Integrated Motion Planning and Model Learning for Mobile Robots with Application to Marine Vehicles

by

Matthew Greytak

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2009

Author .....
Department of Mechanical Engineering
August 7, 2009

Certified by .....
Franz S. Hover
Assistant Professor
Thesis Supervisor

Accepted by .....
David E. Hardt
Chairman, Department Committee on Graduate Students

Integrated Motion Planning and Model Learning for Mobile Robots with Application to Marine Vehicles

by

Matthew Greytak

Submitted to the Department of Mechanical Engineering
on August 7, 2009, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Mechanical Engineering

Abstract

Robust motion planning algorithms for mobile robots consider stochasticity in the dynamic model of the vehicle and the environment. A practical robust planning approach balances the duration of the motion plan with the probability of colliding with obstacles. This thesis develops fast analytic algorithms for predicting the collision probability due to model uncertainty and random disturbances in the environment for a planar holonomic vehicle such as a marine surface vessel. These predictions lead to a robust motion planning algorithm that finds the optimal motion plan quickly and efficiently. By incorporating model learning into the predictions, the integrated algorithm exhibits emergent active learning strategies to autonomously acquire the model data needed to safely and effectively complete the mission.

The motion planner constructs plans through a known environment by concatenating maneuvers based upon speed controller setpoints. A model-based feedforward/feedback controller is used to track the resulting reference trajectory, and the model parameters are learned online with a least squares regression algorithm. The path-following performance of the vehicle depends on the effects of unknown environmental disturbances and modeling error. The convergence rate of the parameter estimates depends on the motion plan, as different plans excite different modes of the system. By predicting how the collision probability is affected by the parameter covariance evolution, the motion planner automatically incorporates active learning strategies into the motion plans. In particular, the vehicle will practice maneuvers in the open regions of the configuration space before using them in the constrained regions to ensure that the collision risk due to modeling error is low. High-level feedback across missions allows the system to recognize configuration changes and quickly learn new model parameters as necessary. Simulations and experimental results using an autonomous marine surface vessel are presented.

Thesis Supervisor: Franz S. Hover
Title: Assistant Professor

Acknowledgments

I would like to thank my advisor, Prof. Franz Hover, and Prof. Michael Triantafyllou for their support throughout my undergraduate and graduate studies. Prof. Brian Williams and various members of his research group, including Hiro Ono and Hui Li, provided great insights during the development of this thesis. I would also like to thank Prof. Nick Roy, Mike Benjamin, Steve Licht, Josh Taylor and Brendan Englot for their advice and support. I am grateful to Sheila McNary for her administrative help. Finally, I would like to thank my family for their support during this research effort.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Classical control theory has been applied successfully to ship maneuvering problems for many years. More recently, modern techniques such as adaptive and non-linear control have appeared in the literature. Meanwhile, the robotics and artificial intelligence (AI) communities have developed many robust search and planning algorithms for generating paths and action sequences. This thesis combines the benefits of both areas of research to generate a robust planning and navigation system for marine vehicles.

A recent document published by the US Navy outlines their plans for the research and development of Unmanned Surface Vehicles (USVs) in the near future [61]. The main message of the document is that the Navy wishes to get these systems operational and in use as soon as possible, on scales from custom 3-meter vehicles to larger 7-meter Rigid Inflatable Boat (RIB)-based vehicles and 11-meter planing craft. One application for these vehicles is for patrolling a harbor environment using video and chemical sensors. The operating environment has many obstacles in the form of shorelines, docks, and other vessels, and disturbances such as wind, waves, and currents. An autonomous patrol craft must be able to plan a safe path through this environment to a goal location in real-time. The characteristics of the craft may change over time, either through gradual wear or due to a configuration change for a new mission. The vessel must be able to adapt its controller and its motion plans to these changes.

The autonomous harbor patrol craft is a specific example of a general class of configurable robots that must be able to navigate through a cluttered environment with very little human interaction. Consider a commercial robot company that sells a standardized robot that can subsequently be modified by the end user with various attachments or enhancements. When the robots are shipped from the factory, the final configuration of each robot is unknown to the designers. However, an end user who reconfigures the robot has no desire (or no ability) to adjust control parameters or planning parameters to reflect the new dynamic model for the robot. The robot must be able to recognize configuration changes and learn the new model while performing its commanded tasks.

In this thesis we develop an integrated motion planning and model learning algorithm for mobile robots that automatically utilizes learning strategies to identify system parameters while finding an optimal route through a cluttered environment. In particular, the vehicle will practice maneuvers in the open regions of the space so that errors are minimized when those maneuvers are used near obstacles.

The emergence of practicing behavior depends on two key predictions: the model uncertainty evolution throughout each motion plan, and the effect that the uncertainty evolution has on the cost function. If those two effects are predicted for each motion plan, then the benefits of practicing are revealed in the cost function, and the planner will choose to perform practicing maneuvers when the benefits outweigh the added costs such as time and fuel usage. While those two predictions are straightforward for a very simple mobile robot model, they are more difficult for a mobile robot with real dynamics.

The specific class of mobile robots that we focus on in this thesis is planar holonomic vehicles, which describes marine vehicles, aerial vehicles and space vehicles that operate in a planar environment. Process noise, sensor noise, and parameter uncertainty are all assumed to be Gaussian random variables. All aspects of the problem are solved with analytic predictions rather than particle simulations such as Monte Carlo to ensure that the planner is fast and repeatable. The problem can be broken down into three core issues: motion planning, model learning, and planning

under uncertainty. The following sections describe these issues in more depth.

1.1 Robust Motion Planning

The overall algorithm for planning and learning is built upon a robust motion planning algorithm. Motion planning is the process through which the vehicle chooses a nominal trajectory through the environment that the vehicle must follow. The control inputs for a marine vehicle are forward thrust, lateral thrust, and possibly an independent yaw moment. Because the input space is continuous, the planning problem is infinite-dimensional and standard optimization techniques are impractical.

To make the motion planning problem tractable, a discretization of the problem is necessary. One approach is to discretize the state and action spaces, as well as discretizing time, in a partially observable Markov decision process (POMDP). The optimal solution to the POMDP is the optimal policy, which maps observed states to actions. However, achieving fine control resolution would require a dense discretization of the state and action spaces, which is impractical for a robot with multiple degrees of freedom.

In our approach, motion plans are constructed from discrete maneuvers that are concatenated together. The Maneuver Automaton is a convenient framework for this problem: each maneuver connects discrete points in the velocity space, so a motion plan is represented as a sequence of speed controller setpoints. Waypoints can be used to enrich the planning domain. This approach preserves continuous time and continuous states, but by discretizing the problem with a finite set of maneuvers and waypoints the planning problem is tractable. The problem is then solved with an expanding tree search algorithm to find the best sequence of maneuvers that arrives at the goal without hitting any obstacles along the way.

During the execution of the plan, the controller simply shifts to the appropriate setpoint at the appropriate time to cause the vehicle to follow the reference trajectory. The controller we employ is a linear quadratic regulator (LQR) designed around the dynamic vehicle model. This approach is open-loop in the sense that position or

orientation perturbations will not be corrected by the velocity controller. To add position feedback, we incorporate waypoints into the motion planning framework.

While a motion plan may be nominally collision-free, environmental disturbances may result in a non-neglible probability that the vehicle will hit an obstacle. Furthermore, instantaneous velocity setpoint changes may result in transient path errors even in the absence of external disturbances. It is therefore important to be able to predict both the mean path-following error as well as the variance of the error. From these statistics it is possible to compute the probability that the vehicle will hit an obstacle when following any particular motion plan. Given the choice between two motion plans of the same duration in which one plan has a higher probability of hitting an obstacle, we would like the planner to choose the safer plan. To accomplish this, we insert the collision probability into the planner's cost function. This is not the same as penalizing large path errors; it does not matter if there are large errors if there are no obstacles nearby that will result in a collision. By incorporating the collision probability, the motion planner is made more robust to disturbances. This robust motion planner is the core of the integrated planning and learning algorithm discussed later.

1.2 Model Learning

The controller used by the vehicle is model-based, meaning that it depends on an estimate of the vehicle's parameters, such as mass and drag. In general these parameters may not be known with a high degree of confidence or they may change over time (either gradually or discretely, as when the vehicle configuration changes for a new mission). Because the dynamic vehicle model is linear in the parameters, the least squares regression algorithm is an appropriate choice for learning the parameters. For compatibility with the motion planning algorithm, the learning takes place in stages: at the end of each maneuver the least squares learning algorithm is applied to the input/output data from that maneuver, then the controller is redesigned based on the new parameter estimates for use in the next maneuver. This approach results in a

more immediate effect on the path-following performance than waiting until the end of the mission to update the parameters, and it is easier to analyze than the recursive least squares algorithm.

