse to enable effective use for planning.

For sensors that provide accurate metric range information, such as LIDAR sensors,

the independence assumption between cells is not particularly problematic for mapping. In such situations, the likelihood of occupancy is simply computed as a ratio of the number of beams that pass through a cell compared to the number that reflect from it. This in effect simply averages the measurement information on a per cell basis. If a robot does not need information beyond the metric sensing horizon of the sensor, then these maps allow a robot to navigate effectively without considering the dependencies between cells in its occupancy map [100]. Exploration can be driven by looking for frontier regions on the map border between known free space, and unknown areas [123].

However, occupancy maps do not work as well for sensors that do not provide accurate metric range information, such as cameras. One can try to use algorithms to estimate the depth of each pixel in the scene (see section 2.2.4), allowing it to be treated as a (noisy) range sensor, however the uncertainty in these estimates is likely to be quite large. Very uncertain ranges are problematic due to the assumed independence between all cells. The simple averaging procedure used for estimating occupancy maps will not reflect the true structure of the environment when such noisy measurements are incorporated. Furthermore, the projection of the uncertainty present in the sensor measurements into the uncertainty over the occupancy probability will not capture the true uncertainty over the structure of the environment as observed from the camera images. This problem is further compounded when we then go to use the map for planning, as the final uncertainty in the feasibility of being able to execute a given plan will now be several algorithmic steps removed from the underlying perceptual uncertainty.

2.1.3 Polygonal Obstacle Maps

Many of the motion planning algorithms in the literature operate on environment models that are described using polygonal obstacles. Such models, are a compact and natural way to manually describe simulated worlds. They also often enable fast and efficient collision checking.

Unfortunately, while many man-made environments are roughly polyhedral, and can be compactly described using polygons (as is commonly done in computer graphics), gener-

ating a polygonal representation of the environment from sensor data is a very challenging problem. At a minimum, the world must be segmented into roughly planar regions (or line segments in 2D [18]), however, it may be preferable to segment further into discrete polyhedra [34]. This segmentation and fitting problem can be extremely challenging in the face of noisy and ambiguous sensor data.

This problem becomes even more difficult in an online setting where we wish to reason about the environment from the partial sensor data collected up to that point. Normally, the vehicle will only have observed the front side of obstacles, and will therefore have to infer what the backside of the obstacles might look like to create closed polygons. Thrun and Wegbreit [112] develop a method to search for possible symmetries in the observed range readings in order to reason about occluded regions, however the algorithm is limited to objects that are symmetric, and requires that the objects are accurately segmented from the background (or other objects). In addition, the models used are specific to range measurements. More often, authors build a complete occupancy map, or point cloud representation, and then fit a polygonal model to this complete map [1].

Finally, the geometry of polygons require inference to be performed on the mixture of the discrete number of faces, each with continuous parameters. The dependencies between these variables are nonlinear, and variables depend on a large number of other variables, which makes inference difficult. If we constrain the obstacles to be rectangular boxes, the models are simplified somewhat, and several authors have recently, explored using such models to reason about scenes observed by a monocular camera sensor [44, 45, 35].

Brooks [13] and Kuan et al. [70] flip the polygonal obstacle representation around, and instead directly represent the free space in the environment. This provided a number of benefits for the planning algorithms that employed the maps. However, the authors assumed that they are given an polygonal obstacle map as input. They do not discuss ways to create the free space map directly from sensor data. Our trajectory bundles take inspiration from these approaches, but seek to enable their creation online, without requiring a complete map ahead of time.

2.1.4 Topological Maps

The last type of commonly used representation in robotics are topological maps. Kuipers and Byun [71] provide a good reference, however many others have also pursued topological maps [20, 68]. Topological maps represent environments as a graph where significant places [24] are vertices. Edges denote the ability to traverse between two places. Edges are usually annotated with information on how to navigate from one place to another, or an applicable control law that will bring the vehicle from one vertex to the next. Alternatively, the topological map may string together a set of local metric occupancy maps [107, 11].

One of the major benefits of topological maps is that they provide a very abstract notion of the connectivity of the environment. The world can be reasoned about in terms of unique places, and connecting corridors. The exact geometric extent and shape of each of these is abstracted away. This enables very fast planning, and reasoning. They can be used to provide high level guidance for robot navigation, however on they must generally be combined with a reactive control law that can follow the edges and connections in the graph, or some other local metric representation.

In general, topological maps are effective representations for very large scale mapping problems, but are not used for local planning and obstacle avoidance. In this work, we focus on providing local guidance of the vehicle. In this regime, we need more structure than is normally provided by topological maps. However, the abstract environment representation is appealing. We have designed our trajectory bundle representation to take advantage of much of the abstraction provided by topological maps, while retaining enough metric structure that they can be used for local planning.

2.2 Planning Through Unknown Environments

We now turn to discuss work related to the problem of building models of the environment as the vehicle moves through it, and using this model for planning purposes.

2.2.1 Occupancy Map Based Navigation

Occupancy map based approaches have been employed in a very wide variety of systems. In ground robots, occupancy maps have been used since the early days of research on mobile robots [82]. Usually, the maps for a ground robot were represented as a 2D grid. More recently, they have also been applied applied in 3D on a number of air vehicle platforms.

Scherer et al. [100] developed a system that exemplifies the state of the art in metric occupancy mapping based approaches. The authors used a high-power custom-built 3D lidar sensor to sweep the area in front of the vehicle for obstacles. This sensor had a maximum range of 150 meters, and a 30×40 degree field of view. The authors track the state of the vehicle by fusing inertial sensors with D-GPS, and aggregated the laser returns into a 3D occupancy map. The map was used to plan paths for the vehicle. By updating the map online, and performing fast local replanning, it was also used for obstacle avoidance. The vehicle used a combination of a global planner for longer term guidance and a local planner for obstacle avoidance. The accurate, and long range lidar sensor allowed this approach to be successful, however it has some relatively major limitations in terms of the size and power of vehicle that is required. The authors used a Yamaha R-MAX helicopter which has a 3m rotor, and maximum payload of 31Kg. Such a large vehicle was required to carry the large, heavy 3D lidar sensor. Due to the active nature of the sensor, which must illuminate the scene with very bright laser light, it is unclear how much smaller and lighter such a system could be made.

Hrabar [53] built a system that also used a 3D occupancy map built online, but used range measurements from a stereo camera instead of the lidar. The maximum range at which obstacles can be detected, and placed in the map with stereo will severely limit the maximum speed of the vehicle, and result in myopic path planning. Andert and Adolf [1] also use a stereo camera to populate a 3D occupancy map that they built online. However, to limit the memory usage of the map, they used a sliding window approach for the 3D occupancy map, only storing areas that were in the immediate vicinity of the vehicle. They then fit an approximate polygonal model to the obstacles in the occupancy map to create a global map of the environment. For simplicity, they assumed a flat ground plane, and then

fit prism shaped obstacles around the convex hull of obstacles. This model of the environment would be useful for cases where the vehicle must backtrack through the environment, but does not improve online navigation performance.

All of these works in micro air vehicle navigation have been targeted towards helicopter platforms. This means that the vehicle has the option to go arbitrarily slowly, or even stop and hover. While Scherer et al. [100] highlight the fact that they are able to fly at speeds of up to 10m/s, they adjust the speed of the vehicle based on the distance to the nearest obstacle. As a result, when operating in and among obstacles, they are likely to be flying much slower. While the range of the lidar sensor is much greater than the stopping range of the vehicle, this caution is required since they make no efforts to reason about occluded areas. With stereo based approaches, the speed would likely need to be reduced even further. In this work, we are interested in developing techniques that would allow for significantly faster speeds, and longer range planning without necessarily requiring high power active sensors, such as lidars.

2.2.2 Optical Flow Based Obstacle Avoidance

Optical flow is a very popular approach for obstacle avoidance, owing to the use of lightweight and low power passive camera sensors. Such techniques are inspired by the navigation of insects, which have been shown to use optical flow extensively [110]. Optical flow based approaches analyze the apparent image motion as the vehicle moves through the environment. Generally, the closer an obstacle is to the imaging sensor, the greater the apparent image motion, or optical flow. A number of authors have presented systems that employ optical flow for state estimation and obstacle avoidance [4, 43, 125].

Beyeler et al. [7] present an autopilot system that uses a battery of modified optical mouse sensors to control a fixed wing vehicle. The measured optical flow rates of the suite of sensors are mixed together and mapped directly to the control inputs of the vehicle. The presented system appears to be impressively stable, and capable of natural obstacle avoidance abilities.

Optical flow can also provide other useful properties, such as the centering response

employed by honeybees. By balancing the apparent optical flow on either side of the vehicle, the vehicle is able to center itself in corridor type environments. Hrabar and Sukhatme [54] used such methods to develop a navigation system targeted for air vehicles in urban environments. The centering response was also investigated by Humbert et al. [56].

Optical flow is an important cue for navigation, however, on its own, it does not enable truly intelligent goal seeking navigation. Optical flow based approaches are better suited to detecting and avoiding large obstacles. Flying in and among obstacles, and performing maneuvers such as flying through a narrow opening, is unlikely to be achievable using optical flow alone. The nearby obstacles are likely appear similar to an obstacle that must be avoided, and higher level reasoning would be required to override the avoidance response that would normally be triggered. In addition, the control laws that result usually must wait for the flow to get large (collision is imminent) before turning to avoid. As a result, they will not be able to allow a vehicle to deliberately understand and avoid an obstacle from long range.

While we do not use optical flow features in our implementation currently, such features would likely be very informative, and could easily be incorporated. When combined with higher level reasoning, optical flow could help enable much more advanced navigation capabilities.

2.2.3 Mapping Sensing To Actions

A number of other researchers have investigated approaches that learn ways to map sensor measurements directly to actions. Pomerleau [93] trained a neural network model to follow a road lane. The neural network directly predicts the desired steering angle from the input camera image. The neural network was trained on data from a human driver operating the vehicle. At first blush, this problem seems similar to ours, however the lane keeping task is a very local decision. If the vehicle is straying to one side, the vehicle should turn back to re-center itself. As a result, it is more reasonable to directly map the perception to a one step action. In contrast, we seek to develop goal directed behavior in unstructured environments. The goal is not necessarily obvious from the sensor data, and we must take into account

decisions further into the future. Training a classifier to predict which action to execute, accounting for actions that may be taken in the future is a much more challenging problem. Solving this problem may be possible with sufficiently powerful learning algorithms and enough training data, but today's learning techniques are unlikely to generalize very well.

In our approach, we provide a significant amount of structure to the learning problem in order to make it more tractable. We break the problem into learning a mapping from the sensor data to the feasibility of the trajectory bundles, and then make decisions based on these predictions. This structure allows the learning problem to be more tractable with available learning methods.

Michels et al. [81] employ a similar breakdown of the problem by first developing a system that estimates the range to the nearest obstacles, and then learning to map these predictions into a steering direction. The authors use supervised learning to estimate the range to obstacles from monocular image cues in 2D. The output of their system ideally approximates a planar lidar sensor. They then use a reinforcement learning algorithm to choose a steering angle based on the output of the range predictions. The authors showed that the system was able to guide a remote control car through a wooded area at relatively high speeds. Once again however, the system is only trained to avoid obstacles. It is unclear whether more complicated decisions with a multi-step lookahead could emerge.

Boots et al. [10] develop a system that uses a predictive state representation to build a model of the environment that enables it to make planning decisions. The model is learned directly from the history of action-observation pairs. In their case, they test the system on a navigation problem, using simulated camera images as the observations. While this approach allows the vehicle to navigate towards a goal using purely monocular camera observations, the work focuses on the situation where the vehicle navigates in the environment for a number of episodes to build the model. As such, the predictive state representation takes the place of a map of the environment. The approach does not seem likely to generalize to a new unknown environment where the vehicle has never been, without providing time for it to collect data, and train the model.

2.2.4 Depth From a Monocular Image

There has been a significant amount of work on triangulation based approaches for estimating depth from multiple images [26, 99], however such approaches are limited by the baseline (distance) between images in their ability to resolve depth. In practice, triangulation based methods are limited to ranges of 20 to 30 meters [80, 106]. Furthermore, these ranges will only be computable for distinct and recognizable regions in the image, such as corners.

We believe that it will be critical for robots to be able to leverage the full richness of information available in a single camera image. Humans use a combination of binocular stereopsis and monocular cues to perceive depth, however beyond a few 10's of meters, the impact of stereo triangulation is minimal.

Initial work on 3D reconstruction from a single image was developed by Horn in the 70's [51]. He developed approaches to estimate the shape of the imaged surface from its shading. Since then, significant advances in shape from shading techniques have been made [124], however such algorithms are only applicable in very specific settings. These methods generally assume uniform color and texture, and even that the surfaces are Lambertian. When these conditions are met, shape from shading can be extremely accurate [59]. However, the algorithms do not generalize to complex, real-world scenes. While shading cues may not provide enough information in isolation, they can certainly be used as features in more general learning algorithms.

In recent years, there has been significant research on estimating the depth of a scene from a single image using machine learning approaches. Torralba and Oliva [114] analyze the texture, and other structure in images to predict the mean depth of an image. Subsequently, several authors such as Hoiem et al. [48] and Delage et al. [28] developed algorithms that model the world as a flat ground-plane with vertical walls. Relaxing assumptions somewhat, Saxena et al. [97] developed a system that infers the 3D geometry of a scene as a triangle mesh. While these approaches are often able to produce visually appealing "pop-up book" type models, the metric accuracy of the predictions would cause problems for planning algorithms seeking to predict the effect of actions. Furthermore,

they do not consider occlusions, and therefore cannot represent openings through which the vehicle could travel.

Since the publication of these initial works, there has been a flurry of further research on related problems. Hoiem et al. [49] extended their original work to reason about occlusion boundaries more explicitly. Lee et al. [76] exploit a Manhattan-world assumption about the environment to reconstruct indoor scenes. Flint et al. [35] combine the Manhattan world monocular cues with 3D information from stereo and structure from motion to infer the layouts of indoor environments.

More recently, a number of researchers have started investigating algorithms that construct a more complete 3D model of environments. Gupta et al. [44] leverage the blocks world assumption where objects have volume and mass. This allows them to consider whether the predicted geometry has physically consistent structure in terms of the support, and stability of the model. The authors use physical simulation of the hypothesized configurations to test for validity. Hedau et al. [45] develop algorithms that seek to estimate the free space in a scene from a single image. The authors fit box shaped regions to the major furniture and other obstacles in the room, and compare their predictions against a corpus of images for which the complete 3D geometry is known.

Nabbe et al. [88] use the system from [48] to allow an autonomous ground vehicle to plan more direct paths through an unknown outdoor environment. By assuming a flat ground plane, and known intrinsic and extrinsic camera parameters, they are able to project the output of their photo pop-up model into a 2D Cartesian frame occupancy map that was used for planning. This approach is very similar to many of the systems developed for the LAGR program (see section 2.2.7).

While all of the approaches described above make significant inroads into the problem of scene understanding from a single monocular image, the algorithms are mostly applied in isolation, and not used for robot navigation. The main result of most of the papers are qualitative results showing visually plausible renderings of the scene from a different vantage point than the original image was taken. Many of the renderings look reasonable, however the textured mesh often hides inaccuracies.

Finally, all of the algorithms will have a significant amount of uncertainty in terms

of their predictions of the environmental structure. As a result, developing a planning algorithm that effectively uses these uncertain predictions for decision making purposes is still a very difficult problem. For example, Kurniawati et al. [72] recently presented an algorithm for solving POMDPs with uncertainty in a polygonal map. Such a map is the most that one could hope that the visual reconstruction algorithms might provide, and the algorithms are still very computationally intensive, even for relatively simple environments.

We believe that it will be more effective to consider the perception and planning problem together, by reasoning about the world in terms of trajectory bundles. The approaches developed for estimating depth from monocular images will provide useful insights into learning approaches and other techniques, but by themselves, do not solve our problem.

2.2.5 Trajectory Libraries

Planning with trajectories libraries has become a standard tool in robotics. It is normally used to allow fast local planning and collision avoidance. To the best of our knowledge a trajectory library has not been directly used as a representation for perception tasks. Instead, the systems that use trajectory libraries generally build a separate occupancy grid or other map representation, and then perform collision checking in this map.

As such, the trajectory libraries are generally used to reduce the computational effort of searching for a trajectory in a continuous action space. The trajectories in the library represent discrete samples from the action space of the robot that are computed offline, and known to be dynamically feasible.

Initial work on trajectory libraries used simple constant curvature arcs. These trajectories generally corresponded to choosing a velocity and steering angle [15]. The theory surrounding their use was further analyzed in Kelly and Stentz [65]. More recently, for the DARPA Urban Challenge, von Hundelshausen et al. [119] used a trajectory library based approach for obstacle avoidance, calling the trajectories “tentacles”. Howard et al. [52] sampled candidate trajectory sets during operation to bias the search towards useful regions of the state space.

Much of the work on planning with trajectory libraries has focused on constructing

good trajectory libraries [42]. Branicky et al. [12] develop the concept of path diversity and use it as a metric to analyze the suitability of different trajectory libraries. They define path diversity such "that a particular subset of paths (of a fixed size) maximizes path diversity if that subset is the most robust to the maintenance of feasible paths regardless of the placement of obstacles" [12]. Erickson and LaValle [33] build on this by developing algorithms that employ the notion of survivability to estimate the diversity of a trajectory library. Recently, Dey et al. [29] leverage the sub-modularity of the problem of searching for a good trajectory library to efficiently select an effective trajectory library. Knepper and Mason [66] argue that path diversity in the static sense is actually not an appropriate metric for selecting a trajectory library. They discuss the need to consider the dynamic situation where the planner will be continually replanning before the trajectory is executed to completion.

In this thesis, we use some of the techniques from the above body of work on designing and using trajectory libraries in the creation of our underlying trajectory library. However, we are less worried about making the trajectory library as sparse as possible. Our work focuses on using the trajectory libraries as an organizational concept for perception, rather than to accelerate local planning. As a result, we care more about coverage, and rely on clustering the trajectories into trajectory bundles to achieve compactness.

2.2.6 Trajectory Prediction

There has been some prior work on predicting entire feasible trajectories from sensor data. Jetchev and Toussaint [58] and Berenson et al. [5] developed a system that predicts trajectories as a way to speed up motion planning, while Goldfeder and Allen [41] developed a data driven system to speed up grasping. The authors note that most motion planners start over from scratch every time they receive a new planning task, or the environment changes. Often however, the new planning problem will be similar to the previous one, or another planning problem that has been solved previously. In such situations, the computation time can be significantly reduced by leveraging this past work.

The authors developed data driven planning systems that allow them to reuse old plans.

When confronted with a new planning problem, they would find the most similar planning problem that was solved previously. The retrieved plan is then used to initialize the solution for the planning problem at hand.

The primary challenge in reusing a previously computed plan is to develop a scene-descriptor that compactly captures the relevant aspects of the planning problem at hand, such that similar planning problems can be identified. Jetchev and Toussaint [58] compute a coarse voxel grid of the obstacles in the environment, and then use the principal components of the voxel grid with the highest variance as a descriptor of the scene. In contrast Berenson et al. [5] simply use the relative location of the start and end pose of the manipulator. Goldfeder and Allen [41] use visual features quantized into a bag-of-words descriptor [108] extracted from views collected of the object to be grasped.

The second challenge is to develop techniques for using the retrieved plan in the current problem. Normally, the plan will be similar, but will not actually solve the problem at hand. As such, some form of plan repair, or modification is required to adapt the plan to solve the problem at hand. Jetchev and Toussaint [58] transfer the planned path in task space instead of the joint space of the robot arm they worked with. The resulting path was then optimized using a trajectory optimizer. In contrast, Berenson et al. [5], simply use an RRT to repair paths that collide with obstacles. The retrieved paths are broken up into collision free segments, and then multiple planning tasks are dispatched to reconnect these pieces.

Approaches similar to these saw a considerable amount of research under the name of "case-based reasoning" [75] in the 90's. More recently, such approaches have generally been called "data-driven". Most of these approaches have been applied to problem of reducing the computation time of motion planning algorithms. However, as computers have gotten faster, and conventional motion planning algorithms have gotten more efficient [92] improvements provided have not tended to be great enough to warrant the added complexity. This is in part because the algorithms were generally only used to provide an initial guess for a complete planning algorithm, so researchers could often make larger improvements by just employing a better planning algorithm.

In contrast to this previous work, we use trajectory prediction to overcome sensing limitations. While the sensors will continue to improve, and open up new possibilities

(consider the recent revolution to robotics brought about by the availability of the Microsoft Kinect), sensors have tended to improve much more slowly than computing power. In addition, we seek to address limitations that are inherent aspects of the sensing modalities. Such challenges will not be eliminated by better sensors.

2.2.7 DARPA Learning Applied to Ground Robots Program

In this thesis we focus on extending the planning horizon for vehicles navigating through unknown environments. The DARPA Learning Applied to Ground Robots (LAGR) program [57] fostered research at a number of universities on a problem that is very similar to the one we address, but focused on ground vehicles navigating outdoors over rough terrain. The goal of the LAGR program was accelerate "progress in autonomous, perception-based, off-road navigation in unmanned ground vehicles (UGVs) by incorporating learned behaviors." The program provided a uniform test vehicle to a number of universities. The vehicle was only equipped with two stereo pairs as the main exteroceptive perception sensors. The teams used stereo triangulation to label traversable regions in the near-field, to make local navigation decisions.