The motion planner requires an *a priori* estimate of the parameter values to generate feasible motion plans, even if the confidence in those parameter estimates is low (i.e. the *a priori* parameter variance is high). As the vehicle executes the motion plan, the parameter estimates will improve and the confidence will increase (the parameter variance will decrease). It is important for the planner to be able to predict this parameter variance evolution as it affects the collision probability and the overall cost of each candidate plan. The parameter convergence is not uniform; it is affected by the information content of the inputs to the learning algorithm, which in turn depend on the motion plan and the path-following error statistics. For example, the parameters related to the yaw dynamics of the vehicle will not improve if the vehicle only drives in a straight line. Predicting the parameter convergence is equivalent to predicting the expected inverse of the information content associated with each parameter. This calculation is very difficult because the input data is correlated in time, yet it is the key component in the integrated planning and model learning algorithm discussed in the next section.

The parameter convergence prediction relies on assumptions about the process noise and the sensor noise in the system, the *a priori* parameter variance, and the steady nature of the true parameter values. If the noise assumptions are incorrect, the initial confidence in the parameter values is not well known, or the parameter values change, then the predicted parameter variance at the end of the mission will be different from the true parameter variance (as determined by many executions of the motion plan, i.e. Monte Carlo simulations). If it is possible to estimate the true parameter variance at the end of the mission based on the prediction error from the mission data, then the *a priori* parameter variance for the next mission can be adjusted appropriately. In this way, the vehicle automatically adjusts to the noise levels in the system, initial confidence errors, and changes to the true parameter values.

1.3 An Integrated Motion Planning and Model Learning Algorithm

The previous section described how to predict the parameter uncertainty for each motion plan considered by the motion planner. To make the optimal motion plan robust against model uncertainty as well as process noise, we must be able to predict the effect of model uncertainty on the collision probability and consequently the planner's cost function. Expressing the collision probability in terms of the parameter uncertainty is straightforward for simple robots but it is quite difficult for a holonomic vehicle with an LQR controller. For the more complex vehicle model we use numerical quadrature to evaluate the collision probability at specific samples in the parameter space and combine those collision probabilities with an appropriate weighting function. A very simple diagram of the integrated planning approach is shown in Figure 1-1.

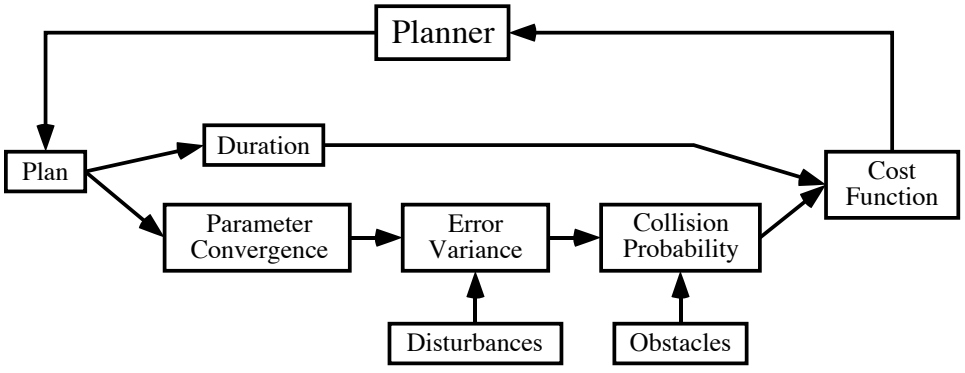

Figure 1-1: A basic diagram of the integrated motion planning and model learning algorithm. The planner predicts how each candidate motion plan contributes to reducing model uncertainty, and it predicts how that model uncertainty evolution affects the collision probability.

Once the planner can predict the parameter convergence for each candidate plan and also predict the effect of that parameter convergence on the cost function, then some interesting behaviors emerge. If a mission requires very tight path-following at a particular point, then the planner will ensure that the relevant parameters have converged to a great extent before that point in the plan; this may require "practicing"

similar maneuvers earlier in the plan to expose the learning algorithm to enough data that the parameter error will be low. In other words, the vehicle will employ active learning as necessary, not because it is explicitly told to learn the parameters, but because it understands that the convergence of certain parameters is necessary to complete the mission successfully.

1.4 Novel Contributions in the Thesis

This section lists the various novel contributions in the thesis, along with relevant papers published by the author.

- Added waypoints to the Maneuver Automaton framework and eliminated the need for motion primitives as a bridge between trim maneuvers. [37, 40].

- Derived predictions of the mean and variance of the path-following error for a planar holonomic vehicle [39].

- Solved the particle absorption prediction problem for dynamic systems of second-order and higher. Derived predictions for the collision probability based on the path-following error statistics and incorporated the collision probability into a robust motion planner [37, 40].

- Derived predictions of the parameter variance evolution when using the least squares learning algorithm applied to a dynamic system with correlations through time.

- Derived posterior parameter variance estimates based on the prediction error to detect changes to the model parameters.

- Developed the integrated motion planning and model learning algorithm to put learning into the motion planning framework, resulting in autonomous learning strategies [41].

1.5 Outline of the Thesis

Chapter 2 surveys the existing literature on the various aspects of the thesis. It first covers the existing practices in marine vehicle navigation and control, including waypoint-based path following and various techniques for lower-level velocity control. As many marine vehicle controllers use adaptation to track the vehicle system parameters, we look at well-known regression techniques and active learning. The bulk of this thesis is built on a kinodynamic motion planner for holonomic vehicles, so Chapter 2 discusses the requirements of a motion planner and the relevant frameworks and graph search algorithms. Next we discuss some aspects of the general topic of planning under uncertainty. Finally we review some of the existing literature on the important tradeoff of exploration vs. exploitation that governs many planning algorithms in uncertain environments.

The construction and analysis of the robust kinodynamic motion planner is presented in Chapter 3. The vehicle we consider is an underactuated planar holonomic vehicle, meaning that the vehicle experiences sideslip that it cannot control independently from the surge and yaw states. After establishing the dynamic and kinematic model for the vehicle, we develop a simple control law and we study the resulting path-following error statistics. The motion planner is made robust by evaluating the collision probability for each plan; this is a non-trivial calculation that is based on the error statistics and the vehicle dynamics. Next, Chapter 3 presents the A* search algorithm that finds the optimal plan from the infinite set of feasible motion plans. Finally, experimental results using a small autonomous vehicle model are presented.

The parameters used by the controller are learned with least squares regression. The regression framework is presented in Chapter 4; it is straightforward except for some subtleties that arise when learning only a subset of all the parameters. The planner must be able to predict the expected convergence rate of the parameter values, which is characterized by the parameter variance evolution. Predicting the parameter variance evolution when learning from a system with correlations in time hinges on the prediction of the expected inverse of the information matrix used by the

learning algorithm. This prediction is developed in great detail in Chapter 4. Next we discuss how to update the parameter variances after the plan has been executed and some learning has taken place.

In Chapter 5 all the pieces are brought together and the motion planner is augmented with information about the parameter uncertainty that arises from the learning algorithm. To understand how the integrated planning and learning algorithm can generate emerging practicing behaviors in the vehicle, we present a simple 2D robot model for which the parameter covariance evolution and the resulting effect on the error variance can be calculated analytically. Based on that example we try to evaluate the problem conditions under which the vehicle exhibits those practicing behaviors. Next the problem is extended to the holonomic vehicle; in that case, parameter uncertainty is handled with numerical quadrature. Chapter 5 concludes with experimental results showing how the vehicle uses its understanding of uncertainty to autonomously perform active learning, when necessary, to learn a model or adapt to changes in the vehicle or the environment.

Chapter 6 summarizes the novel contributions in the thesis and discusses promising avenues for future research. The autonomous marine vehicles used in the experiments are described in Appendix A. Appendix B describes some of the numerical procedures used in the implementation of the planning and prediction algorithms.

Chapter 2

Background

This chapter provides some background material for the integrated motion planning and model learning algorithm described in Chapter 1. Because the main application of this work is marine vehicles, we begin with a summary of existing techniques in marine vehicle navigation and control. Next we discuss model learning and adaptive control strategies. The motion planning algorithm is the core component of this thesis, and it is based on an existing motion planning framework and search algorithm which are described in this chapter. Next we discuss how to handle uncertainty in the environment and in the dynamic model of the vehicle when designing motion plans. We conclude by studying how the exploration vs. exploitation problem has been addressed in various fields.

2.1 Marine Vehicle Navigation and Control

The planning and control task for marine vehicles can be divided into two problems: the navigation problem of finding a route through obstacles, and the control problem of keeping the vehicle on that route. The navigation problem is generally handled by defining as series of waypoints on the way to the goal location, and there are several different methods of low-level control. These different tasks are described below.

2.1.1 Waypoint-Based Navigation

The standard approach to marine vehicle navigation is waypoint-based path following [27]. A waypoint is a point in space: a point on the ocean surface for surface vessels, or a point in the ocean volume for underwater vehicles. Waypoints are positioned between the vehicle's starting point and its destination so that the passage between the waypoints is safe and efficient. The navigation task when using waypoints is simply to drive from one waypoint to the next, in series, until arriving at the destination. In the simplest form of waypoint navigation, the vehicle simply steers so that its heading points toward the target waypoint (Figure 2-1). When the vehicle approaches within a certain distance of the target waypoint, that waypoint is abandoned and the vehicle seeks the next waypoint.

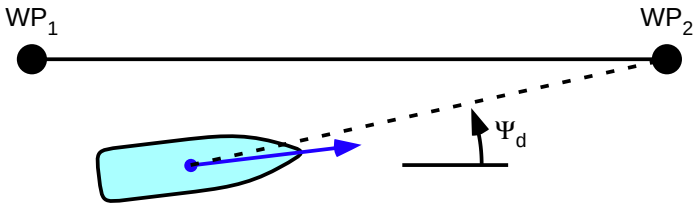

Figure 2-1: Simplest waypoint navigation: the vehicle points to the next waypoint.

An analysis of the sensitivity of the approach described above reveals that, for a fixed heading controller, the closed-loop dynamics change as the vehicle approaches the waypoint [36]. This is an undesirable feature, so the waypoint path-following algorithm is usually modified as follows. First, a line is defined connecting the current waypoint to the last waypoint. Next, the vehicle location is projected onto the line. A target is created a certain distance Δ ahead of the projected vehicle position on the line, and the heading controller is commanded to point the vehicle toward that new point. This procedure, generally known as the line-of-sight (LOS) algorithm [28], is illustrated in Figure 2-2. The look-ahead distance Δ is typically chosen as 2 or 3 boatlengths [65]. Because it is constant, the linearized closed-loop dynamics of the vehicle around the straight path leg do not change with the distance from the waypoint. Furthermore, it has been proven [65] that, when moving forward at

a constant speed in a vehicle with pure yaw control with the LOS algorithm, any exponentially stable heading controller will cause the vehicle to converge to the path leg with global asymptotic stability. This powerful result explains the widespread use of LOS waypoint algorithm. However, the pure yaw control assumption excludes most surface vessels whose dynamics are non-minimum phase.

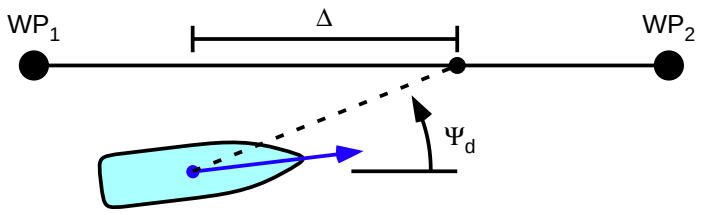

Figure 2-2: Line-of-sight waypoint navigation: the vehicle points to a target on the path leg.

If the heading controller is linear and the deviations from the path are small, the LOS algorithm represents a linear cross-track position feedback controller, and linear control design techniques can be used to tune the system by changing the heading controller gains or Δ.

2.1.2 Low-Level Control

Low-level control for a marine vehicle comprises velocity control, heading control, and position control. It does not consider any knowledge of obstacles or higher-level mission goals; a state vector $\mathbf{x}$, which may include velocities and/or positions, is regulated to match a reference state $\mathbf{r}$ which may or may not be known in advance. In velocity control, the position states are left unregulated, but the higher-level planner may be able to predict the resulting trajectory given assumptions about the low-level controller's rate of convergence and/or steady-state errors.

In some cases, each state in $\mathbf{x}$ is measured directly: this leads to full-state feedback control. In other cases, only some of the states are measured; for example, if a vehicle's only sensor is GPS, then position is measured directly but velocities must be inferred from changes in the measured position over time. In the latter case, either output

feedback must be used to control the system (unmeasured states do not appear in the control law) or a state estimator such as a Kalman filter must be coupled with a full-state feedback controller.

Low-level control can be achieved through many different methods. The simplest method is linear time-invariant (LTI) control, which is particularly suited for vehicles with linear dynamics, linearizable kinematics, and no saturation limits. The simplicity of the analysis of LTI systems means that these restrictive assumptions are often worthwhile. Proving stability for linear control of nonlinear systems, nonlinear control of linear systems, and nonlinear control of nonlinear systems (each resulting in a nonlinear closed-loop system) can be very difficult [78].

Proportional, integral and derivative (PID) control is the most ubiquitous control method used in industry because it is simple to implement, it can be applied to nonlinear systems, and it requires tuning only three gains (on the error signal, its integral, and its derivative). In many applications only PI or PD control is necessary, simplifying the tuning process further. However, general PID control does not have any stability or robustness guarantees and for complex systems even tuning three gains can be difficult. Self-tuning PID control [84] can help with the latter issue, but the lack of guaranteed stability and robustness means that PID control is unattractive for modern applications. Furthermore, PID control can only be used for single-input, single-output (SISO) systems.

The linear quadratic regulator (LQR) is a control method for LTI systems with guaranteed robustness. Unlike PID control, an LQR is designed from the system model, and it can be applied to multiple-input, multiple-output (MIMO) systems. LQR control is also known as optimal control, because it minimizes (optimizes) a cost function J that is a combination of state errors and control effort:

$$J = \int_{t=0}^{\infty} \mathbf{x}^T \mathbf{Q} \mathbf{x} + \boldsymbol{\tau}^T \mathbf{R} \boldsymbol{\tau} dt \tag{2.1}$$

In Equation (2.1), $\boldsymbol{\tau}$ is the control vector and $\mathbf{Q}$ and $\mathbf{R}$ are symmetric positive-definite state and control cost weighting matrices, respectively. The control gain

matrix $\mathbf{K}$ in the control law $\boldsymbol{\tau} = -\mathbf{K}\mathbf{x}$ is designed from $\mathbf{Q}$, $\mathbf{R}$, and the system model. The elements of $\mathbf{Q}$ and $\mathbf{R}$ are the tunable parameters for the controller. If $\mathbf{x}$ and $\boldsymbol{\tau}$ are properly scaled, then the cost matrices can often be simplified to $\mathbf{Q} = \mathbf{I}$ and $\mathbf{R} = \rho\,\mathbf{I}$ where ρ is the single tunable parameter that defines the bandwidth of the closed-loop system. Because robust MIMO control systems can be designed using a single tuning parameter, LQR control is very common; it is even used to control nonlinear systems.

Vehicle kinematics can be linearized around the straight paths between waypoints and folded into the LTI model for the vehicle plant. Consequently, line-of-sight waypoint-based path following (Figure 2-2) can be accomplished with an LQR control law.

Feedback linearization and sliding mode control can be used to control nonlinear systems [78]. Both methods cancel out the known nonlinearities in the system and replace them with stable linear dynamics. As such, the methods work best when the nonlinearities in the system are known exactly. Sliding mode control uses a smoothed switching control law to achieve some robustness to modeling error. Nonlinear backstepping is another approach that derives a control law from a Lyapunov function, resulting in guaranteed stability as long as the system parameters are known. Sliding mode control [1, 38] and nonlinear backstepping [66, 28, 21] have been applied to the low-level control of marine vehicles.

Another nonlinear control technique is fuzzy control, in which the measured state is mapped to a set of discrete sets ("large positive", "small negative", etc.) through one interpolation and the corresponding rules ("large rudder right","zero action", etc.) are mapped to the control values through another interpolation. One advantage of fuzzy control is that the rules can be designed by a human expert. The fuzzy rules can also be learned from human pilot data [58], yet fuzzy control is not significantly better than PID control [14].

Finally, neural networks have proven to be very effective for autonomous vehicle control [53, 62, 9] including ship berthing [20]. The weights in the neural network must be learned from training data, so they require a high-fidelity simulation of the

system or a rich batch of experimental data.

2.2 Learning

Some of the low-level control techniques presented in the previous section depend on a mathematical model of the system: LQR, feedback linearization, sliding mode control, and nonlinear back stepping all require a system model. The others do not: PID control, fuzzy control, and neural networks are tuned without a system model.

A system model can be determined offline from theory (knowing the mass, damping and geometry of all the components, for example, and calculating the system equations from the equations of motion) or empirically with separate model identification techniques. If the system changes, then the theoretical or empirical system model must be recalculated. The alternative approach is to learn the system model online while the system is running; when the system changes, the learning algorithm causes the model to change accordingly. For the control techniques that require a system model, the controller is redesigned online as the system model changes.

For the control techniques that do not require a system model, the controller must be designed from data that is either presented to a learning algorithm offline or collected online. When the system changes, the learning algorithm must be run again.

2.2.1 Adaptive Control

Adaptive control can either refer to learning the system model from which the controller is designed, or (for the controllers that do not use a system model) learning the control parameters or neural network weights. In either case, the parameters are updated online in response to tracking errors of the control system. This approach can be applied to a PID controller [84], neural networks [53, 62, 9], and many other systems [85, 77]. In adaptive control, the system can only learn parameters or control values associated with actions and behaviors that the system has performed. If a vehicle only drives in a straight line then adaptation will occur for the straight-line

parameters but not for the turning parameters, as those parameters have not yet been excited. This property may be undesirable if the goal is to learn all of the parameters in the system. Active learning, described below, is a strategy to use adaptive control to learn all of the parameters by designing trajectories that excite all of the modes of the system.