While each of the teams developed their own unique system, there were a number of commonalities in the approaches. Almost all of the teams sought to leverage machine learning techniques to interpret the rich information in monocular camera images to extend the perceptual horizon beyond the range at which the stereo pairs provided useful information from triangulation. Generally, the models were trained in a near-to-far paradigm, where the short range metric information was used to provide labels for supervised learning methods [78]. In general the algorithms would output a per-pixel labeling of the image into one of several terrain types, which corresponded to different levels of traversability. The output of the classifier is projected into a 2D euclidean space by assuming that the vehicle is navigating on a flat ground plane. This assumption allows them to extrapolate from the pixel coordinates in the image, out to the coordinates in the real world. The Journal of Field Robotics has a pair of special issues focused on the LAGR program that provide a great overview of all the approaches from the different teams [55].

The researched performed for the LAGR program is a useful reference for the planning tasks discussed in this thesis, however they are not directly applicable. All of the approaches developed focused exclusively on ground vehicles, and made extensive use of the assumption of a flat ground plane. For a flying vehicle, we cannot make this assumption as the vehicle is capable of flying over shorter obstacles. As a result, we must take a more complete 3D view of the environment modeling task.

While modeling the full 3D environment makes the problem significantly more challenging, we do not need to concern ourselves with estimating traversability of irregular terrain. For an air vehicle that should never make contact with the environment, traversability is binary quantity. This means that it may suffice to only estimate the gross environmental topology, rather than detailed metric information. In contrast, much of the learning effort in LAGR was devoted to identifying the image statistics of traversable regions. It was critical to differentiate between tall grass or brush, since the vehicle could get stuck if the latter was too tall.

The trajectory bundle representation allows us to reason about the environment at a level of abstraction that is appropriate for an air vehicle. Importantly, it does not require us to make assumptions about the world, such as a 2D ground plane, as was done for the LAGR systems.

Chapter 3

Trajectory Library Based Planning

We are interested in developing algorithms that enable vehicles with highly constrained dynamics to navigate in unknown environments. We wish for the vehicle to be able to find its way using only compact and existing sensors carried onboard the vehicle. As a model of such a system, we investigate navigating a fixed wing vehicle through an unknown environment. In this work, we assume that while the vehicle has not explored its current environment, and does not have a prior map, it has access to training data from similar environments, for which it has a map. For example, even though it may have never navigated around the city of Cambridge before, it would have large amounts of training data from other cities around the US where it had accurate 3D models of all the buildings. Our models seek to allow the vehicle to generalize from the experience gained on this training data to the decisions it must make in Cambridge.

3.1 Simulation Model

To explore this problem space, we have developed a simulation environment that allows us to generate training data and test our algorithms. We also developed a baseline trajectory library based planning system that allows us to evaluate the importance of perceptual limitations on the planning performance of the vehicle.

The key design decisions for the simulation environment entail the vehicle model, the sensor models, and finally the world model. Of these, the most important decisions relate

to the world model. A completely random world model would provide minimal information that could be learned to improve navigation. In such a setting, the local information provided by the sensors on the vehicle would not allow it to extrapolate beyond what was immediately measured. On the other hand, if there is too much structure in the world model, such as a perfectly regular grid, then trivial algorithms such as locally greedy approaches could perform optimally.

The simulation model that we develop is motivated by the problem of a fixed wing micro air vehicle navigating through an urban environment. While a simulated environment lacks many of the challenges faced by perception and planning algorithms in real world environments, it will allow us to carefully understand the trade-offs made by using our trajectory bundle representation. In addition, it is still significantly more challenging than the simple "grid world" planning tasks that are often considered in partially observable planning problems [105]. For simplicity, we develop the simulations around a 2D vehicle model, however nothing in our formulation explicitly assumes a 2D world. For example, we do not assume that pixel labels can be projected onto a ground plane as has been done in much of the previous work on image based planning [57] that came out of the DARPA learning applied to ground robots (LAGR) project.

3.1.1 Vehicle Model

We target a vehicle similar to a small fixed wing airplane. While such airplanes have fairly complex dynamics [85], to a first level approximation, we can assume that the vehicle will travel at a constant altitude, constant speed, and is subject to a minimum turning radius constraint. Such models are often characterized using the two-dimensional Dubins car model [30]. Chitsaz and LaValle [19] extended the Dubins model to 3D by considering a kinematic vehicle that always flies forward and has independent bounded control over the altitude velocity. Such a model could be employed in future work that considers planning paths for the vehicle in 3D.

The Dubins car model is particularly convenient since the time-optimal path between two poses can be computed analytically. The optimal path will always consist of only

straight segments and minimum turning radius curves. This enables its use in optimizing sample based motion planning algorithms that require the ability to exactly connect samples [63, 64].

We assume that the vehicle has a 2m turning radius, and fits within a 1m ball. The circular shape assumption is convenient because it allows us to plan in configuration space [79] by simply dilating the obstacles in the map by 0.5m. More complicated vehicle shapes would make collision checking more computationally demanding, and could make learning feasibility more difficult, but would not change the formulation of the problem. We further assume that the vehicle is able to track the Dubins paths exactly. For a real system, we would need to develop a controller that could follow the selected path. The path following controller would inevitably incur some tracking error, however, mitigating such issues are not a focus of this thesis.

3.1.2 Sensor Models

We assume that the vehicle will carry two primary perception sensors: a short range 2D laser range-finder, and a monocular camera sensor. In addition, we assume that the vehicle has access to its global position in the environment though an oracle. On a real system, this position would need to be estimated from noisy sensors such as GPS, or via a simultaneous localization and mapping algorithm [3], however for the purposes of this thesis, we assume that the state of the vehicle is exactly known (the state of the environment is however, unknown).

The laser sensor is simulated by ray tracing inside of a 2D occupancy map of the world. We use the parameters of a Hokuyo URG-04LX sensor [50]. This sensor has a maximum range of 5m, and a 240° field of view.

We simulate the camera sensor by rendering the environment structures described below using OpenGL. The simulator renders 640×480 pixel color images. We assume a projective camera model with a 90° field of view. No distortion, variable lighting, or other noise is added. An example simulated image is shown in figure 3-1(b). The focus of this thesis is on the representation and higher level modeling tasks rather than the low level vision tasks.

Implementation on a real system would likely require significant further investigation of better image features, and other low-level perceptual learning algorithms.

3.1.3 Environment Model

We created test environments by sampling from a generative model. We dub this environment model the "Tetris-World" due to the chosen shape of the obstacles in the environment. The worlds consist of randomly located non-overlapping Tetris pieces. Each Tetris piece consists of four 5m cubes. The cubes are arranged in one of five possible shapes named with letters I, S, Z, J, C, and O (we omit the "T" Tetris piece). Each piece can be rotated to face along one of the four cardinal directions.

Each world spans a $400\text{m}\times200\text{m}$ region, and contains 200 pieces. The number of pieces can be changed to vary the average obstacle density. The boundaries of the world are lined with "I" pieces to create a surrounding wall. We then sample a type, position, and orientation for each of the 200 obstacle pieces. The positions are sampled uniformly at random from a grid with 5m spacing. The orientations are sampled uniformly from the four cardinal directions. If the sampled pose of a piece causes it to overlap with a previously generated piece, it is rejected, and a new sample is drawn. An example Tetris-World is shown in Figure 3-1(a).

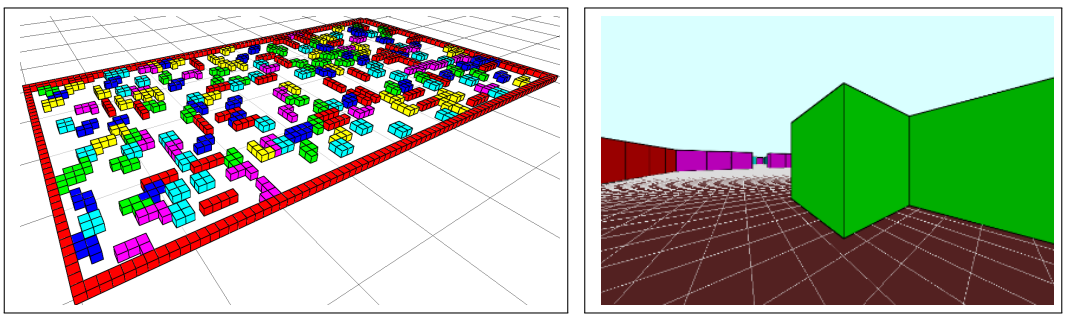


Figure 3-1: (a) An example of one of the randomly generated Tetris-World environments. (b) An image rendered from the vantage point of the vehicle.

3.2 Baseline Planning System

As a baseline to compare our algorithms against, we developed a local navigation strategy based on a precomputed set of trajectories [36] to make navigation decisions.

At each time step the vehicle updates its environment representation with the current sensor measurements. It then performs collision checking for each of the trajectories in the trajectory library. Since the local trajectories in the library do not extend all the way to the goal, a cost-to-go heuristic is used to evaluate the cost required to take the vehicle the rest of the way to the goal. This heuristic cost-to-go is added to the cost of following the trajectory to compute the total cost for each trajectory. The vehicle follows the trajectory with minimal cost from the set of collision free trajectories. Usually, the trajectory is only followed for a portion of its length before a replan cycle occurs, and a new trajectory to follow is selected. The complete planning process is presented in pseudocode in algorithm 1.

3.2.1 Trajectory Library Generation

Much of the previous work on planning using a trajectory library has focused on selecting an effective set of trajectories that enables good navigation safety with minimal computational effort [42, 66, 29]. In contrast, we will be using the trajectories as an organizational paradigm for understanding the layout of the environment. We focus on overcoming perceptual limitations rather than the efficacy of trajectory libraries, so we simply employ a dense library of trajectories.

The planning horizon that we use extends out to a range of 30m, which means that it is infeasible to generate a "complete" trajectory library by discretizing the action space of the robot, as was done in Green and Kelly [42]. The number of possible trajectories is exponential in the planning horizon. So, even for a relatively simplistic vehicle model, such as a Dubins car model, which can be thought of as having 3 actions (left, straight and right), if we discretize 30m trajectories into 1m steps, we would need to consider 3^{30} possible trajectories. Furthermore, if the turning radius of the vehicle is less than the planning horizon, many of these trajectories will double back upon themselves, and are not useful to consider.

Instead of discretizing the action space of the robot we sample useful paths using a motion planning algorithm. This approach is similar in spirit to the one employed by Sermanet et al. [104]. Using an optimizing motion planning algorithm, we can sample random poses in the map, and then plan an optimal path to a random goal location. The output path is then cropped to the desired planning horizon.

To generate our trajectory library we leverage the multi-query capabilities of algorithms such as the probabilistic roadmap [64]. For a given environment, we generate a single very dense graph. Once this graph is generated, to generate a sample trajectory we sample a vertex from the graph, and then run Dijkstra's algorithm from that vertex to a random goal vertex.

We use the above sampling approach to generate a large set of candidate trajectory samples. Each sample has the nice property that it was part of the (approximately) optimal path between two points in a representative environment, thereby eliminating useless paths such as those that drive in circles. To prevent duplicate samples, a trajectory is only added to the trajectory library if the distance between the candidate trajectory and all existing trajectories in the trajectory library is greater than a threshold[1].

Since we are focused on planning problems for what is in front of the vehicle, we place a further constraint on the trajectory library that enforces that the trajectories do not double back upon themselves. At no point along the trajectory may the absolute value of the heading be greater than $5\pi/6$. The resulting dense trajectory library containing 4707 trajectories is shown in Figure 3-2. When none of the trajectories in the forward looking trajectory library are deemed to be feasible, we employ a separate trajectory library consisting of only backtracking trajectories. This trajectory library will be described in more detail below.

3.2.2 Mapping

As the vehicle navigates through the environment, it aggregates the measurements from its local 5m laser sensor into an occupancy map. This map is then used to perform collision checking of the trajectory library. Since the 30m trajectories will often extend beyond the

[1] We discuss distance metrics for trajectories in section 4.1

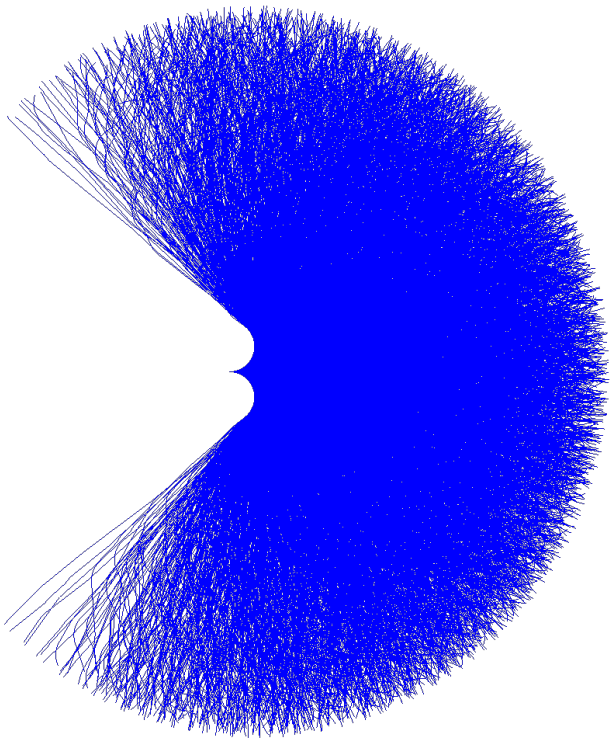


Figure 3-2: The dense trajectory library used for planning. Each trajectory is 30m long.

perception horizon provided by the lidar sensor, map cells that are labeled as unknown, are treated as if they are free. Otherwise, in almost all situations, none of the trajectories would be deemed to be feasible.

3.2.3 Backtracking

We designed the trajectory library to only include trajectories that make forward progress. As a result, the trajectory library does not include trajectories that double back on themselves. If all of the trajectories in the trajectory library are found to be in collision with an observed obstacle (or later on in the thesis predicted to be in collision), we fall back to a backtrack policy. In these situations, we have the vehicle choose a trajectory from a separate set of trajectories that double back upon themselves. The backtracking trajectory library is shown in Figure 3-3. The trajectories are collision checked in the current map, and then the feasible trajectory with the minimum cost to go is selected.

The collision checking is first performed conservatively, treating unknown regions in

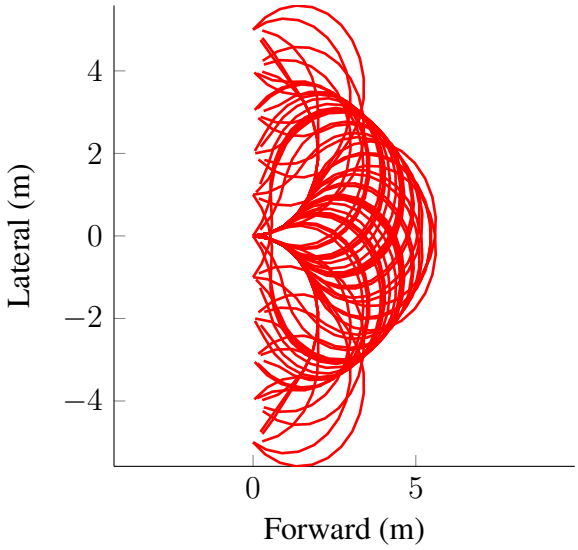


Figure 3-3: The backtracking trajectory library used when none of the trajectories in the normal library are deemed to be feasible.

the occupancy map as being obstacles. If none of the backtrack trajectories are classified as feasible, the collision checking is run again, this time treating unknown regions as free space.

3.3 Baseline Planning Performance

We use the trajectory library based planning system described above (and summarized in pseudocode in algorithm 1) to investigate the effect on the navigation performance of a number of different perception configurations. These experiments provide a baseline for the navigation performance for a vehicle with a range sensor capable of different maximum ranges. In subsequent chapters, we will perform similar experiments, but using only a monocular camera sensor, and show that our predictions enable similar performance to some of the longer range configurations.

We task the vehicle with navigating between two randomly selected points that are 150m apart on average. The map of the environment is generated at random as described above. If the vehicle reaches the goal, we deem the trial a success. Otherwise, if the vehicle collides with an obstacle, or fails to make progress towards the goal for a sufficient number of steps, the trial is deemed a failure.

Algorithm 1 Trajectory Library Based Planning

Require: A forward looking trajectory libraries L and backtracking library B
Require: The distance between replan steps r
Create a local occupancy map M, initially set to unknown
while goal not reached **do**
    Update M with most recent laser measurement
    Determine feasibility of all trajectories in L and B according to M
    if any of trajectories in L are feasible **then**
        Select trajectory in L with minimum total cost (according to the heuristic cost-to-go)
    else if any of trajectories in B are feasible in map **then**
        Select trajectory in B with minimum total cost
    else
        return failure
    end if
    Execute first r steps of the selected trajectory
end while
return success

We evaluate the effect of the sensing horizon and field of view on the navigation performance by performing an experiment where we augment the sensing suite described above with an additional range sensor. We test sensors with either a 90° or 270° field of view, and vary the maximum range between 10m and 30m. As a final baseline, we also compare against a vehicle that has access to the entire ground truth map ahead of time. With this information, the vehicle would be able to compute a full global plan ahead of time, and therefore be able to identify a feasible path if one exists, however we wish to obtain a baseline indicating the best that a vehicle could do with the simple trajectory library navigation strategy described above.

As vehicles move faster, their dynamics become more constrained, and they must look further ahead. To analyze this effect, we vary the distance between replan steps between 1m and 11m. When it replans every 11m, that means that the planner must choose a path that is collision free for at least 11m. Even if the chosen trajectory does not collide with an obstacle during the replan horizon, the vehicle must also be left in a position where it is able to maneuver to avoid obstacles. As a result, while we assume that the vehicle is able to perfectly follow the chosen trajectory, the vehicle may find itself left in a situation where a collision in unavoidable.

Figure 3-4 plots the fraction of trials that are successful against the distance between

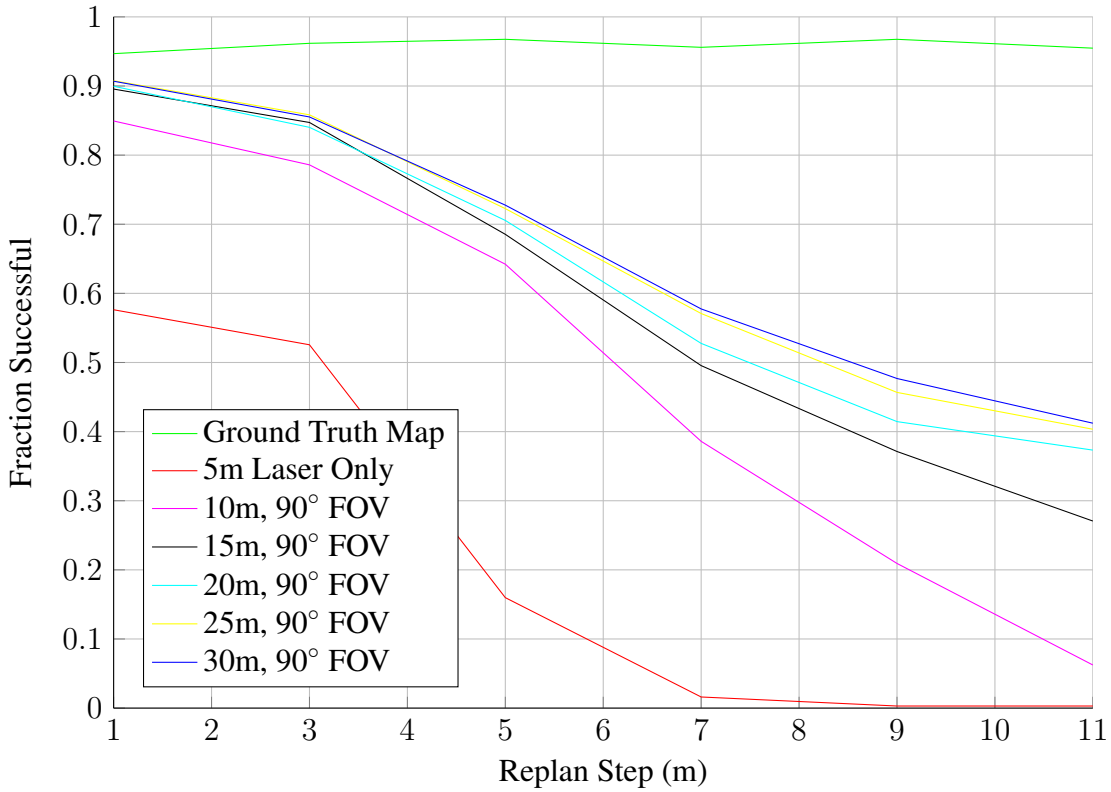


Figure 3-4: A comparison of the success rate in reaching the goal with different sensing capabilities. As the re-plan distance gets larger (as when the vehicle is moving faster) the vehicle requires more sensing information to navigate safely. However, even with a 30m range sensor, there is a significant gap in performance compared to when the vehicle has access to the ground truth map. This is due to the limited field of view, and the effect of occlusion in limiting the perception horizon.

replan steps. At the top of the plot, we can see that when provided with a ground truth map of the environment, the vehicle is able to reach the goal roughly 96% of the time. The failures in this case are not due to perception failures, but due to limitations in the planning approach. Most of these failures represent situations where the vehicle gets stuck in a large cul-de-sac like area in the environment. If a region is deeper than the 30m planning horizon provided by the trajectory library, then the vehicle will get stuck and be unable to make progress. Such situations would normally be handled by a higher level global planner that changes the estimated cost-to-go value for the end of each trajectory in the trajectory library to take into account the observed areas of environment. However, since our focus is on the perception, we omit the complexity of adding a higher level planner, and just accept

that the vehicle will get stuck on some trials, even with perfect perception.