2.2.2 Least Squares System Identification

System identification is a general term referring to any method to learn about the structure of a system or the values of the parameters within that structure using measured data. It is related to adaptive control, except that the only aim is to learn about the system properties instead of learning how to control the system. Often a system model is needed for purposes besides control, such as prediction or planning. Once the system has been identified, the low-level control methods that require a system model (LQR, sliding mode control, nonlinear backstepping, etc.) can be used.

Least squares (LS) system identification is a linear regression technique for learning the values of parameters in a known model structure. If an output of the system y is a linear combination of the n states and inputs $\boldsymbol{\phi}$, then the output can be represented as an n-dimensional hyperplane described by the following equation.

$$y = \boldsymbol{\beta}^T \boldsymbol{\phi} \tag{2.2}$$

We acknowledge that the output measurement y may be corrupted by some sensor noise. If a finite number of input and output samples are collected then the LS algorithm finds the hyperplane such that the sum of the squared error between the hyperplane and each output sample is minimized. The parameter vector that satisfies this condition is shown below,

$$\begin{aligned} \hat{\boldsymbol{\beta}} &= \arg\min_{\boldsymbol{\beta} \in \mathbb{R}^n} \sum_{i=1}^{N} \left(\boldsymbol{\beta}^T \boldsymbol{\phi}_i - y_i \right)^2 \\ &= \left(\boldsymbol{\Phi} \boldsymbol{\Phi}^T \right)^{-1} \boldsymbol{\Phi} \mathbf{Y}^T \end{aligned} \tag{2.3}$$

where $\mathbf{\Phi} = [\boldsymbol{\phi}_1 \dots \boldsymbol{\phi}_N]$ and $\mathbf{Y} = [y_1 \dots y_N]$. The model structure appears in the mapping from the states and control inputs to the learning input vector $\boldsymbol{\phi}$. This mapping can be nonlinear; equation (2.2) is still valid as long as the elements of $\boldsymbol{\phi}$ combine linearly. For a MIMO system, (2.3) is evaluated for each output.

If $\mathbf{\Phi}\mathbf{\Phi}^T$ is ill-conditioned then the parameter estimates will be poor. This happens when the number of data points is less than the size of $\boldsymbol{\beta}$ or not all of the modes of the system have been excited. In these cases, the parameter estimates may need to be regularized [22]. Regularization refers to using *a priori* estimates of the system parameters to initially guide the estimates to the correct values and prevent numerical instability.

The LS algorithm presented above requires computing $\left(\mathbf{\Phi}\mathbf{\Phi}^T\right)^{-1}$ for the entire data set; this may be computationally expensive if it is recomputed whenever a new data point appears in online learning. The Recursive Least Squares (RLS) algorithm produces the same estimate $\hat{\boldsymbol{\beta}}$ but it involves a smaller incremental calculation when each new data point is added. The RLS algorithm can be written with a forgetting factor so that recent data is given more weight than old data; this is a useful feature when the system is changing through time. Several methods exist for adjusting the forgetting factor online to foster fast convergence to a time-varying system without instability [26, 15, 63, 83, 87].

2.2.3 Active Learning

Adaptive control and least squares estimation are both sensitive to the richness of the data provided to them. Adaptive control techniques will only learn the control gains that are used by the system to generate the data, and the least squares parameter estimates only converge if the associated mode has been excited. While it is not necessary to learn all of the control gains or all of the system parameters at the same time, it is often beneficial to learn a model for the entire system as quickly as possible. The data used to learn the parameters in adaptive control and least squares estimation is dependent on the open-loop commands applied to the system or the reference trajectory that the system is asked to follow. In active learning, these

commands are generated online to probe the system so as to learn the parameters as efficiently as possible.

An excellent example of active learning for system identification is found in [69]. It has been applied to the design of an underactuated cart-pole stabilization problem [74] and robot juggling [73]. A related topic, active probing, is used to diagnose problems in distributed network systems [11, 71]. Finally, active learning has been used to learn reward distributions from different actions, similar to multi-armed bandit problems (described in a later section). In fact, many exploration vs. exploitation problems exhibit active learning.

2.3 Motion Planning

The full integrated algorithm presented in this thesis is based around a robust motion planner. This planner generates motion plans that have a low probability of collisions with obstacles in the presence of external disturbances and modeling uncertainty. This section describes the Maneuver Automaton motion planning framework in which the motion plans are built, and the search algorithm that is used to find the optimal plan within the framework.

2.3.1 Requirements

The general requirement of a motion planner is that it must find a feasible sequence of actions that moves a mobile robot from an initial configuration to a goal configuration while avoiding contact with obstacles along the way. The planner may also take the robot's dynamic constraints into account. Ideally the motion planner finds the plan that optimizes some objective function such as plan duration, energy expended, etc. If the motion planning framework discretizes the plans in some way then the optimal plan can be found through a graph search, of which there are many techniques. Otherwise, a continuous optimization method such as mixed integer linear programming must be used.

2.3.2 POMDPs

A common framework for planning in stochastic environments is a partially observable Markov decision process (POMDP) [45]. In this framework, the state space is discretized into the finite set S, actions are discretized into the finite set A, and all of the possible observations (measurements) that the robot might encounter are discretized into the finite set Ω. There is a state transition function T that maps all possible combinations of states and actions to a probability distribution over the set of states; in other words, it defines the probability that the robot will end up in each state given an initial state and an action. Similarly, there is an observation function O that maps all possible combinations of states and actions to a probability distribution over the set of observations; in other words, it defines the probability that the robot will measure each possible observation given that it is in a particular state after executing a particular action. Finally, the reward function R defines the reward (or alternatively the cost) that would be gained by taking each action in each state. The POMDP is subject to the Markov property, which means each state and reward is a only a function of the previous state and action, but not any states or actions farther in the past. The goal is to maximize the reward (or minimize the cost) over a finite or infinite time horizon. Solving a POMDP refers to determining the policy (action as a function each state) that maximizes the reward. The policy may be time-dependent. POMDPs are typically solved with value iteration [72, 82].

The disadvantage of using POMDPs for motion planning is that the discretizations of the states, actions and observations would either be too coarse for practical control implementations or too fine to be solved in a computationally efficient manner. Furthermore, the Markov property is violated for a mobile robot with real dynamics, unless several layers of previous poses are included in the set of discrete states. As an alternative to the POMDP framework, we use the Maneuver Automaton framework, which is described next.

2.3.3 Maneuver Automaton Framework

Frazzoli's Maneuver Automaton (MA) framework [29] is a general method for representing kinodynamic plans built from a discrete set of maneuvers. Maneuvers comprise *trims* or *trim trajectories*, which are periods of constant velocity, and *motion primitives* which are used to connect the various trim states in the state space in a finite amount of time. A simple MA system is shown in Figure 2-3. Motion plans are concatenations of maneuvers, including trims of arbitrary duration and fixed-duration motion primitives.

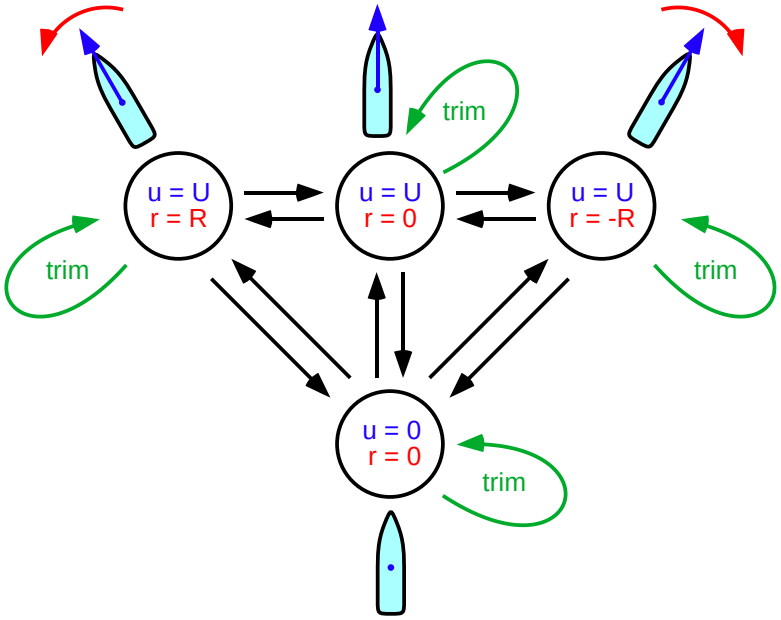

Figure 2-3: A simple maneuver automaton system. The vehicle can hover ($u = 0$, $r = 0$), drive straight forward at a constant speed U, or drive forward and turn at a constant yaw rate R. Each of those behaviors is a trim. A trim trajectory is a maneuver that stays in a trim, indicated with a green arrow, while motion primitives are maneuvers that move between trims, shown as black arrows.

The MA framework has been applied to air vehicles [29, 76, 75, 33] and marine vehicles [30, 81, 37]. Both of these types of vehicles represent holonomic systems, for which the MA framework is particularly attractive; however, this framework can be applied to non-holonomic systems such as wheeled ground vehicles as well.