When the vehicle navigates without access to the ground truth map, it is clear that the sensor capability has a profound impact on the navigation performance. With only the 5m laser rangefinder sensor, the vehicle is only able to reach the goal roughly 57% of the time when it is able to replan every 1m. The success rate falls off quickly with increasing distance between replans, falling to a 15% success rate when it can replan every 5m. The decline in performance is to be expected, since the vehicle must follow the chosen trajectory all the way to its sensing horizon, and has no information about what may lay beyond. If an obstacle is just beyond the sensing horizon, the vehicle will not have space to maneuver to avoid the obstacle on the next replan cycle.

As we increase the maximum range of the long range sensor to 30m, the success rate of reaching the goal increases significantly. When the vehicle is allowed to replan every 1m, it appears that a 10m sensor range is nearly sufficient to navigate as well as the ground truth, however, as the replan distance is increased, performance drops off significantly.

In addition to the range of the sensor, the field of view also has an important effect on the navigation performance of the vehicle. Figure 3-5 investigates this issue by comparing the navigation performance between a vehicle with a sensor that has a 270° field of view compared to one that has a 90° field of view. The gap between the success rate of the vehicle with the two sensors can be directly attributed to limited field of view of the 90° field of view sensor.

Interestingly, even with a 30m range laser sensor, which is the same range as all of the trajectories in our trajectory library, the navigation performance is significantly worse than the performance when the vehicle has access to a prior map of the environment. This performance difference can be attributed to the effect of occlusions in limiting the effective sensing horizon of the vehicle. The regions around corners, and behind obstacles will remain unknown even though they are within the 30m range of the laser sensor. As a result, the vehicle will still venture into unknown regions even with the longer range sensor.

In the following chapters, we will conduct similar experiments, but using the predictions from our perception system to estimate the feasibility of each trajectory in the library. We will show that by using the information in monocular camera imagery, we are able

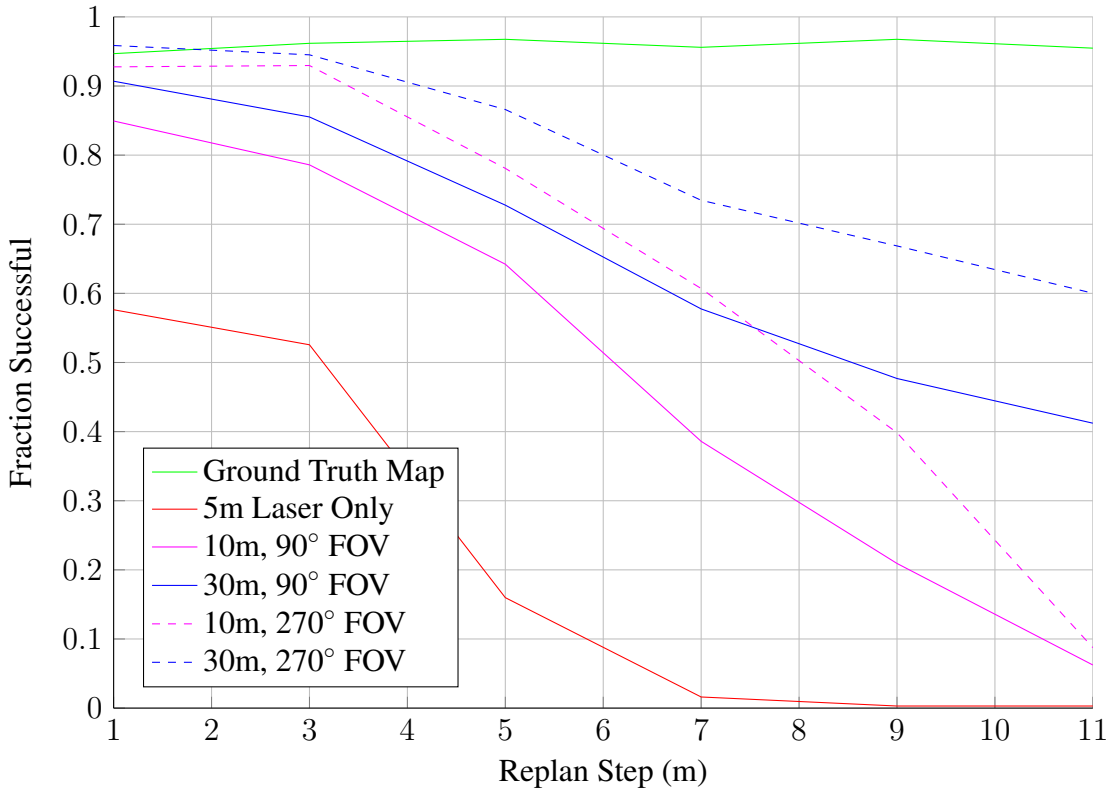


Figure 3-5: A comparison of the success rate in reaching the goal with different sensing capabilities. When the vehicle has access to a sensor with a larger field of view, the success rate increases dramatically. However, there is still a large gap relative to the performance with the ground truth map due to the effect of occlusions.

to significantly outperform the navigation performance using the short range laser alone, and perform almost as well as the long range laser sensors. Finally, we will show that by combining the predictions based on the monocular camera information with the collision checking afforded by long range sensors, we can do better than with range information alone.

Chapter 4

Trajectory Bundle Library Generation

In the introduction, we described trajectory bundles as sets of similar trajectories that can be used as an abstract concept to improve prediction of feasibility for planning purposes. However, we did not say how one would go about generating these sets of similar trajectories, or even what defined similarity. Indeed, the problem of clustering trajectories is challenging due to the often-complicated process of generating feasible trajectories. Many clustering algorithms require the ability to compute the average of a set of samples, however this quantity is undefined for trajectories. In fact, even defining a similarity measure between trajectories is not obvious. This chapter focuses on algorithms for clustering sets of trajectories. We will use the developed clustering algorithm to generate the set of trajectory bundles in our trajectory bundle library.

We develop a clustering algorithm based on the affinity propagation algorithm [37]. Affinity propagation only requires a way to compute the similarity between two trajectories, and does not directly reason about the underlying phenomenon that generated the trajectory samples.

The trajectory clustering algorithm that we develop is presented as an example of an approach for clustering a trajectory library into a set of trajectory bundles. Other methods for clustering trajectories [60, 17, 83, 84] could also be employed, or the process could potentially be done manually.

4.1 Trajectory Similarity Metrics

In the most general form, a trajectory is a continuous function mapping from time to locations in the state space of the robot. While the underlying trajectory of the robot is a continuous function, it is often difficult to work with continuous forms of the trajectory. As such, we will approximate the trajectory as a discrete set of points along the trajectory.

$$\tau = \{s_1, s_2, \ldots s_n\} \tag{4.1}$$

where s_t is the state of the robot at time t and n can vary with the duration of the trajectory. The assumption is that the continuous trajectory is discretized at regular time intervals.

Perhaps the simplest similarity metric one can use is to compute the distance between sequential points along two trajectories.

$$D(\tau^a, \tau^b) = \sum_t ||s_t^a - s_t^b||_2 \tag{4.2}$$

The summation in equation 4.2 can be replaced by a MAX operator if one only cares about the maximum deviation between two trajectories.

If the sampling interval of the two trajectories does not match, or is not uniform in the distance traveled along the trajectory, then a better similarity metric is obtained by normalizing the distance between the trajectories by the distance traveled along each trajectory. In two dimensions, this can be thought of as approximating the area between the two trajectories.

$$D(\tau^a, \tau^b) = \sum_t \frac{||s_t^a - s_{t-1}^a||_2 + ||s_t^b - s_{t-1}^b||_2}{2} ||s_t^a - s_t^b||_2 \tag{4.3}$$

Finally, there may be situations where one wishes to penalize differences in certain parts of the trajectory more than others. For example, in the trajectory bundle libraries that we will be using in place of an environment representation for planning, we will care more about differences close to the vehicle. To achieve this effect, we can add a scaling term

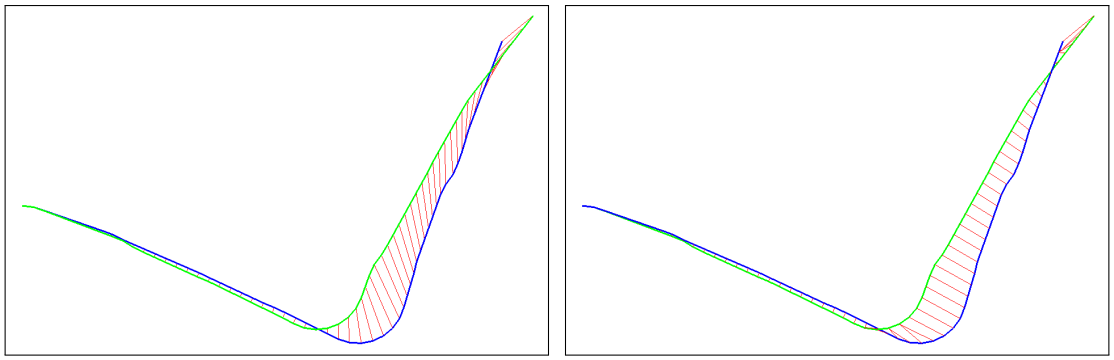
$\alpha(t)$ that scales the distances between points along the trajectories accordingly.

$$D(\tau^a, \tau^b) = \sum_t \alpha(t) \frac{||s_t^a - s_{t-1}^a||_2 + ||s_t^b - s_{t-1}^b||_2}{2} ||s_t^a - s_t^b||_2 \tag{4.4}$$

we use a scaling function of $\alpha(t) = 1/\sqrt{t}$

4.1.1 Dynamic Time Warping

While the above time indexed residual metrics are simple and efficient to compute, they provide poor results in some common situations. For example, consider the two trajectories in Figure 4-1. The two trajectories are roughly the same shape, but one turns a bit earlier. Because the blue trajectory turns after the green one, if we correspond the points along the trajectory simply based on time, points that are further from the turn in the green trajectory will be matched with points closer to the turn on blue trajectory. In other words, the sample points will be out of phase, as seen by the longer diagonal red lines in figure 4-1(a). This results in a larger distance than is appropriate between two otherwise very similar trajectories.



(a) (b)

Figure 4-1: An illustration of the difference between the unaligned and aligned residuals. The red lines indicate the error between the points in the two trajectories. The point pairs get out of phase due to the different turn locations, resulting in the longer diagonal lines in (a). This results in an undesirably large distance between two otherwise similar trajectories, but can easily be corrected by using dynamic time warping to find more appropriate correspondences between the discrete points in the trajectories.

Such curve matching problems have seen much study in terms of time-series analysis.

Algorithm 2 The Dynamic Time Warping (DTW) algorithm.

Require: Trajectories $\tau^a = \{s_1^a, s_2^a, ...s_n^a\}$ and $\tau^b = \{s_1^b, s_2^b, ...s^{b_m}\}$

```
D ← ∞(n, m)
D(0, 0) = 0
for i ∈ {1, ..., n} do
    for j ∈ {1, ..., m} do
        d = ||s_i^a − s_j^b||
        D(i, j) = min[D(i − 1, j − 1),  D(i − 1, j),  D(i, j − 1)] + d
    end for
end for
return D(n, m)
```

One of the most popular algorithms for comparing time series is called Dynamic Time Warping(DTW) [96, 6]. The DTW algorithm seeks to warp the time-axis of the two trajectories in a non-linear manner so as to minimize the residual distance between corresponding points in the two curves, subject to ordering constraints. The correspondences output by DTW can then be used to compute a distance between two curves by summing (or taking the max) residual after alignment. It has previously been used to align robot trajectories by Coates et al. [22]. The dynamic time warping distance was used by Berenson et al. [5] to compare the similarity of planned trajectories.

DTW is a dynamic programming algorithm that minimizes the sum of the residual between points in the two trajectories. Pseudocode for the basic algorithm is shown in algorithm 2.

4.2 Affinity Propagation

With a suitable distance metric between trajectories, we can now turn our attention towards algorithms that will cluster the trajectories into meaningful groups. In this work, we use the Affinity Propagation (AP) algorithm [37] due to the confluence of nice properties for clustering trajectories. AP is computationally efficient, requires only a similarity metric between trajectories (the similarity metric need not be a valid distance metric), automatically determines the appropriate number of clusters based on a single scalar parameter, and is easy to implement. In addition, it selects one of the actual trajectories as an exemplar for each cluster, which is useful for visualizing the results, and could be useful for other appli-

cations of trajectory clustering. Affinity propagation is related to the common k-medoids algorithm [94], but does not require k to be specified explicitly.

Affinity propagation is a message passing algorithm that, given a set of similarities between pairs of data points, exchanges messages between nodes representing the data points so as to find a subset of "exemplar" points that best describe the data. AP associates each data point with one exemplar, resulting in a partitioning of the whole data set into clusters. The goal of AP is to maximize the overall sum of similarities between data points and their exemplars. Let

$$A(i,j) = \begin{cases} 1 - D(\tau^i, \tau^j), & \text{if } i \neq j \\ \gamma, & \text{if } i = j \end{cases} \tag{4.5}$$

be a matrix containing the similarity between trajectories when $i \neq j$, and the proclivity for trajectory i to be a cluster exemplar when $i = j$. The constant γ is a parameter of the algorithm. AP seeks to optimize the function

$$\begin{aligned} \underset{c}{\text{maximize}} \quad & \sum_{i,j} c_{ij} A(i,j) \\ \text{subject to} \quad & \sum_{i} c_{ij} = 1, \ j \in 1, \ldots, n \end{aligned} \tag{4.6}$$

where each c_{ij} is an indicator variable specifying whether trajectory i is mapped to cluster j. If there are n trajectories, such that A is an $n \times n$ matrix, then there are n^2 indicator variables.

The message passing algorithm employed by affinity propagation can be thought of as a version of the well known max-sum algorithm [69] on an appropriately constructed factor graph [40]. The factor graph model for AP is shown in Figure 4-2. The variables $c_{ij} \in \{0, 1\}$ are equal to 1 if and only if trajectory j is an exemplar, and trajectory i is mapped to cluster j.

There are three types of factors in the model: S_{ij}, I_i, and E_j. The factors S_{ij} represent

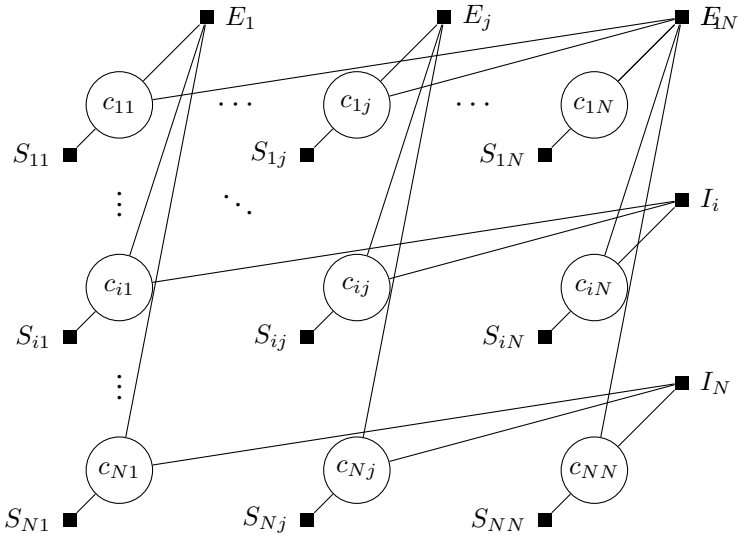


Figure 4-2: The factor graph model representation of the Affinity Propagation model of clustering

the data likelihood factors, and are defined as

$$S_{ij}(c_{ij}) = \begin{cases} A(i,j), & \text{if } c_{ij} = 1 \\ 0, & \text{otherwise} \end{cases} \tag{4.7}$$

The factors I_i enforce the constraint that a given trajectory may only be part of a single cluster.

$$I_i(c_{i1}, \ldots, c_{iN}) = \begin{cases} -\infty, & \text{if } \sum_j c_{ij} \neq 1 \\ 0, & \text{otherwise} \end{cases} \tag{4.8}$$

Finally, the factors E_j enforce that trajectory j may only choose trajectory j as its exemplar if trajectory j has chosen itself as an exemplar.

$$E_j(c_{1j}, \ldots, c_{Nj}) = \begin{cases} -\infty, & \text{if } c_{jj} = 0, \text{and } \sum_i c_{ij} \neq 0 \\ 0, & \text{otherwise} \end{cases} \tag{4.9}$$

Together, the factors and variables define the following likelihood function

$$S(c_{11}, \ldots, c_{NN}) = \sum_{i,j} S_{ij}(c_{ij}) + \sum_{i} I_i(c_{i1}, \ldots, c_{iN}) + \sum_{j} E_j(c_{1j}, \ldots, c_{Nj}) \qquad (4.10)$$

which is equivalent to maximizing the net similarity from equation 4.6.

The factor graph in Figure 4-2 has cycles in it, which means that exact inference, and therefore finding the global optima of the likelihood function is computationally intractable. However, an efficient variant of loopy belief propagation has been shown to work very well in practice. We refer the interested reader to Givoni and Frey [40] for details on the message passing scheme used to perform inference.

4.3 Clustering Algorithm Evaluation

We have evaluated the clustering algorithms on data generated by a sample based motion planner. The expected output of the clustering algorithms is intuitively that the trajectories are grouped into sets that approximate the major decision points in the environment.

To generate the trajectories, we ran a probabilistic roadmap planner with a Dubins vehicle model in several environments. However, rather than rely on the planner to output a single path between a start and goal location, we used the complete graph to generate sets of trajectories that are roughly the same length. We generated trajectories originating from a randomly chosen vehicle location and taking the vehicle a fixed distance away.

To generate the sets of trajectories out to a fixed horizon, we computed the optimal path from the start node to every node in the graph using Dijkstra's algorithm. We then tile the workspace with cells of size d, and subsample the nodes to select the node with the lowest cost to reach each cell. Finally, since we are interested in trajectories that reach our planning horizon h, we select the remaining nodes that have a path cost of between h, and $h + d$. Trajectories that are longer than h are then truncated to be exactly h long. For the sets of trajectories shown in figure 4-3, we used a planning horizon of 30 meters, and a subsampling cell size of one meter.

As can be seen in figure 4-3, clustering the trajectories using our algorithms splits the

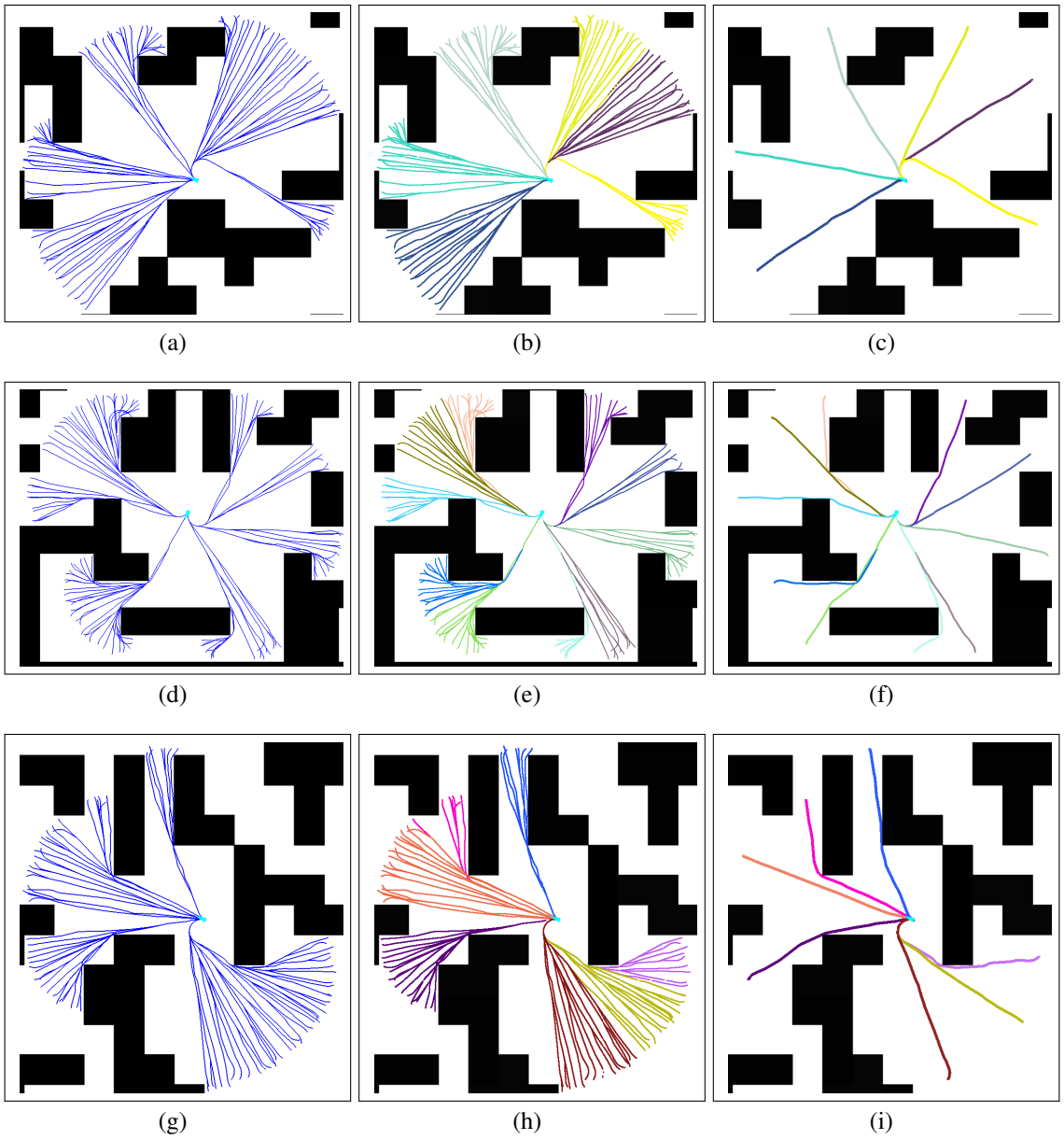


Figure 4-3: Three examples of the output of the similarity metric trajectory clustering. The first column shows the set of trajectories to be clustered. The second column colors these trajectories by the cluster membership. Each cluster is given a unique random color. Finally, the third column highlights the trajectories chosen as the cluster exemplar by the Affinity Propagation algorithm.

trajectories into sets that correspond to qualitatively different paths. While the clusters do not directly correspond to notions of homotopy, depending on the use case, the partitioning provided by the trajectory clustering seems to be closer to a human interpretation of the partitioning of the space. For example, while it would be reasonable to group all the trajectories that go straight past an obstacle together, separating out the trajectories that go past the obstacle, and then turn behind it probably makes sense.

4.4 Trajectory Bundle Library Generation

To generate the trajectory bundle library, we cluster the trajectories in our dense trajectory library from section 3.2.1. We wish to encourage the bundles to be tightly grouped near their start, and fan out further away from the vehicle. To accentuate this behavior, we weight the residuals in the similarity metric by $1/\sqrt{t}$, as discussed in equation 4.4.

The trajectory clustering algorithm clustered the 4707 trajectories into 121 trajectory bundles. The individual bundles are visualized in figure 4-4. As we can see, the trajectories are grouped into clusters that correspond to a sets of trajectory that all go in the same general direction. In addition, they are more tightly grouped at the start, and then fan out further from the origin. This should result in a representation that is more specific to the details of obstacles close to the vehicle, and has more invariance to small changes further afield.

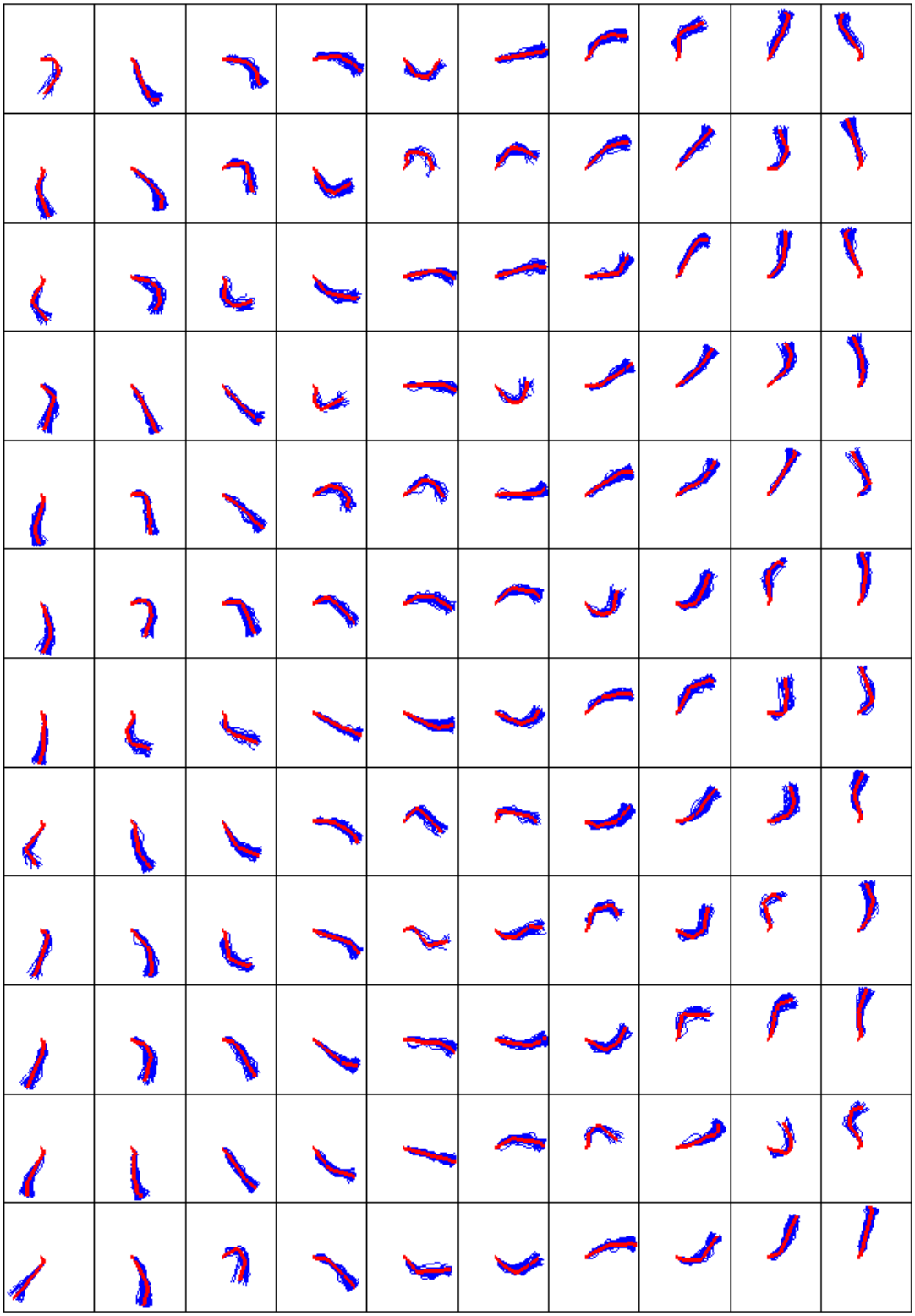


Figure 4-4: A visualization of the trajectory bundle library. Each trajectory bundle is drawn individually with the trajectories in the bundle drawn in blue, and the exemplar for each cluster drawn in red.

Chapter 5

Trajectory Bundle Prediction

In this chapter, we focus on methods for predicting the feasibility of trajectory bundles using the information measured by sensors carried on the vehicle. We employ machine learning methods to leverage past experience and interpret how the sensor measurements relate to the trajectory bundles. If we treat the feasibility of trajectory bundles as independent, then this problem reduces to training an independent binary classifier for each bundle.

The aim of a binary classification algorithm is the following: given a set of training examples consisting of tuples of features extracted from the raw sensor measurements Φ and a binary output variable $x \in \{0, 1\}$, learn a function

$$\Phi \to x \tag{5.1}$$

In our application, x is a binary variable indicating whether bundle β is feasible. We will train an independent binary classifier for each bundle in the trajectory bundle library.

Binary classification is one of the most well studied problem in machine learning, and there has been tremendous progress in developing accurate, fast, and efficient algorithms [16, 98]. One of the advantages of framing our planning problem in terms of trajectory bundles is that it allows us to leverage these algorithms to learn predictive models. In contrast, if one seeks to train models to predict the environmental geometry as a set of polyhedra, more complicated structured prediction algorithms [89] are needed to model the

complex dependencies between variables. For occupancy maps, one could train a binary classifier to predict whether each cell in is occupied, however, the training data for such a low level concept would be extremely noisy, hampering generalization to new environments. In section 5.1.2 we will compare the utility of predicting trajectory bundles against directly predicting an occupancy map.

5.1 Nearest Neighbor Prediction

While any classification method such as support vector machines [16] or boosting [98] could be applied, in this work, we choose to employ a k-nearest neighbor classifier [23]. Nearest neighbor classifiers leverage the intuition that scenes that have similar visual features should have similar sets of feasible trajectory bundles. Nearest neighbor techniques are convenient because with a single retrieval, we can compute the predicted output for all bundles rather than having to train a separate classifier for each bundle. Finally, nearest neighbor prediction methods allow us to compare the prediction performance when we use the same features and distance function for retrieval, but predict different output labels. For example we can compare the performance when we predict the occupancy of a grid map, as compared to predicting the feasibility of trajectory bundles. Normally, this would require training different classifiers, and it would be more difficult to attribute the differences in the prediction performance.

We will used the trained classifier to predict whether each trajectory bundle is feasible based on features of the environment observed by its sensors. With an unknown map, this is a very challenging problem, however, for areas where we have access to a map (either beforehand, or computed after operating the vehicle), we can use the map to label training data automatically. This situation is in contrast to many situations in machine learning, and especially in computer vision where it is very expensive to acquire labeled training data since each training sample must generally be manually annotated by a person.

Nearest neighbor prediction methods are learning algorithms that directly use the training data to predict new test samples. The "training" phase consists simply of aggregating a large set of tuples, each containing the input feature values, and the output decision values

for each tuple.

$$(\Phi, X) \tag{5.2}$$

The variable Φ is a vector of features extracted from the sensor data observed at that sample point, and $X = [x^1, \ldots, x^n]$ are the feasibility labels for each of the trajectory bundles in the library.

At test time, we find the K samples (or nearest neighbors) whose feature distance

$$||\Phi_t - \Phi_i|| \tag{5.3}$$

to the test point Φ_t is minimal. The output predictor variable for all tuples $\hat{X}_t = [\hat{x}_t^1, \ldots, \hat{x}_t^n]$ is then computed by determining whether the bitwise average of the predictor variables in the K nearest neighbors is greater than the decision threshold η.

$$\hat{X}_t = \frac{1}{K} \sum_{i=1}^{K} X_i > \eta \tag{5.4}$$

The nearest neighbor retrieval is performed once, but the output prediction is computed independently for each trajectory bundle. The bitvector representing the feasibility of each bundle in the library is averaged on a per-bundle basis.

Features

In general, one does not directly use the raw sensor data to query for similar samples. The measurements are first transformed by computing a set of features Φ from the data. In addition, one must choose a distance function to evaluate the similarity between the extracted features.

In this work, we focus on making navigation decisions based on the information contained in camera data. While we focus on visual imagery, once the form of the raw data is abstracted by feature extraction, our method is not specific to cameras. Any other features of the data, or features computed from other sensors could be used just as easily.

There has been a considerable amount of work in the computer vision community on

developing effective visual image features. For our purposes, we seek to use the idea that scenes with similar visual appearance will also have similar geometry and therefore similar feasible trajectories. As such, we care more about matching the gross, large scale image features than local details. For this reason, we employ the GIST global image feature [90]. The feature was developed for the purpose of evaluating holistic scene structure with the goal of categorizing the type of environment.

The GIST feature aggregates the responses of a battery of Gabor filters over the candidate image. It splits the image into a set of horizontal and vertical regions, and then computes the energy of Gabor filters at different orientations and scales. It is closely related to the popular histogram of oriented gradients (HOG) descriptors that are often used in object detection [25]. The output energy from each of the filters is aggregated into a single feature vector that is used to classify the samples.

We use the chi-squared distance between the GIST features as described in Xiao et al. [121].

$$||\Phi(I_t) - \Phi(I_i)|| = \frac{(\Phi(I_t) - \Phi(I_i))^2}{\Phi(I_t) + \Phi(I_i)} \tag{5.5}$$

Figure 5-1 shows some examples of the retrievals based on the GIST features in our Tetris World simulation environment. The retrieved images in general have similar gross scene structure, however they differ in terms of the fine details. The invariance to minor changes in the environment provided by the use of trajectory bundles allows the classifier to still capture relevant information despite the slightly differing scenes.

As we shall see, this combination of image features and distance function provided reasonable performance in the simulated Tetris-World environment. For a deployed system, further evaluation of the myriad of image features that have been proposed in the literature would likely be in order [121, 62]. The distance metric used for query retrieval could also be optimized using distance metric learning [122]. Alternatively, one could use a completely different learning method such as support vector machines (SVMs) [16] or boosting classifiers [98]. This flexibility points to another benefit in using the trajectory bundle representation. Since the library contains a relatively small number of bundles, one could simply train an independent binary classifier for each one. Customizing the features

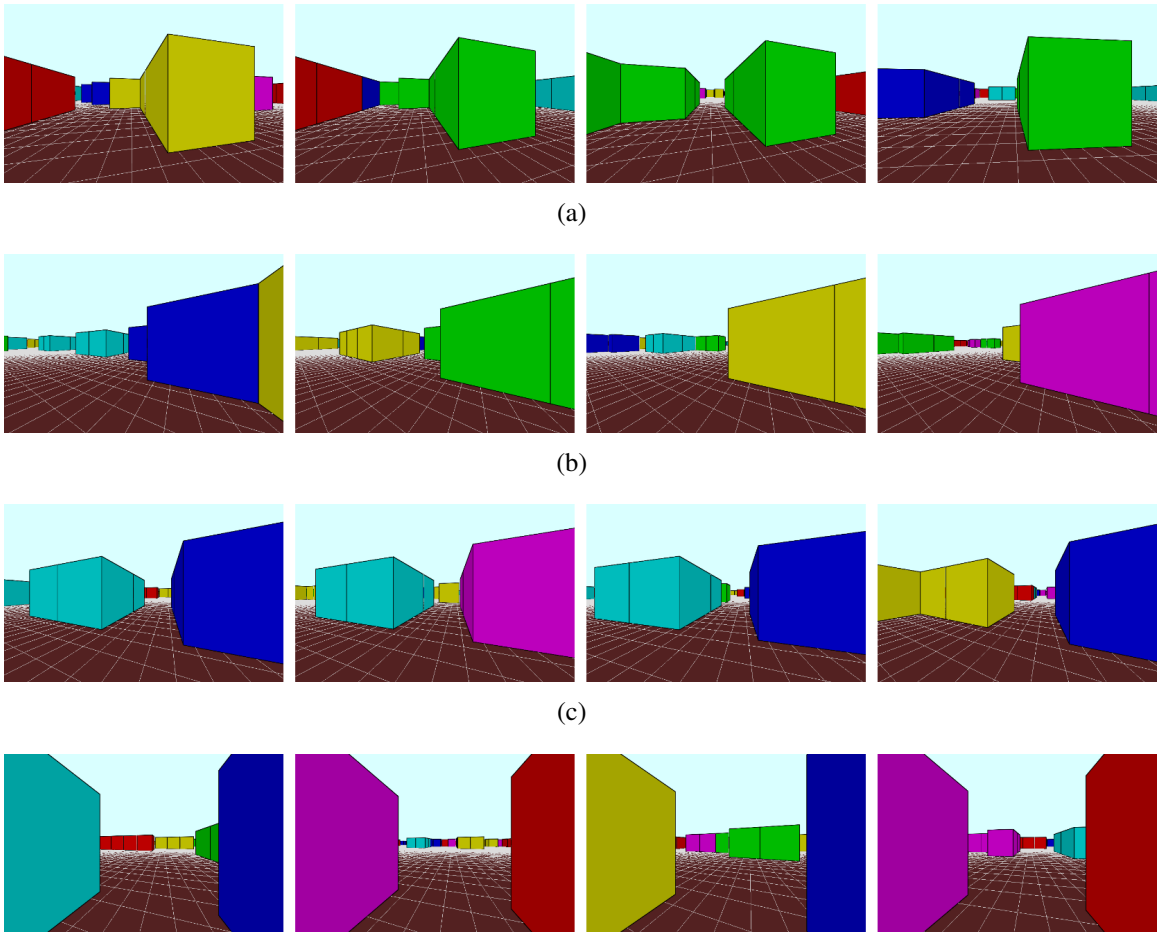


Figure 5-1: Examples of the images retrieved by the GIST feature distance. The left column shows the query image. The images to the right show the three nearest neighbors in the database.

used for each bundle could probably further improve performance.

5.1.1 Classification Performance

To evaluate the accuracy of the nearest neighbor classifier prediction, we performed an experiment in our Tetris-World simulation environment. We created a database containing 140K tuples from 7 different environments, and then created 20K further tuples in a different environment for testing.

Classifiers often output a continuous value that gets thresholded to determine the output class. In the case of our nearest neighbor classifier, the output value is computed as the

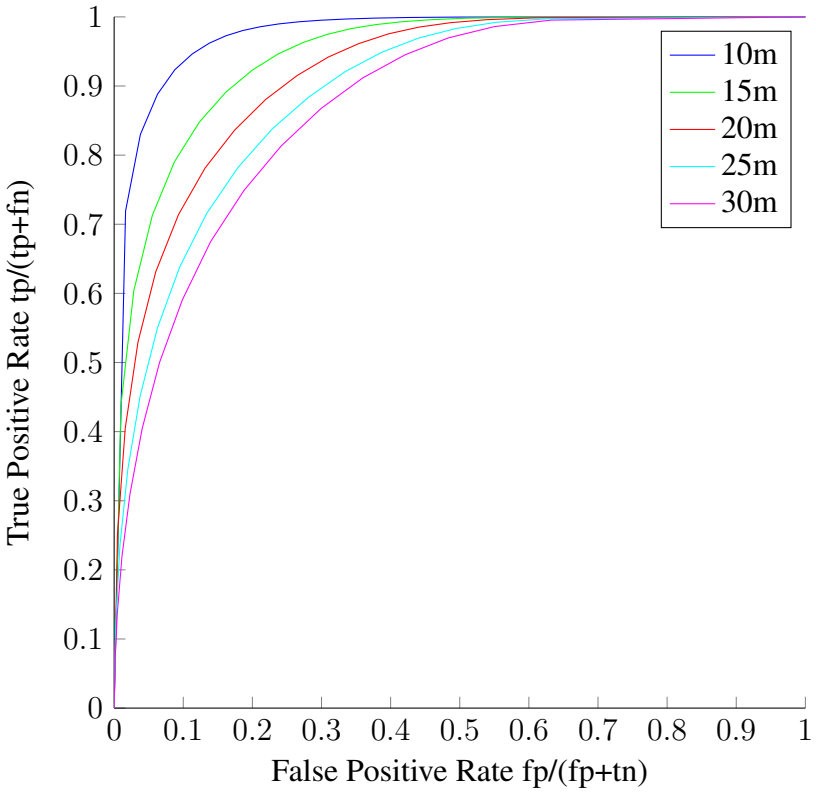


Figure 5-2: A Comparison of the prediction performance in terms of the ROC curves for nearest neighbor classifier using GIST features to make predictions at different ranges. At a given range, we set the labels on the trajectory bundles to only consider collisions within that planning horizon. The plot indicates that our predictions are more accurate closer to the vehicle.

average of the labels of all of 16 retrieved examples. The fraction of samples that are feasible provides a measure of the confidence in the prediction. This means that the output will be 0 if none of the retrieved tuples are feasible. In contrast, it will be 1 if all of the retrieved tuples are feasible. To predict whether the test example is feasible or not, we apply a threshold η to the computed average. By increasing η we require the classifier to be more confidant before we will say that a bundle is feasible.

The receiver operator characteristic (ROC) curve is a common way of evaluating the performance of a binary classifier. The curve plots the true positive rate

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5.6}$$

versus the false positive rate.

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{5.7}$$

The ROC curve plots the TPR versus the FPR as the decision threshold is swept between 0 and 1. An ideal classifier would make no errors, corresponding to a point in the top left of the plot. In contrast, a random (or uninformative classifier) would result in an ROC curve that is a diagonal line between (0,0) and (1,1).

Figure 5-2 shows the ROC curve of the output of our NN classifier with GIST features. We predict the feasibility of the trajectory bundles at varying ranges. As one might expect, the image provides less information about scene structure that is far away from the camera. As a result, the prediction accuracy decreases with range. Despite the decline, the prediction performance at 30m range is still significantly above the diagonal line of an uninformative classifier.

The ROC curves in figure 5-2 show the average prediction performance over all trajectory bundles in the library. However, one should expect that some trajectory bundles are more difficult to predict than others. For example, there are trajectory bundles that move outside of the field of view of the camera almost immediately. Predicting the feasibility of these bundles requires more reliance on context than directly observed scene structure. Figure 5-3 plots the area under the ROC curve broken down per trajectory bundle. As one might expect, the trajectories at the extremes of the trajectory bundle library have the lowest area under the ROC curve.

5.1.2 Occupancy Map Prediction

As an interesting comparison, another way that one could try to use a nearest neighbor classifier for planning would be to directly predict an occupancy map of the area in front of the vehicle. To do this, instead of labeling each tuple with the feasible trajectory bundles, one would extract a vehicle centered occupancy map from the global map used to generate the training data. At query time, the retrieved maps can be averaged on a cell by cell basis. This results in a "blurry" predicted map, such as the one shown in Figure 5-4.