If there are no obstacles in the configuration space of the vehicle, it is possible to

prove which regions of the space can be reached by a given set of trims and motion primitives [31]. It is almost certain that there are multiple possible motion plans that can be generated within the MA framework to get to any particular point in the space. Obstacles limit the possible endpoints of the plans (a plan cannot end in an obstacle or in a region of the space entirely cut off from the vehicle by obstacles) and the number of plans that can be used to get to other points in the space. Yet even with obstacles and a finite set of trims, there may be an infinite number of feasible plans to get to a goal point in the free space. While the MA framework can be used to represent plans, it does not offer any mechanism for finding plans, let alone optimal plans (by any metric of optimality).

Motion plans are described by two lists: the sequence of maneuvers that make up the plan, and the list of durations for each maneuver. Motion primitives often have fixed durations; the time it takes to go from one trim to the next can be evaluated experimentally or in simulations. Trim trajectories, on the other hand, can have arbitrary durations. The list of maneuver durations in a plan includes fixed values for the motion primitives and variable trim durations. The planning task is twofold: (a) find the optimal sequence of maneuvers, and (b) find the optimal values of the different variable-length trims within that sequence.

Within a given maneuver sequence, if the only variable-length trims are straight trims ($r = 0$ in Figure 2-3) and there are no obstacles, then the trim-length problem becomes a linear programming (LP) problem. (Turning trims add nonlinearities due to the rotating coordinate frame, making the trim-length problem a nonlinear programming problem). Therefore one solution to the motion planning problem is to concatenate maneuvers together one by one in all combinations, and then solve the LP problem for each maneuver sequence [29]. As more time is devoted to the problem, the number of maneuvers increases and the possibility of finding a lower-cost solution increases. This approach only works if the number of maneuvers is small and there are no obstacles (meaning the probability that any particular maneuver sequence *can* end at the goal is high).

This approach can be extended to include obstacles that are represented as poly-

gons; optimal plans can be found using dynamic programming to generate the sequences and solving a Mixed Integer Linear Programming (MILP) problem for each sequence [76, 75]. The MILP is necessary to encode the obstacle constraints.

The various motion plans represented by the MA framework form a tree structure in the state space; as different maneuvers are concatenated to an existing plan, the new plans branch out in different directions according to the configuration change resulting from the maneuver. Branches of the tree that intersect with obstacles or violate other constraints are deleted. A feasible motion plan is a path through the tree that ends at the goal point. Motion plans have been generated using the MA framework with randomized tree search algorithms [32, 81] and deterministic search algorithms [37]. As trees are a subset of graphs, we now consider various graphs search algorithms that could be used to find feasible and optimal motion plans.

2.3.4 Graph Search Algorithms

A network graph is a collection of *nodes* (points in the state space) that are connected to each other by *edges* (state transitions). Usually each node is only directly connected to a small number of neighboring nodes. Nodes can have a cost or a reward associated with being at the node; similarly, traversing an edge is usually associated with a cost or reward. A typical problem involving graphs is to find the path through the graph from a start node to a goal node that incurs the smallest cost or reaps the largest reward. Graph search algorithms find these optimal paths. Because nodes and edges can represent very abstract concepts, graph search algorithms can be applied to many different problems in robotics and planning [6]. The two most common graph search algorithms are Dijkstra's Algorithm and A*.

Dijkstra's Algorithm

Dijkstra's algorithm was first introduced in 1959 [18]. For a graph with known non-negative edge costs, the algorithm computes the minimum cost from every node to a goal node. The algorithm first initializes each node with an infinite (or very large)

cost. Then the goal node is assigned a cost of zero and added to the queue. Next the algorithm spreads outward from the goal, updating the minimum cost of each node. When a node cost is updated, the node is added to the queue so that its neighboring nodes can be updated as well. The complete algorithm is presented in Algorithm 1.

Algorithm 1 Dijkstra's Algorithm.

1: Create a network graph $\mathcal{G}$ consisting of nodes $\mathcal{N}$, edges $\mathcal{E}$, and edge costs $c(e),\ \forall e \in \mathcal{E}$.
2: Initialize every node with an infinite cost: $\forall n \in \mathcal{N},\ c(n) = \infty$.
3: Set the cost of the goal node to zero and add it to the queue $\mathcal{Q}$.
4: **while** $|\mathcal{Q}| > 0$ **do**
5: Remove the first (lowest-cost) node from the queue: $n \leftarrow \mathcal{Q}(0)$.
6: **for** each edge e entering node n **do**
7: Compare the cost of the connecting node m to the cost of n plus the cost of e, and assign the smaller value to m: $c(m) \leftarrow \min(c(m),\ c(n) + c(e))$.
8: If $c(m)$ has been updated, add node m to the queue, sorted by cost.
9: **end for**
10: **end while**

Once the minimum costs have been assigned using Algorithm 1, the minimum-cost path from any node to the goal can be found by simply moving to the neighboring node with the minimum cost until arriving at the goal node. If the graph represents a physical space through which a mobile robot moves, then Dijkstra's algorithm provides a robust method to get to the goal: if the robot is pushed off the path due to external disturbances or control errors, then the new optimal path to the goal is easily computed by following the minima from the new node.

If a node's cost is infinity after running Dijkstra's algorithm, then there is no path from that node to the goal. A similar algorithm is the Bellman-Ford algorithm [5]. It can handle negative edge costs, as long as those edges do not form a negative-valued loop, but it has a longer runtime than Dijkstra's algorithm.

A* Algorithm

While Dijkstra's algorithm finds the optimal path from all nodes to a goal node by working backwards, it is often more desirable to compute the optimal path from a single node to the goal node. Dijkstra's algorithm would of course work in this case,

but it would involve evaluating far more nodes than are necessary for the task. A* is an algorithm introduced in 1968 [43] to perform an optimal forward search to find a path to the goal. It is a best-first algorithm, which expands nodes in order from the "best" to the worst.

There are three costs involved in the A* algorithm. For each node n there is the cost accrued from the start of the path to the node: this is $g(n)$. A heuristic function h is used to estimate the cost-to-go from n to the goal, $h(n)$. The estimated overall cost of the plan f is the known cost from the start to n plus the estimated cost from n to the goal: $f(n) = g(n) + h(n)$. The steps of A* are listed in Algorithm 2.

Algorithm 2 A* Algorithm.

1: Create a network graph $\mathcal{G}$ consisting of nodes $\mathcal{N}$, edges $\mathcal{E}$, and edge costs $c(e)$, $\forall e \in \mathcal{E}$.
2: Add the initial node n to the queue $\mathcal{Q}$.
3: **while** $|\mathcal{Q}| > 0$ **do**
4: Remove the first (lowest-cost) node from the queue: $n \leftarrow \mathcal{Q}(0)$.
5: **if** n is the goal **then**
6: **return** with success.
7: **end if**
8: **for** each edge e leaving node n **do**
9: The cost of the path up to the connecting node m is $g(m) = g(n) + c(e)$.
10: Compute the estimated cost-to-go $h(m)$.
11: The cost of node m is $f(m) = g(m) + h(m)$.
12: Insert m into $\mathcal{Q}$ sorted by f.
13: **end for**
14: **end while**
15: **return** with failure.

If A* returns with failure, then no path exists from the start node to the goal. Otherwise it is guaranteed to return the optimal path. It should be noted that the algorithm does not terminate when a path ending at the goal is placed onto the queue, but rather when it is removed from the queue; otherwise it cannot be guaranteed that the path is optimal.

The optimality of A* depends on the heuristic function h. If h never overestimates the cost to the goal, then A* is guaranteed to return the lowest-cost path from the start to the goal. The speed of the algorithm (the number of node expansions)

depends on the accuracy of h; if h exactly equals the cost-to-go, then the algorithm will only expand the nodes along the optimal solution path. If the heuristic is poor or nonexistent ($h = 0$) then A* will return the optimal solution but it may expand many unnecessary nodes. In fact, in the limiting case $h = 0$, A* degenerates to a forward version of Dijkstra's algorithm. If the heuristic overestimates the cost-to-go then it is an inadmissible heuristic; the planner may return a solution more quickly than with an admissible heuristic, but the solution is not guaranteed to be optimal. A detailed discussion on the optimality of A* is found in [17].

Extensions to A*

If the network graph used by Dijkstra's algorithm or A* changes (either the connections between the nodes change, or the edge costs change) then those algorithms do not have facilities to find the new optimal path without starting the algorithm from scratch. A* was extended to changing graphs in 1994 [79] with the introduction of Dynamic A*, or D*. D* was rewritten and combined with an incremental version of A* in 2002 to become D* Lite [47]. In D* Lite, when edge costs are found to have changed during the execution of the plan, the changes are propagated downstream to find a new optimal path given the known cost change without rerunning A*.

Anytime Repairing A* (ARA*) [56] uses the property that inadmissible heuristics (a function h which overestimates the cost-to-go) may return a solution faster than an admissible heuristic would, although the solution may be sub-optimal. If the planning time is limited, then a sub-optimal solution may be more desirable than no solution at all. ARA* runs A* with an inflated heuristic (the admissible heuristic multiplied by an inflation factor $\alpha \geq 1$ to generate a solution quickly. If some planning time remains, the algorithm then reduces the inflation factor and runs the algorithm again. This process continues until planning time runs out. If the inflation factor has reduced to $\alpha = 1$, then the solution is optimal; otherwise it is sub-optimal.

Physical A* (PHA*) adds a twist to the planning problem; rather than exploring the state space in the planning process, the space is physically explored by a mobile robot to determine edge costs and node feasibility [25].

2.3.5 Kinodynamic Planning

The planning algorithms described above can be used to find the minimum-cost path through a network graph. For basic path planning, the graph represents points in the physical space in which the vehicle is moving, such as grid points or features on a map. However, graphs can represent much more abstract concepts [6]. Each node is a discrete state of the system, and edges are simply the transitions between those discrete states. The states and transitions may be as specific or general as necessary. Similarly, obstacles in the physical environment translate to general state constraints in the abstract environment. These constraints may be time-dependent.

Vehicles are also subject to equations of motion, which serve as dynamic and kinematic constraints. A path generated in a physical environment may not be realizable by the vehicle. The vehicle's turning radius and stopping distance prevent vehicles from following paths with sharp corners. Wheeled vehicles have additional nonholonomic constraints. In these situations, state transitions are *maneuvers* that the vehicle is known to be able to perform. The maneuvers themselves are designed to obey control input constraints, velocity constraints, and other dynamic and nonholonomic constraints. If all the edges in the graph are admissible maneuvers, and none of the maneuvers collide with physical obstacles, then every path through the graph can be executed by the vehicle. The motion plan is a concatenation of maneuvers. Using maneuvers to construct a network graph, and planning within that graph, is known as *kinodynamic planning*.

It is not necessary to generate the entire graph before finding the plan. Using a best-first search algorithm such as A*, not all of the nodes of the graph need to be explored. Furthermore, the maneuvers can often be added in an infinite number of combinations, leading to infinite graphs. Expanding search algorithms build the graphs as necessary during the search process.

2.3.6 Expanding Search Algorithms

Graph search algorithms can find paths through existing graphs whose connections and edge costs are already known. Simply constructing these graphs can be an enormous computational effort, especially when planning in higher dimensions. The planning effort then involves the comparable computational costs of generating a collision-free graph that connects the start configuration to the goal, and finding the best path through that graph. In these situations merely finding *a path* is considered as success, to say nothing of *the optimal path*.

In high-dimension planning spaces, such as those describing robotic arms with many degrees of freedom, the "obstacles" may be physical obstacles, kinematics constraints, or (for state spaces involving velocities) dynamic constraints. The effect of these obstacles is that a path to the goal configuration may require significant explorations of the state space. Furthermore, constructing heuristics for this space is very difficult.

An expanding search algorithm adds actions or maneuvers to a tree-shaped graph. The algorithm selects a node of the graph to expand using one or several of the actions available to the vehicle. The new actions are added to the tree and a new node is selected for expansion. An example of an expanding search is shown in Figure 2-4.

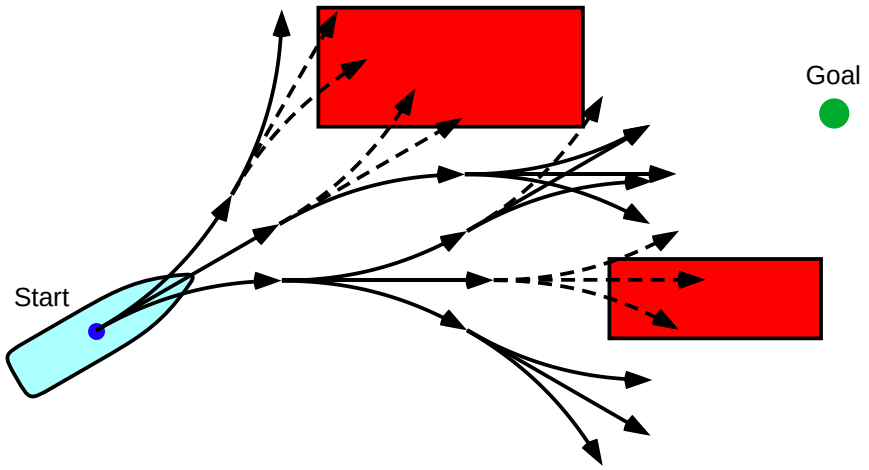

Figure 2-4: An expanding search algorithm adds maneuvers to a tree in order to find the goal. Maneuvers that collide with obstacles (dashed) are discarded.

Rapidly-Exploring Random Trees

The motivations listed above led to the introduction of Rapidly-Exploring Random Trees (RRTs) in 1998 [49]. The algorithm expands a tree by picking a random point in the state space, choosing the nearest node of the tree to that point, and extending the tree from the node in the direction of the selected point using an admissible maneuver or control input. If the new edge crosses an obstacle, then it is discarded; otherwise the algorithm continues. The general form of the algorithm is shown in Algorithm 3.

Algorithm 3 RRT Algorithm.

1: Initialize a network graph $\mathcal{G}$ with a node representing the start configuration.
2: Define g, the goal region of the state space $\mathcal{X}$.
3: **while** $\forall n \in \mathcal{G},\ n \notin g$ **do**
4: Choose a random point x in $\mathcal{X}$.
5: Find the node n in $\mathcal{G}$ that is closest to x.
6: Add an edge e from n toward x, terminating at a new node m.
7: Check if e crosses an obstacle in the state space; if it does, discard e and m.
8: **end while**

The RRT algorithm expands outward from the start node, with the nodes of the graph approaching the distribution of the random samples. In particular, the graph on average breaks up the large Voronoi regions of the free space; this means that the algorithm has excellent exploration properties. The RRT algorithm can take many flavors depending on the methods used in Algorithm 3. The probability density function for the random samples can be skewed toward the goal node to encourage the tree to expand there [49]; alternatively, a small fraction of the time the goal node can be used for x in step 4 instead of a random point. Both methods are forms of goal-biasing. The distance metric used in step 5 may be the Euclidian distance or any other metric. Finally, there are many possible methods that could be used in step 6 to add a new node to the tree. (i) An inverse problem could be solved to find the set of inputs that drive the state of the system in the direction of x. (ii) A discrete set of inputs could be compared to see which input drives the system closest to x. (iii) A feedback control law can be constructed to move the system toward x for a finite amount of time.

A similar approach combines a probabilistic roadmap (PRM) with predictive models (the edges in the RRT) to explore the configuration space [13]. However, neither method uses the plan cost in a meaningful way; different collision-free paths in the configuration space may both arrive at the goal with vastly different costs, and an optimal motion planner must find the path that arrives with the guaranteed lowest cost. Fortunately, the A* algorithm can be modified to expand into the configuration space like the RRT algorithm. This technique is described below.

Expanding A*

The standard A* algorithm is used to find the shortest path through a known graph. However, for kinodynamic planning the problem is to find the optimal path through a possibly infinite tree-shaped network like the one shown in Figure 2-4. By combining the heuristic of Algorithm 2 with the plan concatenation of Algorithm 3, we can create a search algorithm that finds the lowest-cost plan that ends at the goal. The robust motion planning algorithm used in this thesis is based on Algorithm 4.

Algorithm 4 Expanding A* Algorithm.

Load a maneuver library $\mathcal{M}$.
Initialize the search queue $\mathcal{Q}$ with a plan containing only the initial node.
while $|\mathcal{Q}| > 0$ **do**
  Remove the first plan from the queue: $p \leftarrow \mathcal{Q}(0)$.
  if p ends at the goal **then**
    return p with success.
  end if
  for each maneuver m in $\mathcal{M}$ **do**
    Add m to p: $p' \leftarrow p + m$.
    if p' is collision-free **then**
      Compute the cost $g(p')$.
      Compute the predicted cost-to-go $h(p')$.
      The predicted total cost of p' is $f(p') = g(p') + h(p')$.
      Insert p' into $\mathcal{Q}$ sorted by $f(p')$.
    end if
  end for
end while
return with failure.

These planning algorithms can either apply to a deterministic system or a stochas-

tic system, as the actions or maneuvers can be direct low-level signals ("turn the left wheel through three revolutions") or high-level policies ("apply a feedback law until the desired terminal condition is achieved"). It is important for a robust planner to know how the execution of each action will contribute to the plan cost, as the cost may be higher than expected if execution errors arise. The next section how the planner can handle uncertainty in the system or the environment.

2.4 Planning Under Uncertainty

The planning community has recently shifted from planning for deterministic systems in known environments to planning for stochastic systems in unknown environments. This section outlines several existing views on the problem.

2.4.1 Various Sources of Uncertainty

LaValle and Sharma (1998) list four different types of uncertainty that may be present in motion planning problems: robot sensing uncertainty (RS), robot predictability uncertainty (RP), environment sensing uncertainty (ES), and environment predictability (EP) [52].

RS uncertainty means that the robot does not exactly know its position and configuration within the environment; this may be the result of sensor noise. Similarly ES uncertainty means that the robot does not exactly know about the environment around it, which can also result from incomplete sensing or sensor noise. Simultaneous Localization and Mapping (SLAM) seeks to reduce RS and ES uncertainty.