To evaluate these predictions in the same manner as the trajectory bundles, we threshold

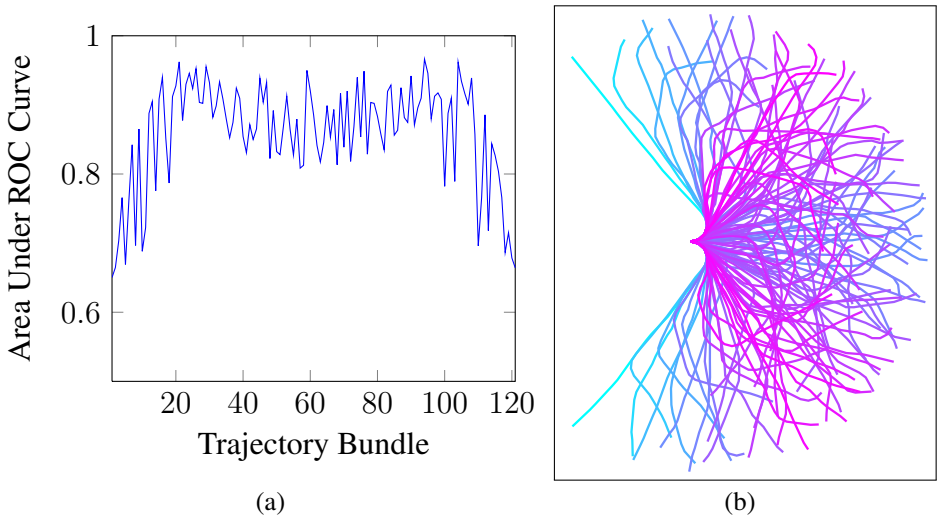
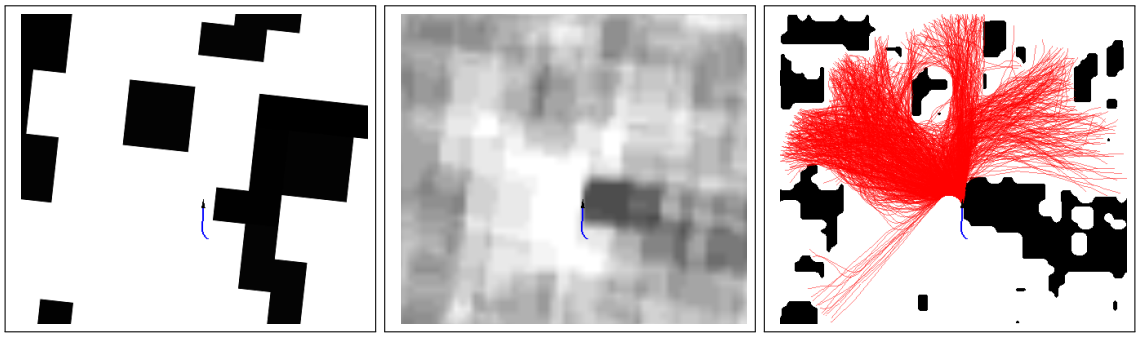


Figure 5-3: An analysis of the classification accuracy on a per trajectory bundle basis. The area under the ROC curve for each individual trajectory bundle is plotted in (a). Figure (b) draws the trajectory bundle exemplars, colored by the area under the ROC curve. Magenta trajectories are classified most accurately, cyan trajectories are predicted the least accurately.

the map, and then perform collision checking to evaluate the feasibility of each bundle. The ROC curve is then computed by sweeping the collision occupancy threshold between 0 and 1.

As one can see in the ROC curves in Figure 5-5, despite using identical image features, making the predictions by averaging the trajectory bundles bit-vector outperforms the occupancy map method. We believe that this is due to the fact that the averaging of trajectories does a better job of smoothing over the noise in the nearest neighbor retrievals than the cell-wise averaging.

In addition to the improvements due to the more complementary averaging of the bit-vectors, the compactness of the trajectory bundle bitvector makes it more amenable to learning and modeling the higher level environmental structure. As we will show in later sections, the bundle library representation allows us to learn models of the spatial and temporal regularities imposed by the environmental structure.



(a) (b) (c)

Figure 5-4: An illustration of the nearest neighbor prediction of an occupancy map. (a) shows the true environment where the vehicle (tiny blue triangle) is located. (b) shows the average of the retrieved submaps. (c) shows a thresholded version of (b) along with the trajectories that are collision free in the thresholded map.

5.2 Spatial Conditional Random Field Model

In the previous section, we developed prediction methods that treat each trajectory bundle independently. Due to the nature of the nearest neighbor classifier, we implicitly obtain a joint prediction over all bundles in the library, but the feasibility of each bundle is modeled independently.

This independent assumption is not always appropriate, in fact, there is a significant amount of correlation between the feasibility of trajectory bundles in our library. This correlation stems from the fact that the bundle feasibility depends on the underlying structure of the environment. Many environments (including our Tetris-World environment) are composed of obstacles and regions that are larger than the "spread" of the trajectories assigned to the same bundle. This means that adjacent trajectory bundles are more likely to have the same value than differing values.

Indeed, if we think about the feasibility of individual trajectory bundles as random variables, we can quantify this notion by looking at the mutual information between the random variables. The mutual information between two random variables X and Y can be computed as:

$$I(X,Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) log \left(\frac{p(x,y)}{p(x)p(y)} \right) \tag{5.8}$$

By analyzing the mutual information between trajectory bundles in the training data,

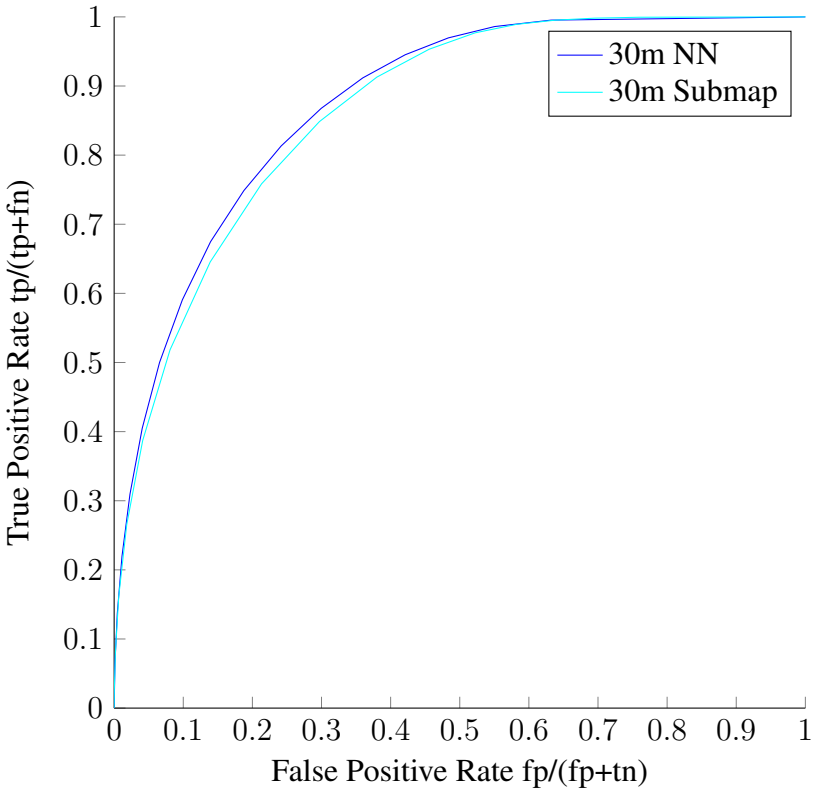
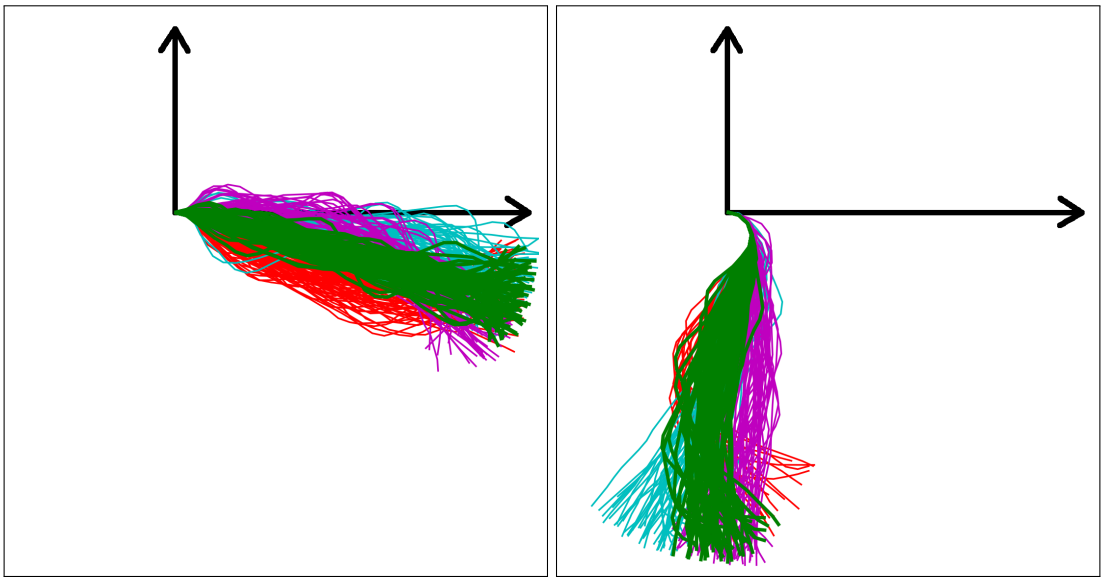


Figure 5-5: A comparison of the prediction performance in terms of the ROC curves for a nearest neighbor classifier predicting trajectory bundles directly versus predicting an occupancy map.

we can see that there are indeed significant correlations between trajectory bundles. In addition, the mutual information provides a useful metric to indicate which trajectory bundles are most similar to each other. Figure 5-6 shows two trajectory bundles along with the 3 other bundles with the highest mutual information.

If we take into account the structure inherent in the output variable of our predictions, we will likely be able to improve the accuracy of the predictions. Structured prediction problems [89] have seen a significant amount of research interest in recent years, and a number of algorithms and techniques for training prediction models have been developed. Of particular note are conditional random field models [73], and algorithms such as the structured support vector machine [116]. Despite advances, they are still unable to scale to large, high-dimensional datasets such as our trajectory bundle prediction problem. As a result, if we wish to apply such algorithms directly to the image data, we would need



(a) (b)

Figure 5-6: An illustration of trajectory bundles and the neighbors with maximal mutual information in the training data from section5.1.1. The red, cyan, and magenta trajectories make up the three bundles with greatest mutual information with the green bundle.

to considerably reduce the number of training samples compared to the nearest neighbor classifier described above, thereby limiting the prediction performance.

Instead, we employ a different approach, where we learn a conditional random field model to capture correlations in the output of the nearest neighbor classifier. The model seeks to estimate the distribution

$$p(X|\hat{X}) \tag{5.9}$$

where X represents the true feasibility of the trajectory bundles, and $\hat{X}$ are the predictions made by the nearest neighbor classifier. This approach is similar in spirit to the one employed in Liu et al. [77] which used a Markov random field model to combine the output of a nearest neighbor prediction of scene labels.

In this approach, we formulate the problem as an inference problem where we are attempting to infer the feasibility of each trajectory bundle, using the output of the nearest neighbor classifier as noisy observations. We take the formalism of factor graphs [69, 9], to develop our models. Factor graph models provide a convenient way to describe probabilistic models of complicated systems. Factor graphs are a class of bipartite graphs

with two types of vertices. Nodes correspond to random variables, which in our case are binary random variables representing whether a given trajectory bundle is feasible or not. Factor vertices describe the local interactions between the variables that are connected to the factor. A key question that remains is to identify a model structure that is captures as much of the true dependencies as possible, while remaining computationally tractable to perform inference over. This problem is called the graphical model structure learning problem [39, 102].

We consider two different model structures. The first model is an independent-node, shared-feature model. In this model, the nodes are treated as independent, however each node depends on the outputs of the observations of nodes that have high mutual information with it. The second model we look at is a Chow-Liu tree model [21]. Factor graphs for the two models are shown in Figures 5-7 and 5-8.

5.2.1 Independent-Node Shared-Feature Model

For the independent-node model, we assume that all nodes are independent of one another, but share features. In this model, we connect each node to the features (observations) from the K nodes that have the highest mutual information. A factor graph representation of this model with six nodes, and three shared features per node is shown in Figure 5-7. In this model, we assume that

$$p(X|\hat{X}) = \prod_{x^i \in X} p(x_t^i|\hat{X}_t) \tag{5.10}$$

Because there are no explicit dependencies between the hidden variables (nodes), inference and parameter learning in this model is very efficient. In addition, the learning problem can be separated, with each node computed independently.

On the other hand, this model does not explicitly enforce consistency between the output variables. For example, the statistics of the nodes are contractive, which means that they are more likely to have the same value than to differ. While we would expect the output of nodes that take into account many of the same features to produce consistent values among them, nothing in the model explicitly enforces this.

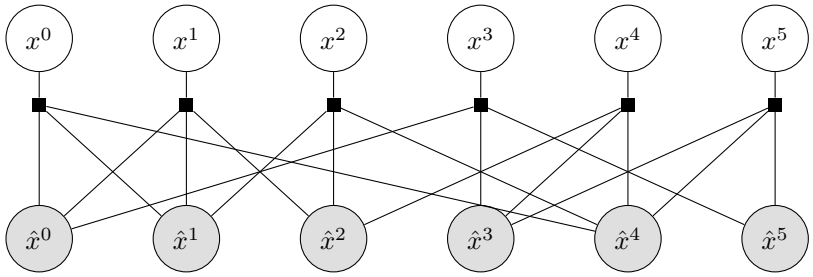


Figure 5-7: The factor graph representation of the independent-node shared-feature model with 6 bundles, and 2 neighbors. The black boxes represent factors relating the nodes, which are drawn as circles. The shading of the nodes indicate whether the variable is observed, or must be inferred.

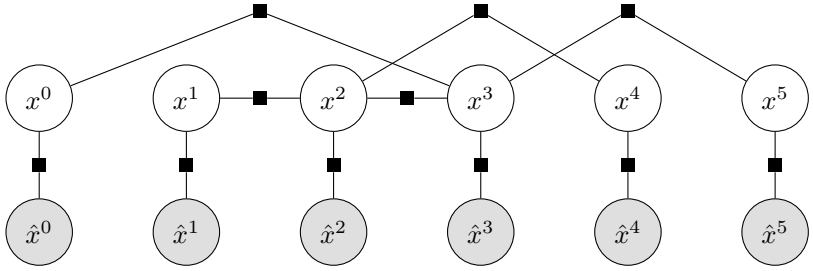


Figure 5-8: An example of the factor graph representation of the Chow-Liu tree model for 6 bundles.

5.2.2 Chow-Liu Tree Model

The second model that we investigate is a Chow-Liu tree model. The Chow-Liu method seeks to approximate a joint probability distribution as the product of pairwise potentials. The Chow-Liu algorithm provides a method for choosing which pairwise dependencies to use in order to obtain the closest approximation to the original joint distribution in terms of the Kullback-Leibler divergence. The algorithm proceeds in two steps. First, the mutual information between all nodes is computed. The next step is to compute the maximum spanning tree, using the mutual information between nodes as the edge weights. Any efficient spanning tree algorithm such as Kruzkal's algorithm, or Prim's algorithm may be used. An example factor graph for a Chow-Liu tree model with six nodes is shown in figure 5-8.

5.2.3 CRF Prediction Performance

With the structure chosen using the two methods described above, we learn the maximum likelihood model parameters using Schmidt's undirected graphical model toolbox [101].

We trained both models on a subset of 10000 training examples from the world described in 3. After learning the parameters for both models, we compared the prediction performance on a held out test set consisting of 10000 samples. Figure 5-9 shows the ROC curves for the two models alongside the prediction performance of just using the classifier outputs (observations) directly.

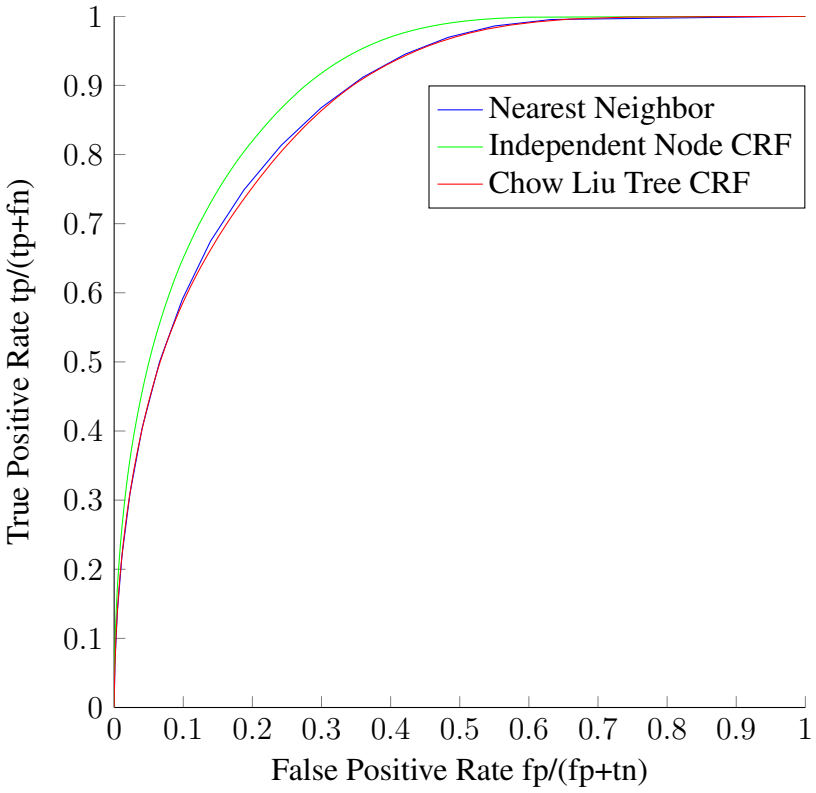


Figure 5-9: The ROC curves for the different CRF models, alongside the NN-classifier baseline. The Chow-Liu tree model does not improve over the NN-classifier, but the independent-node shared feature model significantly outperforms them both.

These results show that Chow-Liu model does not seem to provide any benefit over using the nearest neighbor classifier directly. It appears that a tree model is not expressive enough to capture the dependencies between the nodes. In contrast, the independent-node model is able to leverage information contained in the predictions from many similar nodes

to improve on the prediction performance for all of them. Investigation of further models, with higher order structure may improve performance further, however this is left for future work.

5.3 Planning Performance

We have shown that our nearest neighbor based classifier is able to predict which trajectory bundles are feasible with reasonable accuracy. We now turn to using these predictions to make planning decisions. We will show that the predictions are useful for making navigation decisions that enable the vehicle to reach the goal without colliding with obstacles much more frequently than when using a short range metric sensor alone. In addition, we will show that the trajectory bundle feasibility predictions result in superior planning performance compared to when the classifier predicts either the trajectory feasibility directly, or an intermediate occupancy map.

For these tests, we set up the same type of simulation trials as discussed in section 3.2, however, instead of augmenting the vehicle with a more powerful range sensor, we will use our predictions based on cues from the monocular camera images.

A convenient aspect of the trajectory bundle representation is that it enables very simple decision making. To identify which bundles are feasible, it is sufficient to simply threshold the prediction confidence, and then treat the bundles that are above that threshold as feasible. We use a threshold of 0.5 in the results that follow.

Based on our predictions of which trajectory bundles are feasible, we can project that information back down to the underlying trajectory library by simply labeling all of the component trajectories inside each bundle as feasible if the bundle is predicted to be feasible.

We then use the occupancy map built using the short range laser sensor to prune away any false-positive trajectories that were predicted to be feasible, but are known to be in collision. This provides some sanity checking, and short range obstacle avoidance, but does not influence the long range predictions. The final set of predicted collision-free trajectories are then used for planning using the same decision making rules as described

in section 3.2. The heuristic cost to go from the end of each of the feasible trajectory is evaluated, and the vehicle chooses to follow the feasible trajectory with the minimum cost to go. The backtracking trajectory library is invoked if none of the trajectory bundles are predicted to be feasible.

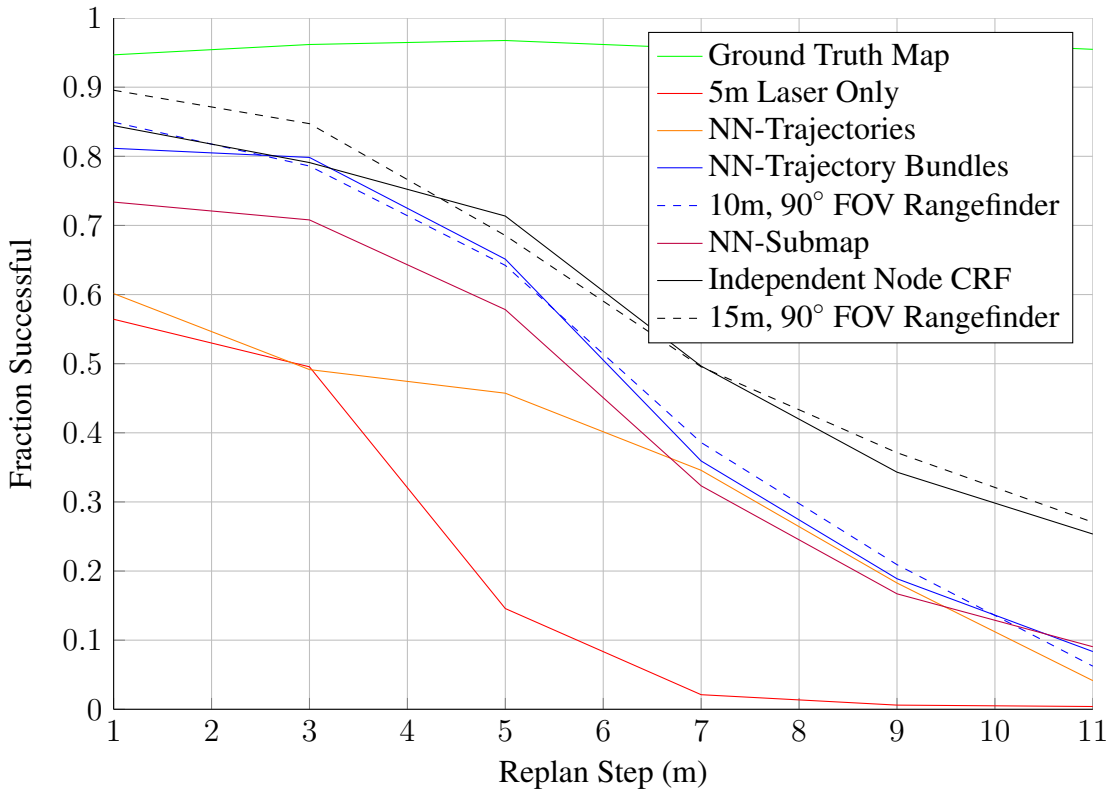


Figure 5-10: A comparison of the success rate in reaching the goal with different prediction methods. The nearest neighbor prediction with trajectory bundles outperforms the nearest neighbor prediction of occupancy maps, and the trajectories directly. The plain neighbor prediction results in performance similar to a 10m rangefinder. The independent-node CRF improves the success rate to be similar to the 15m rangefinder.