RP uncertainty means that the robot cannot accurately predict its future configurations given a known sequence of control actions. This can arise from an imperfect system model or external disturbances. If the uncertainty is bounded then a backprojection algorithm can be used to identify plans that will be guaranteed to succeed [23]. The backprojection is also known as a preimage; it has been extended to encompass uncertainty probability density functions [51]. If environmental sensing is good and the robot's sensors are more reliable when near obstacles (as with most laser and

acoustic positioning systems), then the planner will choose paths that hug the walls of obstacles to reduce the position uncertainty on the way to the goal [4, 72]. The Maneuver Automaton framework has been extended to include maneuver uncertainties by simply adding a cost associated with each maneuver's uncertainty to the cost function [76]. This author's work uses the Maneuver Automaton framework with an analytic error variance prediction to explicitly include the risk of collisions in the cost function [37].

EP uncertainty arises in dynamic environments: the future state of the environment, including other robotic or human agents, may not be known exactly. A* has been extended to try to handle this case [86], and an evolutionary algorithm has been created for a deterministic aerial robot in which collision probabilities depend on the time-varying probability density functions for the obstacles [68].

2.4.2 SLAM

An important problem in mobile robotics is to build a map of an unknown environment without the use of a global position reference. Lunar and subsea navigation fall into this category because in both cases it is unlikely that a ground-truth position reference like GPS is available. To build a map with body-fixed sensors, the vehicle must have a good knowledge of its own position, but with out an accurate global reference it must infer its position from the features it is mapping. This chicken-and-the-egg problem is called Simultaneous Localization and Mapping (SLAM) [54, 19]. In the most basic form, the vehicle follows a pre-planned trajectory and uses SLAM to build a map and find its position within that map. However, it is possible to design trajectories that actively reduce either the uncertainty in the map [48] or the uncertainty of the robot's position through a one-step look-ahead [24] or a multi-step look-ahead [44].

2.4.3 Probabilistic Algorithms

All forms of uncertainty can be handled using probabilistic algorithms [82]. In particular, particle methods can be used for robot localization. The general idea behind particle methods is that the probability distribution for the robot's position within the configuration space is represented by a set of particles. New sensor data affects the belief distribution across the particles. The particles are redistributed in the space according to the updated probability distribution. Applying this approach to the mapping problem results in a huge increase in complexity. Furthermore, Kalman Filter approaches may not be effective because the distributions are often multimodal.

Particle methods can also be used to generate optimal control sequences in the presence of stochastic process disturbances [7]. Each control action throughout the time horizon is a free parameter. If the system dynamics and the constraints can be linearized, then the particle problem can be written as a Mixed Integer Linear Program (MILP). The result is a control sequence that can be guaranteed to result in a collision-free trajectory with a specified probability. The accuracy of this guarantee increases as the number of particles increases.

In this work we consider primarily RS and RP uncertainty; we assume that the robot has a complete and accurate map, and any changes to the environment are known in advance. This is a very limiting assumption in terms of real robotics, but it is necessary to limit the scope of this work.

It may be possible for the mobile robot to reduce its uncertainty through the actions it chooses to take. However, those actions may delay or add cost to the mission. This tradeoff is discussed in the next section.

2.5 Exploration vs. Exploitation

Throughout the planning world there are many examples of the "exploration vs. exploitation" problem. This problem is particularly relevant in reinforcement learning [50]. On the one hand, it is often beneficial for a robot to explore its environment (either the physical environment or the robot's state space) and learn about the risks

and rewards [80, 57]. This could be considered to be a form of intrinsic motivation [46]. On the other hand, if the robot is seeking to maximize an objective function then it makes sense to exploit the rewards that are already known. When faced with a new or uncertain environment, a robust strategy strikes a balance between exploration and exploitation.

Two examples in the literature apply this tradeoff to the path search problem. The exploring/exploiting tree (EET) algorithm explicitly balances exploration and exploitation. The algorithm is weighted more toward exploitation when the expanding tree makes successful connections, and it is weighted more toward exploration when it does not [70]. Physical A* (PHA*) is an example of a planning algorithm that balances exploration (in this case physically performing the search with a mobile robot to determine the connectivity and edge costs in the graph) with exploitation (the cost of moving within the physical graph) [25].

Reinforcement learning is a form of integrated planning and learning in which an agent moves through an unknown world learning about reward and cost distributions. Two examples are [80], which uses dynamic programming to find strategies to move through a grid-based world, and [57], which operates in a world with obstacles, enemies, and food.

More recently, path planning has been used to reduce map uncertainty [48]. In this case a high-level coverage planner chooses a set of destination points within the known space to optimally reduce the map accuracy. Next a reinforcement learning algorithm learns the best way to arrive at the destination point while most improving the map. This approach can be adapted to ensure a high probability of arriving at a destination by choosing to drive near known features [72, 67]. This process is known as *coastal navigation*, because it is similar to a ship sailing within sight of a coastline rather than taking the shortest path to a destination across open water.

However, these examples deal with learning about the environment. In this thesis we assume to have a perfect knowledge of the obstacles in the environment and we learn about the robot's own dynamic model instead. An active learning approach similar to the multi-armed bandit is used to learn the parameters of an economic

system in [88]. Another model-learning example is given in [74], in which a minimum-time cart-pole maneuver is learned through many trials using stochastic dynamic programming. As the system is exposed to more and more data through the trials and the confidence in the model improves, the controller chooses more aggressive maneuvers that span larger regions of the state space. This increase in aggressiveness with improving confidence is a feature that we hope to capture with the integrated motion planning and model learning algorithm described in this thesis. A similar result is obtained in [73], where the system slowly builds up data in the state space over time. A controller is designed using the data near the system's current state. Randomness (real and artificial) causes the system to gather data in a small region around the desired state. Once the confidence in that data is large enough, the desired state is shifted slightly towards the ultimate goal state and the procedure repeats. Again, this consideration of the model uncertainty when moving the setpoint is a feature we incorporate into our algorithm.

The most relevant work in the literature uses trajectory optimization to discriminate between multiple system models [8]. A system is assumed to have a finite number of failure modes, each with a known system model. A finite sequence of control values can be designed using sequential quadratic programming to discern which failure mode best describes the data collected from the system. The model discrimination can be performed while executing another mission, such as driving to a goal. The integrated motion planning and model learning algorithm presented in this thesis is similar in that the planning algorithm takes the predicted reduction of uncertainty into account in the planning process, but we allow for a continuous variation of the system model rather than being restricted to finite set of known models.

2.6 Summary

In this chapter we have outlined the existing research in adaptive marine vehicle control, motion planning algorithms, and planning algorithms that seek to reduce uncertainty in the environment or the dynamic model of a robot. With few exceptions

there has not been a focus on planning based on what knowledge will be acquired during the execution of the motion plan, and using such predictions to improve the robustness of the plans to external disturbances and modeling errors. In this thesis we develop an integrated motion planning and model learning algorithm that optimally balances active learning strategies (exploration of the state space) with goal-seeking behavior (exploitation). This algorithm is built around a robust motion planner, which is described in the next chapter.

Chapter 3

Robust Motion Planning

This chapter describes the robust motion planning algorithm that is the core of the thesis. Standard motion planning is fairly straightforward: using a simplifying framework, the planner finds a sequence of actions that moves the vehicle from a start configuration to a goal configuration without intersecting with obstacles along the way. A rapidly-exploring random tree (RRT) is a common example of a standard motion planner [49]. An optimal motion planner such as the A* algorithm uses a similar expanding tree, but optimal algorithms ensure that the returned path is the best (with respect to a cost function) among all motion plans in the planning framework. A robust motion planner must consider disturbances and other effects that may push the vehicle off the specified motion plan so that the returned plan has a good chance of succeeding even in the presence of those disturbances. In this chapter, we develop a robust motion planner that minimizes a combination of the plan duration and the probability of collisions with obstacles.

In the Section 3.1 we describe the requirements of the motion planner and the planning problems we expect to be able to solve. In the Section 3.2 we list the dynamic equations for a planar holonomic vehicle. The subsequent two sections describe the planning framework and the maneuvers that are used to construct the motion plans. In Section 3.5 we introduce the simple controller that is used to keep the vehicle on the reference path in the presence of stochastic disturbances and we compute predictions of the resulting error (Algorithm 5). Computing the collision probability

from the path-following error distribution is a non-trivial task that can be reduced to the *particle absorption* problem from statistical physics [12]. Section 3.7 deals with the analytic prediction of the absorption rate (Algorithm 6) and the corresponding collision probability (Algorithm 7). Section 3.8 describes the A* planning algorithm (Algorithm 8) that finds the optimal motion plan within the framework. The final sections present experimental results for the robust motion planning algorithm and summarize the innovations developed in this thesis.

3.1 Problem Definition

The motion planner is used to navigate a planar holonomic vehicle through a cluttered environment to arrive at a target location as efficiently as possible. The following pieces of information are required to specify the planning problem.

- Start position, orientation and velocity.
- Goal position.
- Obstacle positions.
- Nominal forward/backward velocity and yaw rate.