Figure 5-10 demonstrates the planning performance of the vehicle using a number of different prediction methods. As we can see, using the nearest neighbor based predictions from the monocular camera imagery significantly outperforms the short range laser-only case. Furthermore, we can see that there is a significant improvement that results from predicting the feasibility of trajectory bundles.

Directly predicting the feasibility of individual trajectories performs almost as badly as using the short range laser sensor alone. We believe that the poor performance is due to the

damaging effect of random false positives. If a random trajectory is predicted as feasible when it is not, and that trajectory has a small heuristic cost-to-go, the vehicle will follow that trajectory even if none of the other similar trajectories are deemed to be feasible. As a result, even if a classifier always predicts all but one trajectory correctly, if that trajectory is a false positive with a low cost-to-go, the vehicle will likely collide with an obstacle. Since a trajectory bundle represents a more abstract concept about a whole group of trajectories, the classifier output is less likely to contain such damaging random false positives in its output.

Making predictions based on the trajectory bundles also consistently outperforms the predictions made by averaging the occupancy maps. The trajectory bundles capture the structure of the problem better than the flat occupancy map, so averaging the feasibility of the retrieved nearest neighbors provides more useful information for decision making.

All in all, the performance of the monocular image-based nearest neighbor prediction methods gave the vehicle a similar success rate to when it had access to a 10m, 90° field of view range sensor.

When we employed the CRF model, the success rate improved further. At shorter replan intervals the CRF and the NN predictions perform pretty similarly. However, for replan intervals greater than 5m it outperforms. The spatial CRF model increases the navigation performance to be similar to the situation when the vehicle has access to a 15m range camera.

Chapter 6

Bayesian Filtering In The Space of Trajectory Bundles

At each time step, the vehicle will take an action, and receive a measurement from its sensors. This sequence of observations should continually improve the vehicle's understanding of the environment around it. In the previous chapter, we introduced prediction methods for predicting which trajectory bundles are feasible based on the sensor measurements observed by the vehicle's sensors at a given time step. However, rather than make an independent prediction based on the latest measurement, we can improve the prediction accuracy by taking into account the entire history of measurements.

Framed in this manner, we can think of the trajectory bundle prediction problem as a state-estimation task where the state represents the feasibility of each bundle in the library. Since we cannot be certain whether a given bundle is feasible based on noisy and ambiguous sensor data, we instead estimate a belief over the feasibility of each trajectory bundle. In state-estimation, Bayesian filtering has become the method of choice for estimating the unknown state of the vehicle from a series of noisy measurements. In this chapter we develop models that allows us to apply Bayesian filtering techniques in the space of trajectory bundles.

Bayesian filtering algorithms seek to recursively estimate a belief over the state of the system at time t, denoted X_t, from the history of actions taken $a_{1:t}$, and the history of

observations $Z_{1:t}$.

$$bel(X_t) = p(X_t|Z_{1:t}, a_{1:t}) \tag{6.1}$$

For our problem, the state X_t is a bitvector representing the feasibility of each of the trajectory bundles. We will develop our models in terms of generic observations Z, however in practice, rather than use the raw sensor data as the input to our model, we will use the output of the prediction models described in chapter 5. This means that we will actually be computing the belief over X conditioned on the history of predictions $\hat{X}_{1:t}$.

$$bel(X_t) = p(X_t|\hat{X}_{1:t}, a_{1:t}) \tag{6.2}$$

6.1 Hidden Markov Models

One of the most commonly used models for temporal inference problems is the hidden Markov model (HMM). In an HMM, it is assumed that the current state of the system is independent of its history given the state at the previous time step.

$$p(X_t|X_1, \ldots, X_{t-1}) = p(X_t|X_{t-1}) \tag{6.3}$$

Furthermore, it is assumed that the observations depend only on the state of the system at the time when the measurement was taken.

$$p(Z_t|X_1, \ldots, X_t) = p(Z_t|X_t) \tag{6.4}$$

The factor graph for an HMM is shown in Figure 6-1.

HMMs are attractive for their simplicity, and because efficient algorithms for performing inference and parameter learning in HMMs are well established [113]. For example, if the state of the system can be represented as a (multi-variate) Gaussian, and the system dynamics are linear, then Bayesian filtering can be solved using the well known Kalman Filter [61]. The Kalman filter is a specific instance of a Bayesian filtering algorithm applied to a hidden Markov model system with a linear models and a Gaussian belief.

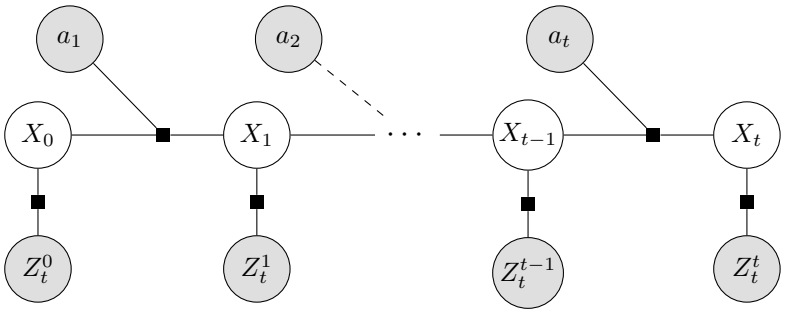


Figure 6-1: The factor graph representation of a Hidden Markov Model.

For discrete models, the belief over X is tracked as a multinomial distribution. The discrete Bayes filtering algorithm computes $bel(X_t)$ recursively, basing the current estimate of the belief on the previous belief $bel(X_{t-1})$. The algorithm splits the computation into two steps: the control update step, which computes

$$\overline{bel}(X_t) = \sum_{X_{t-1}} p(X_t|X_{t-1}, a_t) bel(X_{t-1}) \tag{6.5}$$

and the measurement update step

$$bel(X_t) = \eta p(Z_t|X_t)\overline{bel}(X_t) \tag{6.6}$$

where η is a normalizing constant.

Unfortunately our trajectory bundle library representation does not lend itself to a single discrete state representation as required by an HMM model. If we have n bundles, then the state of the system can be thought of as an n-dimensional bitvector. That means that the single discrete state must take on 2^n values. For extremely simple systems or short planning horizons, one might be able to use a small enough number of bundles that this would be feasible, however for the system described in chapter 3 we use 121 bundles. By considering bundles instead of the underlying trajectories we reduce the number of states in the system considerably, however, using a single discrete state representation is still clearly computationally infeasible.

Fortunately, we can leverage the inherent structure among the trajectory bundles to factor the model into more manageable pieces. Such factored temporal models are often

called dynamic Bayes nets (DBNs) [86].

6.2 Dynamic Bayes Network Models

The feasibility of different clusters are correlated with each other both in space and across time due to their dependence on the underlying structure of the environment. If we knew the true underlying state of the environment, then, conditioned on this map, the bundle feasibilities at all time steps would be conditionally independent of each other. However, this environmental structure is exactly the information that we are trying to estimate. While the trajectories will be correlated, the extent of this correlation will depend on the spatial layout. For example, clusters that overlap or cross over each other are more likely to be correlated than ones that go in opposite directions.

It is important to note however, that these correlations extend beyond simple geometric overlap due to the non-uniform statistics of obstacles in the environment. Two clusters that are very close to one another but do not overlap will still be correlated due to the possibility that they are both colliding with the same obstacle in the environment. That being said, the strength of these correlations will likely vary significantly, and be depend on the underlying spatial overlap of the trajectory clusters.

The relative strengths of the correlations means that in practice we can treat many of the clusters as independent. These independence assumptions between trajectories are what separates the DBN from a HMM model. By factoring the probability distribution further using these conditional independence assumptions the computational complexity is significantly reduced.

Unfortunately, as can be seen in Figure 6-2, the resulting graph still has a number of cycles. This means that exact inference will be computationally intractable. While approximate inference methods such as loopy belief propagation [87] have seen considerable success for many problems, it is usually assumed that the model parameters (potential functions) are either known, or can be tuned by hand.

We seek to discover an effective temporal model for the dependencies between clusters. This means that we must learn both the structure of the model, as well as its parameters.

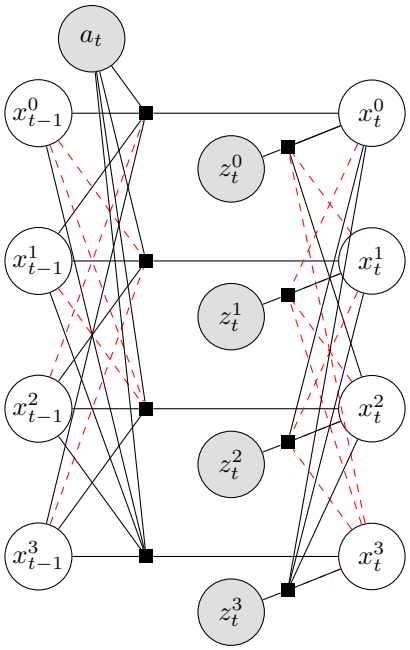


Figure 6-2: The factor graph representation of a Simplified Dynamic Bayes Net Model with 4 nodes. The missing edges are drawn by dashed red lines.

We first focus on the challenge of choosing an appropriate structure.

6.3 Bayes Filter Model

For simplicity, we investigate a model where we assume that each node will depend on the action taken, and a small number of parent nodes. Let $x^i \in X$ be a random variable indicating whether trajectory bundle β^i is feasible. For each bundle, we model the temporal dynamics of the state as

$$p(x^i_t | X_{t-1}, a_{t-1}) = p(x^i_t | x^{p1}_{t-1}, \ldots, x^{pk}_{t-1}, a_{t-1}) \tag{6.7}$$

where $x^{pi}_{t-1} \in X_{t-1}$ represents the i^{th} parent. The factor graph representation for such a model with two parents is shown in figure 6-3. Even with this simplification, we must still decide which parents each node will depend on, and the parameters of the factor.

As in the case of the spatial model, the geometric overlap between clusters (when transformed by the action), will result in different strengths of dependencies between clusters

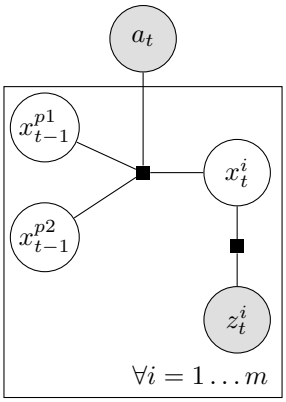


Figure 6-3: The factor graph representation of our Bayesian filter model. The factor graph is drawn for a model where each node has two parents. We use plate notation to designate the repeated structure of each node associated with two parents.

across time. Rather than use a heuristic measure based on geometry, we learned the best structure from data. We explored using between one and three parent nodes.

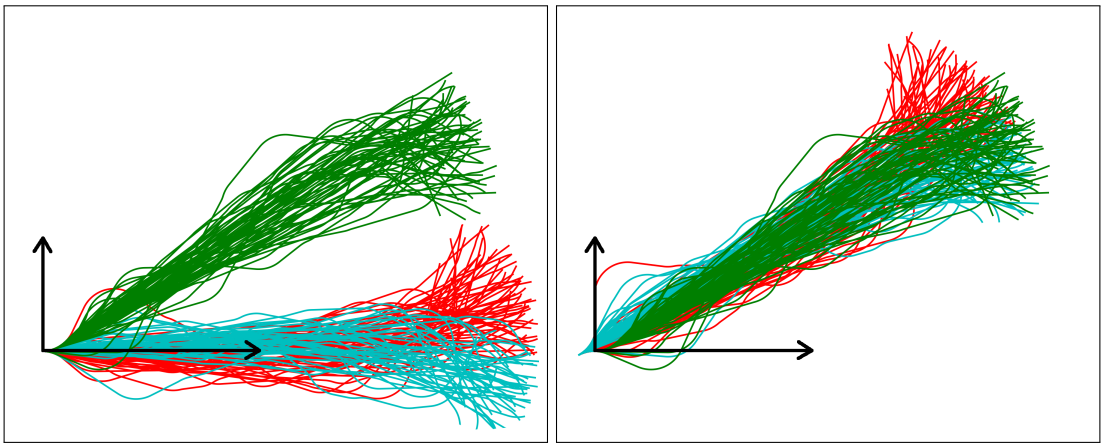
We discretized the continuous action space of the vehicle, and then for each node we chose the set of parents independently for each action. To determine which parents would be the most informative, for each bundle, we computed the conditional distribution for all sets of possible parents. Parent sets that are very informative should produce a conditional probability table with very low entropy, while uninformative parents will have high entropy.

We select the parents that minimize the conditional entropy in the distribution over the transition dynamics. The minimum conditional entropy of the empirical distribution of $p(x_t^i|x_{t-1}^{p1}, \ldots, x_{t-1}^{pk})$ computed for two parents is:

$$\underset{p1,p2}{\operatorname{argmin}} H(x_t^i|x_{t-1}^{p1}, x_{t-1}^{p2}) = \sum_{x_t^i, x_{t-1}^{p1}, x_{t-1}^{p2}} p(x_t^i, x_{t-1}^{p1}, x_{t-1}^{p2}) log\left(\frac{p(x_t^i)}{p(x_t^i, x_{t-1}^{p1}, x_{t-1}^{p2})}\right) \quad (6.8)$$

As shown in figure 6-4, the trajectories in the selected parent bundles have significant spatial overlap, once the transformed by the action.

The empirical conditional distribution is used as the potential for the temporal factor in the model. This approach to learning the model structure is related to the minimum description length approach presented by Friedman and Goldszmidt [38].



(a) (b)

Figure 6-4: An illustration of trajectory bundles and their parents with minimum conditional entropy for a sharp left action. The considered trajectory bundle is drawn in green, and the pair of parents are drawn in red and cyan. In (a) the bundles are shown originating from the same origin. In (b), we transform the parent bundles to originate from the initial pose before the vehicle took the action. Once transformed by the motion, the trajectories in the bundles have more overlap.

6.4 Inference

With the models described as above, we can compute the posterior feasibility estimate, combining the spatial and temporal models. Since each node depends only on its observation, and the distribution over the feasibility of its parent nodes at the previous time step, we able able to compute the posterior probability independently for each node. In addition, since we condition on the belief at the previous time step, the posterior probability can be computed using the law of total probability.

For each bundle β^i we compute the prediction using the independent node spatial CRF model discussed in the chapter 5. The output of the model is a prediction of the feasibility of each trajectory conditioned on the underlying features computed from the sensor data $p(x_t^i|\hat{X}_t)$.

Based on the motion executed, we can lookup the parents $p1$ and $p2$ of node i, and the associated conditional probability table $p(x_t^i|x_{t-1}^{p1}, x_{t-1}^{p2})$. Having kept track of our belief for the parents from the previous time step $bel(x_{t-1}^{p1})$ and $bel(x_{t-1}^{p2})$, We multiply these probabilities, along with the observation probability to obtain our posterior estimate of the

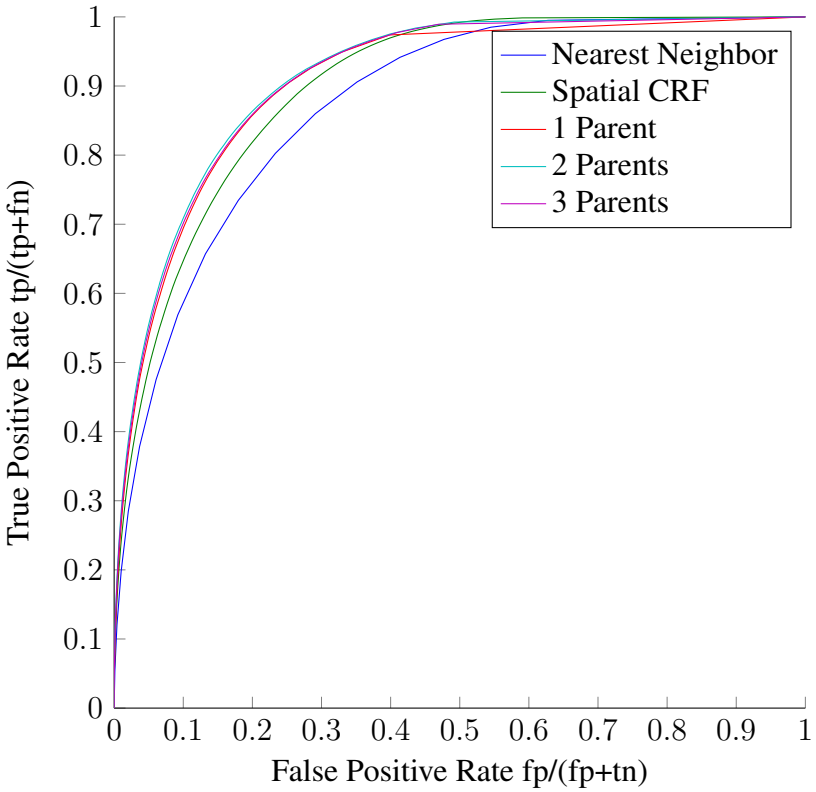


Figure 6-5: A comparison of the prediction performance in terms of the ROC curves for the temporal filtering with different numbers of parents. The temporal filtering does not seem to be particularly sensitive to the number of parents used, as all models provide a similar improvement.

trajectory bundle feasibility.

$$bel(x_t^i) \propto p(x_t^i | x_{t-1}^{p1}, x_{t-1}^{p2}) bel(x_{t-1}^{p1}) bel(x_{t-1}^{p2}) p(x_t^i | \hat{X}_t) \tag{6.9}$$

6.5 Prediction Accuracy

We evaluated the prediction accuracy of the filtering model with different numbers of parents by comparing their receiver operator characteristic curves. The prediction performance evaluation was conducted on the same held-out dataset as was used previously in section 5.1.1. The test data was generated by navigating the vehicle between random locations in the test environment, so we have the temporal sequence of actions, along with the sensor measurements, and feasibility of trajectory bundles at each sample point.

As can be seen in figure 6-5, temporal filtering provided a performance improvement over the predictions from the spatial CRF. The temporal filtering model does not seem to be particularly sensitive to the number of parents, but two parents slightly outperformed one or three parents. As a result, we will restrict our attention to two parents going forward.

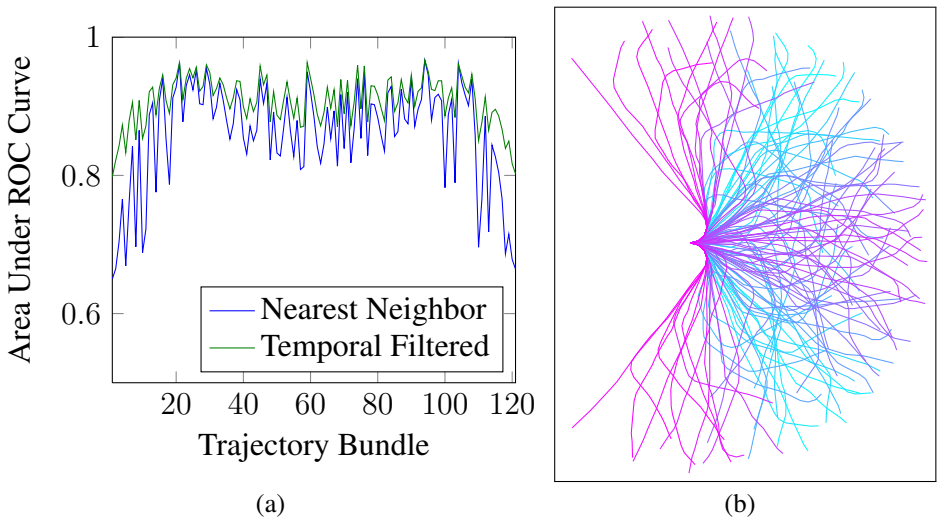


Figure 6-6: An illustrations of where the temporal filtering provides the most improvement. (a) plots the area under the ROC curve for each individual trajectory bundle. (b) draws the trajectory bundle exemplars, colored by the *increase* in the area under the ROC curve. Magenta trajectories received the largest improvement, while cyan trajectories saw the least improvement.

One interesting point to note is to look at which trajectory bundles receive the most benefit from the temporal smoothing. As we noted in figure 5-3, the prediction performance of the nearest neighbor classifier is significantly lower for the bundles at the extremes of the trajectory library. This is likely due to the fact, that most of the area through which these bundles pass is actually outside the field of view of the camera.

If we perform the same analysis, and look at the area under the ROC curve for the predictions with the temporal filtering, we can see that these are in fact the areas where we gain the most improvement. Figure 6-6 shows the area under the ROC curve for each trajectory bundle. The areas with the greatest improvement are the trajectory bundles that go towards the sides, and straight ahead. Intuitively, the areas to the sides should be helped the most by the temporal smoothing since these areas would likely have been in the field of view previously, and one would expect that the prior predictions from when those parts of

the environment were visible will be more accurate than the extrapolation from what is currently visible. As a result, conditioning our predictions on the past history of observations should be expected to improve the accuracy of our predictions.

6.6 Planning Performance

In addition to improving the prediction accuracy in terms of the area under the ROC curve, temporal filtering in the space of trajectory bundles significantly improves the planning performance. We once again compare the success rate of the navigator in reaching the goal as we change the interval between replan steps.

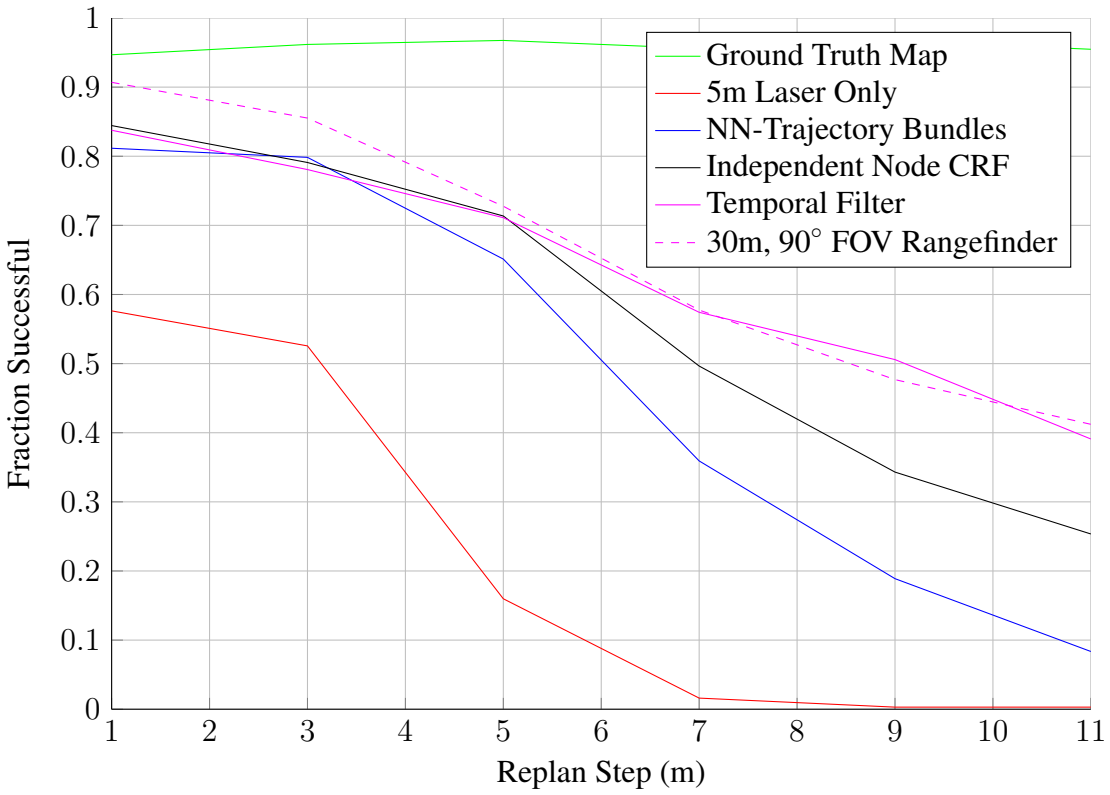


Figure 6-7: A comparison of the success rate in reaching the goal with different prediction methods. The Bayesian temporal filtering provides a significant improvement in the success rate. The improvement becomes more pronounced as the distance between replan steps increases. The planning performance with the temporal filtering is similar to that obtained with a 30m rangefinder.

Figure 6-7 shows the success rate of the vehicle in reaching the goal for the model

with and without temporal filtering. Initially, for short replan distances, all of the models perform roughly the same, however as the replan distances get larger, the temporal filtering model significantly outperforms the others. The performance is largely similar to the case where the vehicle carries a hypothetical rangefinder sensor with a maximum range of 30m. An additional qualitative benefit is that the path taken by the vehicle is smoother. Without the temporal smoothing, the predictions can jump around considerably as the vehicle moves through the environment. However, the temporal filtering makes it so that the predictions have history, and don't change as quickly.

6.6.1 Combining Predictions with Range Sensing

The approaches presented in this thesis are largely aimed at situations where accurate metric sensing is not feasible to obtain, however, even if the vehicle does have access to a powerful range sensor, the predictions should be able to improve the navigation performance by allowing the vehicle to make inferences about regions that are occluded from the range sensor. To investigate this possibility we ran a further set of experiments where we combine the predictions made by our prediction system with the more accurate collision checking capability that is provided by a longer range sensor.

As the vehicle moves through the environment, we accumulate the sensor measurements from the laser rangefinder into an occupancy map. This allows us to perform collision checking of the trajectory library. The collision checking partitions the trajectories in the trajectory library into three groups: trajectories that are known to collide with an obstacle, trajectories that are entirely in known free space, and finally trajectories that pass through some unknown regions.

For this last set of trajectories that pass through the unknown regions, we assign the value output by our temporal filtering model. Figure 6-8 shows the improvement in the planning performance provided by augmenting the range sensing with the predictions from our system. When replanning every 11m, the vehicle is able to reach the goal just over 60% of the time, compared to only 40% of the time using the range sensing alone.

The two sensing modalities were combined in a very loose arrangement. The perfor-

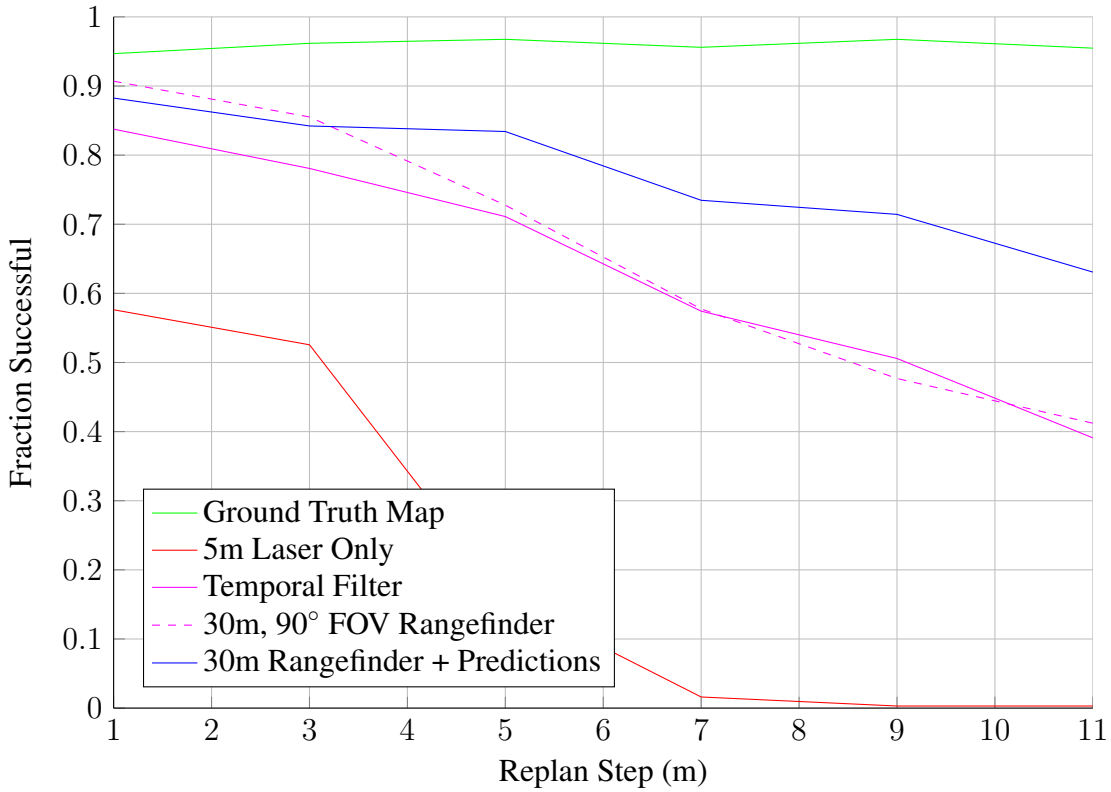


Figure 6-8: A comparison of the success rate in reaching the goal with different prediction methods. When we combine our predictions with the metric information from the 30m rangefinder, we see a significant improvement in the success rate of reaching the goal.

mance of the combination could probably be improved further by using the range measurements directly as features in the prediction models.

Chapter 7

Experimental Analysis

In addition to the simulation experiments presented in the previous chapters, we have conducted a preliminary analysis of our algorithm on a challenging real-world dataset. The data allows us to investigate the prediction accuracy of the nearest neighbor based classifier presented in chapter 5. However, since the data was collected passively, with a person driving the vehicle, we were unable to investigate the closed loop decision making performance. In the future, we hope to extend such experiments to evaluate the decisions made by our system.

Testing the system in the real world allows us to investigate whether the assumptions made by our algorithm are overly restrictive. The core assumption that is made by our algorithms is that the training data used to build the models and make predictions will have similar structure to testing data. If the training data differs considerably from the testing data then the predictions are unlikely to be accurate. The likelihood of encountering such situations can be reduced by ensuring that the vehicle is trained with a large and diverse training set. In addition, it will be important to ensure that the features used are powerful enough to generalize to new environments.

Even with a large dataset however, certain situations could cause problems. A particularly pathological case would be if the vehicle encounters a large mural in the environment that would present fake perspective cues to the vehicle's camera sensor. Since the image features used by our algorithms currently consider only a single image, the system would be unable to differentiate between a flat picture of a scene, and the actual 3D geometry that

was depicted. As a result, the fake perspective cues could draw the vehicle into a collision. These issues could be mitigated by enriching the features used for prediction to include other cues such as parallax from motion between image frames. Such improvements are left for future work. For the current analysis, we use the same features as the simulations.

7.1 Data Collection

The data was collected using the Robust Robotics Group's autonomous wheelchair platform [46], shown in figure 7-1. The vehicle was outfitted with a number of sensors including cameras and lidar sensors. The sensor data was logged to disk, and post-processed offline.

While the wheelchair is capable of turning in place, it was driven such that it was always moving forward at a constant speed. By driving in this manner, the vehicle's dynamics could be approximated by the Dubins vehicle model used in our simulations. The path followed by the vehicle is shown by the blue lines in figure 7-2.

The data was collected in the underground parking garage of the Stata Center at MIT during three data collection sessions. The data was collected at times when only a few cars were parked in the garage. This was done to ensure that the locations of the parked cars were likely to be randomly distributed across the environment. Between sessions, the layout of the cars changed, as can be seen in the maps in figure 7-2.

The maps were created based on the 2D planar lidar data using SLAM system developed in [3]. The maps output by the SLAM algorithm were subsequently manually cleaned to remove noise, and fill holes in the map.

We spatially downsample the camera data to produce an image every $.25$m. Across the three data collection sessions, we recorded roughly 9.5k images. Examples of the images captured by the vehicles camera are shown in figure 7-3.

Using the position estimate from the SLAM algorithm, we registered the images to the occupancy map. To label each image sample, we computed the trajectory bundle feasibility using the occupancy map. We employed the same trajectory bundle library, and underlying trajectory library as in the simulations.

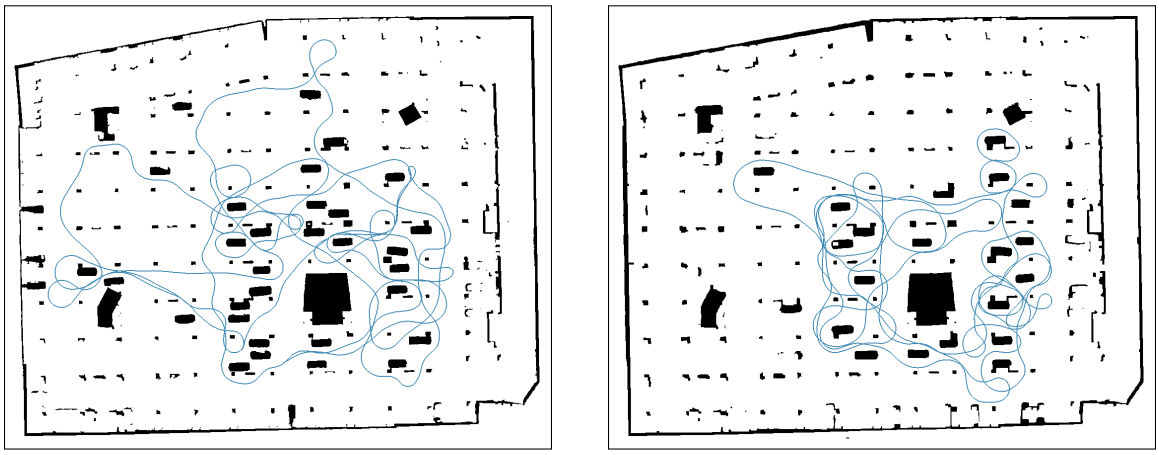
Figure 7-1: A picture of the robust robotics group's autonomous wheelchair. The vehicle is outfitted with a number of sensors including cameras and lidars that we use to collect data.

7.2 Evaluation

We split the dataset into a training set, and a testing set. The training set consisted of the data from two of the data collection sessions for a total of 8087 training samples. The testing set was taken from the third data collection session, and consisted of 1474 test images.

As we did in chapter 5, we employed a nearest neighbor classifier with the GIST feature descriptor to retrieve images from the training database. The retrieved images were used to predict whether each bundle in the trajectory bundle library is feasible from the vantage point where the image was taken.

As a first experiment, we qualitatively investigated the images retrieved by our nearest neighbor classifier, using the GIST image descriptor. As can be seen in figure 7-3, in many



(a) (b)

Figure 7-2: Occupancy maps of the Stata Center parking garage during two of the data collection sessions. Obstacles are drawn in black, while free space is white. The path of the vehicle during the data collection session is drawn by the blue line. While the gross structure of the walls and pillars does not change between sessions, different cars were parked in different places in each data collection session.

cases, the retrieved images have a fairly similar layout of obstacles to the query image. The fine details are often different, but the gross environmental layout in terms of the pillars and cars are similar. The spatial invariance provided by reasoning about the world in terms of trajectory bundles should allow the retrieved scenes to be informative, even though they do not match the query scene exactly.

In other situations the retrieved images seem to be very different, and the predictions are unlikely to be useful. These errors may be due to the lack of specificity in the GIST feature descriptor. Other features might provide richer and more accurate information with which to make the retrievals. Another problem may simply be the small database size. With only 8K images in the database, there are bound to be many scenes for which none of the scenes in the database are similar.

In addition to this qualitative analysis, we computed the receiver operator characteristic curves of the nearest neighbor classifier at different ranges. As was the case in the simulation experiments, the prediction accuracy, as measured by the area under the ROC curve increases at shorter ranges. However, for all ranges, the prediction accuracy is less than in the simulation. Despite the reduced prediction accuracy, the predictions are still better than random.

Figure 7-3: Examples of the images retrieved by the GIST feature distance on the parking garage data. The left column shows the query image. The images to the right show the three nearest neighbors in the database.

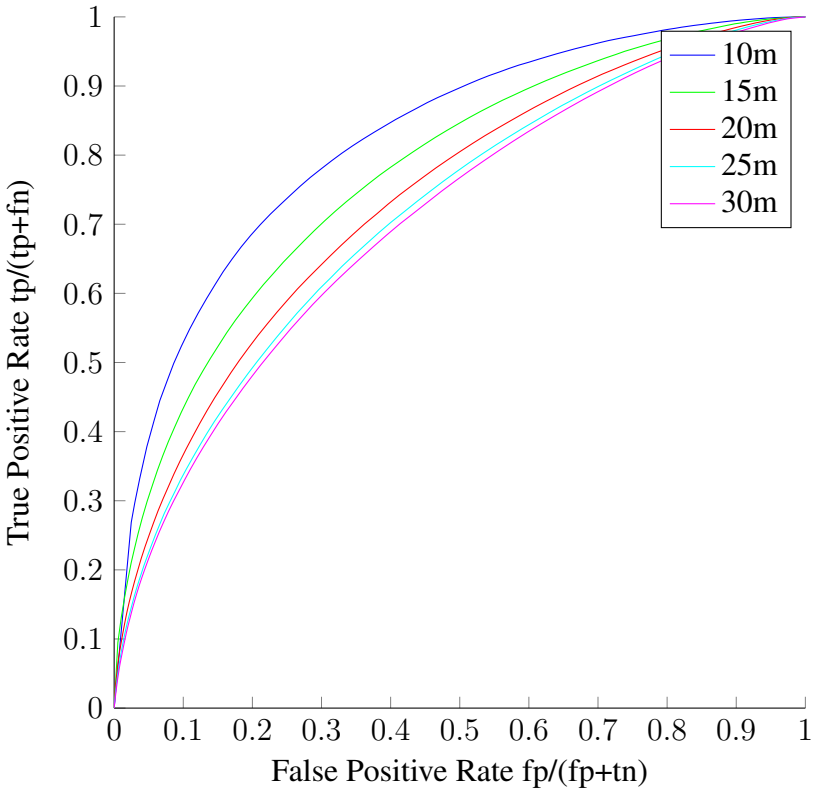


Figure 7-4: A comparison of the prediction performance in terms of the ROC curves for our nearest neighbor classifier using GIST features to make trajectory bundle predictions at different ranges.

Further investigation will be required to determine whether such predictions are accurate enough to improve planning. For the current analysis, we use the algorithms "as is", however for a deployed system, it is expected that further modifications to the algorithms, and components used could significantly improve the performance. While the individual modules may change, we believe that the trajectory bundle prediction and planning framework presented in this thesis will provide a useful paradigm to frame the perception and planning problem.

Chapter 8

Conclusion

In this thesis we have presented the trajectory bundle representation, and developed algorithms for using trajectory bundles for perception. We have presented a proof of concept system that employs trajectory bundles to understand the world around us. The models in our systems are all built from data, and are general enough to allow significant extensions and broad applicability.

We have developed a trajectory clustering algorithm that can be used to group trajectories into trajectory bundles. We developed algorithms for learning discriminative models to predict which trajectory bundles are feasible from local sensor data such as camera images. We then developed models to reason about the structure inherent in these predictions. A conditional random field model was used to reason about the correlations between trajectory bundles in space, taking into account the fact that the bundles will be correlated due to the fact that they will often collide with the same obstacle, or pass through the same opening. Finally, we developed a Bayesian filtering model to take into account the history of sensor measurements to improve the accuracy of our predictions.

In all, the trajectory bundle representation provided a useful way to structure our learning, prediction, and planning algorithms.

8.1 Future Work

We have developed a proof of concept system to explore the use of trajectory bundles. While the results have been encouraging, we believe that we have just barely scratched the surface on an exciting direction in robotics research. As such, there is considerable future work that would be interesting to explore.

Higher Dimensions For simplicity, we have tested our algorithms on a system with a 2D Dubins dynamics model. None of the presented approaches make any assumptions about the 2D nature of the data or model, in particular, we do not make assumptions about a flat ground plane that allows us to project image features to a euclidean space. It will be interesting to see how the presented algorithms scale to planning problems in higher dimensions. A particular challenge will be determining a reasonable trajectory library, and associated library of trajectory bundles. With more degrees of freedom, the system will likely need more underlying trajectories to maintain coverage. However, the abstractions provided by reasoning in terms of trajectory bundles should still allow the creation of relatively compact models.

Richer Dynamics In addition, to understanding vehicles that operate in 3D space, we must also consider vehicles with richer dynamics, for example, vehicles that move at variable speeds. This may be able to be handled by reasoning about a trajectory bundle library with various planning horizons, and simply expanding the trajectory library to contain trajectories with more varied dynamics. We have defined our planning horizon in terms of distance (which is equivalent to time for a constant velocity vehicle), but for a vehicle that operates are various speeds, it may be preferable to construct the trajectory bundles from trajectories that operate over a fixed time horizon. Determining a reasonable similarity metric between trajectories with more diverse dynamics, and a structure for the trajectory bundles will be a challenge.

Multiple Horizons It may also be useful to reason about and maintain a belief over trajectory bundle libraries that extend out to multiple sensing horizons. We believe that this

approach would provide a number of benefits, however it will require careful thought as to how to incorporate the different range and overlapping predictions for planning. A key challenge therein will be how to trade off between the higher confidence at shorter ranges, with the increased progress at longer ranges. The shorter range predictions will require more reliance on the heuristic cost-to-go metric.

Theoretical Analysis We have designed the trajectory bundle library based on our experience and intuitive understanding of the challenges at hand. A deeper understanding of their theoretical properties in terms of coverage and completeness would be very informative. This research could entail optimality bounds in terms of their use for planning. It could also shed light on the desired properties for constructing the trajectory bundle library. A very nice outcome would be to not require the designer to exercise as much judgment in creating the trajectory bundle library.

Global Planning In the thesis, we developed algorithms for local planning and decision making. The estimated belief over the feasibility of trajectory bundles allow us to react to, and avoid obstacles from a much greater range than is possible when relying on metric information alone. However, we rely on a simple heuristic cost-to-go metric to provide guidance from the prediction horizon to the goal. As the vehicle navigates through the environment, it may find itself in a position where it needs to backtrack. In these situations, the planning performance would benefit from better global guidance that takes previous experience into account. The trajectory bundle representation is a local representation of the environment. It aids in interpreting the noisy and ambiguous sensor data measured by the vehicles sensors. It would be useful to investigate how these local predictions could be aggregated into a global representation that could inform navigation when the vehicle must backtrack.

Information Gathering The decision rules employed to guide the vehicle based on our belief over the feasibility of trajectory bundles do not account for the ability of the vehicle to choose actions that will reduce its uncertainty. We believe that the estimated distribution over trajectory bundles would allow the development of algorithms that take into account

information gathering when choosing which trajectory to follow. Such extensions of the algorithms could significantly improve the navigation performance in some situations.

Richer Models In this thesis we developed a Bayesian filtering model to estimate a belief over the feasibility to trajectory bundles. In a number of situations we have made simplifying assumptions that make the probabilistic models easier to learn from data, and perform inference over. As such, the power of our computational models are potentially limited in their inference power. Developing richer models could significantly improve the prediction performance of the algorithms. Of particular interest would be to investigate methods for learning the structure and parameters of dynamic Bayesian networks.

Real World Testing Finally, the learning, and testing presented this thesis were mostly performed in simulation. While the simulation allowed us to explicitly control more variables, and gather a better understanding of the problem, there will undoubtedly be significant further challenges that must be overcome to deploy the presented algorithms on a real robot. We have explicitly constrained the approaches to be general enough that they should transfer over to the real world, however improvements may be necessary.

One area in particular that will require further investigation is on the low level vision processing. We employed a nearest neighbor classifier, retrieving similar scenes to the query image by looking at the distance in terms of the GIST feature distance. To handle the full visual richness of the real world in terms of variable lighting, texture, and geometry, other features will likely provide significant improvement. Combining features computed from other sensing modalities such as optical flow, stereo triangulation, or lidar range measurements would also be very interesting.

In addition, we have assumed that the vehicle will be able to execute the selected trajectory exactly, and with no errors. In the real world, this is obviously not possible. We believe that a planning system such as the one presented in the thesis would provide effective higher level guidance to the vehicle, but it would need to be integrated with low level controllers, and obstacle avoidance routines.

Other Possibilities Beyond these potential areas for future work, we hope that this thesis will inspire research in a number of directions that we have not considered. Hopefully, this will enable new lines of research that will further advance the capabilities of autonomous robots.

Bibliography

[1] F. Andert and F. Adolf. Online world modeling and path planning for an unmanned helicopter. *Autonomous Robots*, 27(3):147–164, 2009.

[2] D. Anguelov, C. Dulong, D. Filip, C. Frueh, S. a. Lafon, R. Lyon, A. Ogale, L. Vincent, and J. Weaver. Google street view: Capturing the world at street level. *Computer*, 43(6):32 –38, 2010.

[3] A. Bachrach, S. Prentice, R. He, and N. Roy. Range–robust autonomous navigation in gps-denied environments. *Journal of Field Robotics*, 28(5):644–666, 2011.

[4] G. Barrows. *Mixed-mode VLSI optic flow sensors for micro air vehicles*. PhD thesis, University of Maryland, 1999.

[5] D. Berenson, P. Abbeel, and K. Goldberg. A robot path planning framework that learns from experience. In *IEEE International Conference Robotics and Automation*, 2012.

[6] D. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *AAAI workshop on knowledge discovery in databases*, volume 2, 1994.

[7] A. Beyeler, J.-C. Zufferey, and D. Floreano. Vision-based control of near-obstacle flight. *Autonomous Robots*, 27:201–219, 2009.

[8] S. Bhattacharya, M. Likhachev, and V. Kumar. Topological constraints in search-based robot path planning. *Autonomous Robots*, pages 1–18, 2012.

[9] C. Bishop. *Pattern recognition and machine learning*, volume 4. springer, 2006.

[10] B. Boots, S. Siddiqi, and G. Gordon. Closing the learning-planning loop with predictive state representations. *International Journal of Robotics Research*, 30(7): 954–966, 2011.

[11] M. Bosse, P. Newman, J. Leonard, and S. Teller. Simultaneous localization and map building in large-scale cyclic environments using the atlas framework. *International Journal of Robotics Research*, 23(12):1113–1139, 2004.

[12] M. Branicky, R. Knepper, and J. Kuffner. Path and trajectory diversity: Theory and algorithms. In *IEEE International Conference Robotics and Automation*, pages 1359–1364, 2008.

[13] R. Brooks. Solving the find-path problem by representing free space as generalized cones. Technical Report AI Memos (AIM-674), MIT, 1982.

[14] A. Bry, A. Bachrach, and N. Roy. State estimation for aggressive flight in gps-denied environments using onboard sensing. In *IEEE International Conference Robotics and Automation*, 2012.

[15] J. Buhmann, W. Burgard, A. Cremers, D. Fox, T. Hofmann, F. Schneider, J. Strikos, and S. Thrun. The mobile robot rhino. *AI Magazine*, 16(2):31, 1995.

[16] C. Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[17] D. Buzan, S. Sclaroff, and G. Kollios. Extraction and clustering of motion trajectories in video. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 521–524, 2004.

[18] R. Chatila and J. Laumond. Position referencing and consistent world modeling for mobile robots. In *IEEE International Conference Robotics and Automation*, volume 2, pages 138–145, 1985.

[19] H. Chitsaz and S. LaValle. Time-optimal paths for a Dubins airplane. In *IEEE Conference on Decision and Control*, pages 2379–2384, 2007.

[20] H. Choset and J. Burdick. Sensor-based exploration: The hierarchical generalized voronoi graph. *International Journal of Robotics Research*, 19(2):96–125, 2000.

[21] C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.

[22] A. Coates, P. Abbeel, and A. Ng. Learning for control from multiple demonstrations. In *International Conference On Machine Learning*, pages 144–151, 2008.

[23] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[24] M. Cummins and P. Newman. FAB-MAP: Probabilistic localization and mapping in the space of appearance. *International Journal of Robotics Research*, 27(6):647–665, 2008.

[25] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 886–893, 2005.

[26] J. Davis, R. Ramamoorthi, and S. Rusinkiewicz. Spacetime stereo: A unifying framework for depth from triangulation. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages II–359, 2003.

[27] A. Davison, I. Reid, N. Molton, and O. Stasse. Monoslam: Real-time single camera slam. *IEEE Transactions Pattern Analysis and Machine Intelligence*, 29(6):1052–1067, 2007.

[28] E. Delage, H. Lee, and A. Ng. A dynamic bayesian network model for autonomous 3d reconstruction from a single indoor image. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 2418–2428, 2006.

[29] D. Dey, T. Liu, B. Sofman, and D. Bagnell. Efficient optimization of control libraries. Technical Report CMU-RI-TR-11-20, CMU, 2011.

[30] L. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79(3):497–516, 1957.

[31] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part I. *IEEE Robotics and Automation Magazine*, 13(2):99–110, 2006.

[32] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46 –57, june 1989.

[33] L. Erickson and S. LaValle. Survivability: Measuring and ensuring path diversity. In *IEEE International Conference Robotics and Automation*, pages 2068–2073, 2009.

[34] J. Feddema and C. Little. Rapid world modeling: fitting range data to geometric primitives. In *IEEE International Conference Robotics and Automation*, volume 4, pages 2807–2812, 1997.

[35] A. Flint, D. Murray, and I. Reid. Manhattan scene understanding using monocular, stereo, and 3d features. In *International Conference on Computer Vision*, pages 2228–2235, 2011.

[36] E. Frazzoli, M. A. Dahleh, and E. Feron. Real-time motion planning for agile autonomous vehicles. *AIAA Journal of Guidance, Control, and Dynamics*, 25(1):116–129, 2002.

[37] B. Frey and D. Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007.

[38] N. Friedman and M. Goldszmidt. Learning bayesian networks with local structure. In *Conference on Uncertainty in artificial intelligence*, pages 252–262, 1996.

[39] N. Friedman, K. Murphy, and S. Russell. Learning the structure of dynamic probabilistic networks. In *Conference on Uncertainty in artificial intelligence*, pages 139–147, 1998.

[40] I. Givoni and B. Frey. A binary variable model for affinity propagation. *Neural computation*, 21(6):1589–1600, 2009.

[41] C. Goldfeder and P. Allen. Data-driven grasping. *Autonomous Robots*, pages 1–20, 2011.

[42] C. Green and A. Kelly. Toward optimal sampling in the space of paths. *International Journal of Robotics Research*, pages 281–292, 2011.

[43] W. Green, P. Oh, and G. Barrows. Flying insect inspired vision for autonomous aerial robot maneuvers in near-earth environments. In *IEEE International Conference Robotics and Automation*, volume 3, pages 2347–2352, 2004.

[44] A. Gupta, A. Efros, and M. Hebert. Blocks world revisited: Image understanding using qualitative geometry and mechanics. *European Conference on Computer Vision*, pages 482–496, 2010.

[45] V. Hedau, D. Hoiem, and D. Forsyth. Recovering free space of indoor scenes from a single image. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[46] S. Hemachandra, T. Kollar, N. Roy, and S. Teller. Following and interpreting narrated guided tours. In *IEEE International Conference Robotics and Automation*, pages 2574–2579, 2011.

[47] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGB-D mapping: Using depth cameras for dense 3d modeling of indoor environments. In *International Symposium on Experimental Robotics*, 2010.

[48] D. Hoiem, A. Efros, and M. Hebert. Recovering surface layout from an image. *International Journal of Computer Vision*, 75:151–172, 2007.

[49] D. Hoiem, A. Stein, A. Efros, and M. Hebert. Recovering occlusion boundaries from a single image. In *International Conference on Computer Vision*, pages 1–8, 2007.

[50] Hokuyo. Hokuyo URG-04LX scanning rangefinder. https://www.hokuyo-aut.jp/, 2010.

[51] B. Horn. *Shape from shading: A method for obtaining the shape of a smooth opaque object from one view*. PhD thesis, Massachusetts Institute of Technology, 1970.

[52] T. Howard, C. Green, A. Kelly, and D. Ferguson. State space sampling of feasible motions for high-performance mobile robot navigation in complex environments. *Journal of Field Robotics*, 25(6-7):325–345, 2008.

[53] S. Hrabar. 3D path planning and stereo-based obstacle avoidance for rotorcraft uavs. In *IEEE International Conference on Intelligent Robots and Systems*, pages 807–814, 2008.

[54] S. Hrabar and G. Sukhatme. Vision-based navigation through urban canyons. *Journal of Field Robotics*, 26(5):431–452, 2009.

[55] W. Huang, G. Grudic, and L. Matthies. Special issue: Special issue on LAGR program, part I editorial. *Journal of Field Robotics*, 26(1):1–2, 2009.

[56] J. Humbert, J. Conroy, C. Neely, and G. Barrows. Wide-field integration methods for visuomotor control. *Flying Insects and Robots*, page 63, 2009.

[57] L. D. Jackel, E. Krotkov, M. Perschbacher, J. Pippine, and C. Sullivan. The DARPA LAGR program: Goals, challenges, methodology, and phase i results. *Journal of Field Robotics*, 23(11-12):945–973, 2006.

[58] N. Jetchev and M. Toussaint. Trajectory prediction in cluttered voxel environments. In *IEEE International Conference Robotics and Automation*, pages 2523–2528, 2010.

[59] M. Johnson, F. Cole, A. Raj, and E. Adelson. Microgeometry capture using an elastomeric sensor. *ACM Transactions on Graphics*, 30(4):46, 2011.

[60] J. Joseph, F. Doshi-Velez, A. Huang, and N. Roy. A bayesian nonparametric approach to modeling motion patterns. *Autonomous Robots*, pages 1–18, 2011.

[61] R. Kalman et al. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[62] B. Kaneva. *Large databases of real and synthetic images for feature evaluation and prediction*. PhD thesis, Massachusetts Institute of Technology, 2012.

[63] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30(7):846–894, 2011.

[64] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions Robotics and Automation*, 12(4):566–580, 1996.

[65] A. Kelly and A. Stentz. Rough terrain autonomous mobilitypart 1: A theoretical analysis of requirements. *Autonomous Robots*, 5(2):129–161, 1998.

[66] R. Knepper and M. Mason. Path diversity is only part of the problem. In *IEEE International Conference Robotics and Automation*, pages 3224–3229, 2009.

[67] R. Knepper, S. Srinivasa, and M. Mason. An equivalence relation for local path sets. *Workshop on the Algorithmic Foundations of Robotics*, pages 19–35, 2011.

[68] D. Kortenkamp and T. Weymouth. Topological mapping for mobile robots using a combination of sonar and vision sensing. In *National Conference on Artificial Intelligence*, pages 979–979, 1995.

[69] F. Kschischang, B. Frey, and H. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001.

[70] D. Kuan, J. Zamiska, and R. Brooks. Natural decomposition of free space for path planning. In *IEEE International Conference Robotics and Automation*, volume 2, pages 168–173, 1985.

[71] B. Kuipers and Y. Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Robotics and Autonomous Systems*, 8 (1-2):47–63, 1991.

[72] H. Kurniawati, T. Bandyopadhyay, and N. Patrikalakis. Global motion planning under uncertain motion, sensing, and environment map. In *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA, June 2011.

[73] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *International Conference On Machine Learning*, pages 282–289, 2001. ISBN 1-55860-778-1.

[74] S. LaValle and J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400, 2001.

[75] D. Leake. Case-based reasoning. *The knowledge engineering review*, 9(01):61–64, 1994.

[76] D. Lee, M. Hebert, and T. Kanade. Geometric reasoning for single image structure recovery. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2136–2143, 2009.

[77] C. Liu, J. Yuen, and A. Torralba. Nonparametric scene parsing: Label transfer via dense scene alignment. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1972–1979, 2009.

[78] A. Lookingbill, J. Rogers, D. Lieb, J. Curry, and S. Thrun. Reverse optical flow for self-supervised adaptive autonomous robot navigation. *International Journal of Robotics Research*, 74(3):287–302, 2007.

[79] T. Lozano-Perez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, 100(2):108–120, 1983.

[80] L. Matthies and S. Shafer. Error modeling in stereo navigation. *IEEE Journal of Robotics and Automation*, 3(3):239–248, 1987.

[81] J. Michels, A. Saxena, and A. Y. Ng. High speed obstacle avoidance using monocular vision and reinforcement learning. In *International Conference On Machine Learning*, ICML '05, pages 593–600, 2005. ISBN 1-59593-180-5.

[82] H. Moravec. Sensor fusion in certainty grids for mobile robots. *AI magazine*, 9(2): 61, 1988.

[83] B. Morris and M. Trivedi. A survey of vision-based trajectory learning and analysis for surveillance. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(8):1114–1127, 2008.

[84] B. Morris and M. Trivedi. Learning trajectory patterns by clustering: Experimental studies and comparative evaluation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 312–319, 2009.

[85] T. Mueller and J. DeLaurier. An overview of micro air vehicle aerodynamics. *Progress in Astronautics and Aeronautics*, 195:1–10, 2001.

[86] K. Murphy. *Dynamic bayesian networks: representation, inference and learning*. PhD thesis, University of California, 2002.

[87] K. Murphy, Y. Weiss, and M. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Conference on Uncertainty in artificial intelligence*, pages 467–475, 1999.

[88] B. Nabbe, D. Hoeim, A. Efros, and M. Herbert. Opportunistic use of vision to push back the path-planning horizon. In *IEEE International Conference on Intelligent Robots and Systems*, pages 2388–2393, 2006.

[89] S. Nowozin and C. Lampert. *Structured learning and prediction in computer vision*, volume 6. Now Publishers, 2011.

[90] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42(3):145–175, 2001.

[91] S. T. OCallaghan and F. T. Ramos. Gaussian process occupancy maps. *International Journal of Robotics Research*, 31(1):42–62, 2012.

[92] A. Perez, S. Karaman, A. Shkolnik, E. Frazzoli, S. Teller, and M. Walter. Asymptotically-optimal path planning for manipulation using incremental sampling-based algorithms. In *IEEE International Conference on Intelligent Robots and Systems*, pages 4307–4313, 2011.

[93] D. Pomerleau. Neural network vision for robot driving. *Intelligent Unmanned Ground Vehicles*, pages 53–72, 1997.

[94] P. J. Rousseeuw and L. Kaufman. Clustering by means of mediods. *Statistical data analysis based on the L1-norm and related methods*, 405, 1987.

[95] R. Rusu, Z. Marton, N. Blodow, M. Dolha, and M. Beetz. Towards 3d point cloud based object maps for household environments. *Robotics and Autonomous Systems*, 56(11):927–941, 2008.

[96] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 26(1):43–49, 1978.

[97] A. Saxena, M. Sun, and A. Ng. Make3d: Learning 3d scene structure from a single still image. *IEEE Transactions Pattern Analysis and Machine Intelligence*, 31(5): 824–840, 2009.

[98] R. Schapire. The boosting approach to machine learning: An overview. *MSRI Workshop on Nonlinear Estimation and Classification*, pages 149–172, 2001.

[99] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47(1):7–42, 2002.

[100] S. Scherer, S. Singh, L. Chamberlain, and M. Elgersma. Flying fast and low among obstacles: Methodology and experiments. *International Journal of Robotics Research*, 27(5):549–574, 2008.

[101] M. Schmidt. Ugm: A matlab toolbox for probabilistic undirected graphical models, 2011. URL `http://www.di.ens.fr/~mschmidt/Software/UGM.html`.

[102] M. Schmidt, A. Niculescu-Mizil, and K. Murphy. Learning graphical model structure using l1-regularization paths. In *National Conference on Artificial Intelligence*, volume 22, page 1278, 2007.

[103] T. Schouwenaars, J. How, and E. Feron. Receding horizon path planning with implicit safety guarantees. In *American Control Conference*, volume 6, pages 5576–5581, 2004.

[104] P. Sermanet, M. Scoffier, and C. LeCun. Learning maneuver dictionaries for ground robot planning. In *International Symposium on Robotics*, 2008.

[105] G. Shani, J. Pineau, and R. Kaplow. A survey of point-based pomdp solvers. *Autonomous Agents and Multi-Agent Systems*, pages 1–51, 2012.

[106] G. Sibley, G. Sukhatme, and L. Matthies. The iterated sigma point kalman filter with applications to long range stereo. In *Proceedings of Robotics: Science and Systems*, pages 263–270, 2006.

[107] S. Simhon and G. Dudek. A global topological map formed by local metric maps. In *IEEE International Conference on Intelligent Robots and Systems*, volume 3, pages 1708–1714, 1998.

[108] J. Sivic, B. Russell, A. Efros, A. Zisserman, and W. Freeman. Discovering objects and their location in images. In *International Conference on Computer Vision*, volume 1, pages 370–377, 2005.

[109] M. Smith, I. Posner, and P. Newman. Efficient non-parametric surface representations using active sampling for push broom laser data. In *Proceedings of Robotics: Science and Systems*, 2010.

[110] M. Srinivasan, J. Chahl, K. Weber, S. Venkatesh, M. Nagle, and S. Zhang. Robot navigation inspired by principles of insect vision. *Robotics and Autonomous Systems*, 26(2):203–216, 1999.

[111] S. Thrun. Robotic mapping: A survey. *Exploring artificial intelligence in the new millennium*, 1:1–35, 2003.

[112] S. Thrun and B. Wegbreit. Shape from symmetry. In *International Conference on Computer Vision*, volume 2, pages 1824–1831, 2005.

[113] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. Intelligent Robotics and Autonomous Agents. MIT Press, 2005.

[114] A. Torralba and A. Oliva. Depth estimation from image structure. *IEEE Transactions Pattern Analysis and Machine Intelligence*, 24(9):1226–1238, 2002.

[115] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon. Bundle adjustment a modern synthesis. In *Vision Algorithms: Theory and Practice*, volume 1883 of *Lecture Notes in Computer Science*, pages 153–177. Springer, 2000.

[116] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6(2):1453, 2006.

[117] S. Ullman. The interpretation of structure from motion. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 203(1153):405–426, 1979.

[118] N. Vandapel, J. Kuffner, and O. Amidi. Planning 3-d path networks in unstructured environments. In *IEEE International Conference Robotics and Automation*, pages 4624–4629, 2005.

[119] F. von Hundelshausen, M. Himmelsbach, F. Hecker, A. Mueller, and H. Wuensche. Driving with tentacles: Integral structures for sensing and motion. *Journal of Field Robotics*, 25(9):640–673, 2008.

[120] M. Walter, R. Eustice, and J. Leonard. Exactly sparse extended information filters for feature-based slam. *International Journal of Robotics Research*, 26(4):335–359, 2007.

[121] J. Xiao, J. Hays, K. Ehinger, A. Oliva, and A. Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3485–3492, 2010.

[122] E. Xing, A. Ng, M. Jordan, and S. Russell. Distance metric learning, with application to clustering with side-information. *Advances in neural information processing systems*, 15:505–512, 2002.

[123] B. Yamauchi. A frontier-based approach for autonomous exploration. In *IEEE International Symposium on Computational Intelligence, Robotics and Automation*, pages 146–151, 1997.

[124] R. Zhang, P. Tsai, J. Cryer, and M. Shah. Shape-from-shading: a survey. *IEEE Transactions Pattern Analysis and Machine Intelligence*, 21(8):690–706, 1999.

[125] J. Zufferey and D. Floreano. Fly-inspired visual steering of an ultralight indoor aircraft. *IEEE Transactions on Robotics*, 22(1):137–146, 2006.