The start position is ideally the vehicle's position when it begins executing the plan, which may in practice be different from the vehicle's position when starting the planning process. In the interim the vehicle may be in motion due to the thrusters or external disturbances. Because the motion plan may be very sensitive to the start position, care should be taken to ensure that the vehicle is at the planned start position and orientation when it begins executing the plan.

The terminal state of any motion plan is a full state vector, but we may only want to use a subset of the terminal state when defining the goal. Using the full state (that is, specifying a goal position, orientation, speed and yaw rate) may be desirable, but it is problematic for three reasons. First, it may be impossible to independently specify three goal velocity components for an underactuated vehicle.

Second, specifying both a goal position and a goal velocity requires a backward search to ensure that the proper terminal states are reached. Third, it is very difficult to find a heuristic function that incorporates both the distance to the goal (position error) and the relative goal heading (orientation error). For example, for most motion planning problems for marine surface vehicles, heading errors relative to the goal heading should not be penalized at the beginning of the plan as the vehicle may need to steer around obstacles. To avoid these issues we specify only a goal position, leaving the terminal heading and velocities unconstrained. This simplification leads to hazardous conditions in planning scenarios such as pulling into a slip, as the velocity at the end of the plan may be large, or docking the vehicle, as the terminal orientation may be undesirable, but it is a necessary assumption to make the planning problem tractable.

The obstacle positions are assumed to be known exactly during the planning process. If the obstacles are moving then it is assumed that their positions can be accurately predicted throughout the duration of the motion plan. The obstacles can have any representation (points, polygons, curved objects, concave objects, etc.) as long as it is possible to test if a point is inside the obstacle and it is possible to compute the distance and angle to the closest point on the obstacle from any point in the space.

The final pieces of information needed to define the problem are the nominal forward speed and the nominal turn rate. These speeds define the library of maneuvers and trims that are used to generate the motion plans. Ideally these speeds are not the maximum speeds that the vehicle can achieve, because for combined position and velocity feedback it is useful to have some extra control authority when driving at the reference speed. Even for vehicles whose dynamic model is not well known, it is usually easy to define an expected cruise speed and a safely achievable yaw rate. Using these speeds one can define a 3×3 grid of points in the surge/yaw space. These points define the setpoints for the speed controller. A denser grid of setpoints would provide a better resolution for the planner, at the expense of added computational cost.

Figure 3-1 shows a typical planning problem using these pieces of information. The initial vehicle position and orientation are indicated in the figure. The goal position is shown as a green square. The obstacles are drawn in red. The black path is a motion plan using maneuvers built around the nominal forward speed and yaw rate. The motion plan shown in the figure is the optimal motion plan within the planning framework for a particular cost function that includes the estimated vehicle dynamics and process noise; this cost function is developed throughout this chapter. The next section describes the equations of motion for the vehicle, including the dynamics and the kinematics.

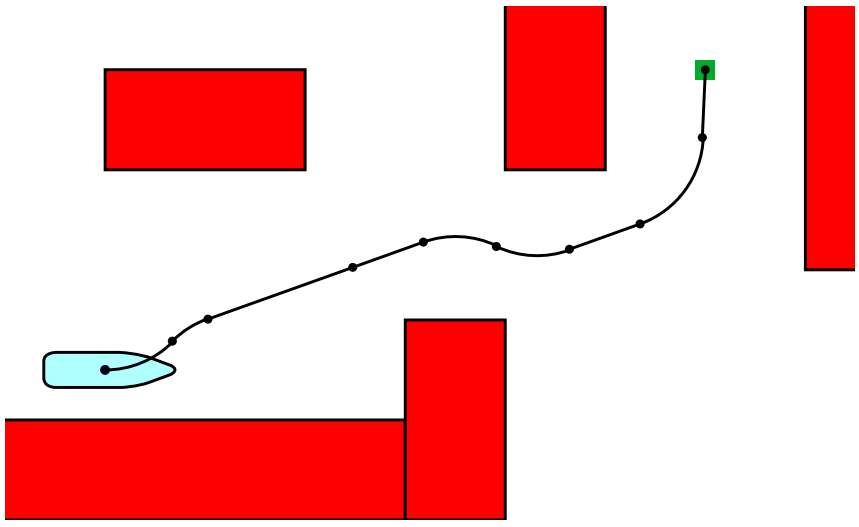

Figure 3-1: This motion plan brings the vehicle from the indicated start position and orientation to the goal position (the green square) while avoiding the obstacles (red regions). This plan is optimal given the vehicle dynamics and a particular metric based on collision probability.

3.2 Equations of Motion for a Planar Holonomic Vehicle

This section lists the equations of motion for a planar holonomic vehicle. These equations are the core of the various prediction algorithms used in this chapter and Chapter 4.

The (X, Y) coordinates of the center of mass of the vehicle and the vehicle's heading angle ψ are contained in the vector $\boldsymbol{\eta}$ (3.1). When $\boldsymbol{\eta} = \mathbf{0}$, the vehicle points along the X axis with the Y axis extending to the left (port) side of the vehicle (Figure 3-2). The Z axis points up from the center of mass of the vehicle.

$$\boldsymbol{\eta} = \begin{bmatrix} X \\ Y \\ \psi \end{bmatrix} \qquad \boldsymbol{\nu} = \begin{bmatrix} u \\ v \\ r \end{bmatrix} \qquad \boldsymbol{\tau} = \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_\psi \end{bmatrix} \tag{3.1}$$

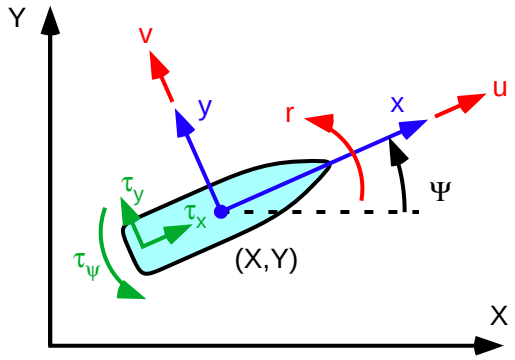

Figure 3-2: Global (X, Y) and vehicle-fixed local (x, y) coordinate systems. The velocities (u, v, r) and control inputs $(\tau_x, \tau_y, \tau_\psi)$ are shown as well.

A second coordinate system (x, y) is aligned with the vehicle and attached to its center of mass, as shown in Figure 3-2. The z axis points up from the center of mass. The surge velocity u, sway velocity v and yaw rate r are assembled in the velocity vector $\boldsymbol{\nu}$ (3.1). The control inputs are $\boldsymbol{\tau}$; these represent forces in the x and y directions, τ_x and τ_y, and a torque about the z axis, τ_ψ. In general in this thesis we consider an underactuated vehicle for which $\tau_\psi = 0$; this is valid for vehicles powered by a single azimuthing thruster. Fossen studies underactuated vehicles for which $\tau_y = 0$, which is the case when a vehicle is steered with differential thrust, such as many ROVs [28, 64]. The algorithms developed in this thesis are valid for both types of underactuated vehicles.

3.2.1 Dynamics

The dynamics for a marine vehicle, which can be adapted to any holonomic vehicle moving in a planar environment, can be expressed as follows [27]:

$$\mathbf{M}\dot{\boldsymbol{\nu}} + \mathbf{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{D}(\boldsymbol{\nu})\boldsymbol{\nu} = \boldsymbol{\tau} + \mathbf{w_T} \tag{3.2}$$

where the mass matrix $\mathbf{M}$, the Coriolis matrix $\mathbf{C}(\boldsymbol{\nu})$, and the drag matrix $\mathbf{D}(\boldsymbol{\nu})$ (up to the quadratic terms) are defined below and $\mathbf{w_T}$ is a vector of environmental disturbance forces and moments.

$$\begin{aligned} \mathbf{M} &= \begin{bmatrix} m_{11} & 0 & 0 \\ 0 & m_{22} & 0 \\ 0 & 0 & m_{33} \end{bmatrix} \qquad \mathbf{C}(\boldsymbol{\nu}) = \begin{bmatrix} 0 & 0 & -m_{22}v \\ 0 & 0 & m_{11}u \\ m_{22}v & -m_{11}u & 0 \end{bmatrix} \\ \mathbf{D}(\boldsymbol{\nu}) &= \begin{bmatrix} d_{11} + d_{u^2}|u| & 0 & 0 \\ 0 & d_{22} + d_{v^2}|v| & d_{23} + d_{vr}|v| \\ 0 & d_{32} + d_{rv}|r| & d_{33} + d_{r^2}|r| \end{bmatrix} \end{aligned} \tag{3.3}$$

Equation (3.2) can be rearranged into the following state space form,

$$\dot{\boldsymbol{\nu}} = \mathbf{a}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{b}\boldsymbol{\tau} + \mathbf{f}_{\boldsymbol{\nu}}(\psi) + \mathbf{w}_{\boldsymbol{\nu}} \tag{3.4}$$

where

$$\begin{aligned} \mathbf{a}(\boldsymbol{\nu}) &= \begin{bmatrix} -\frac{d_{11}}{m_{11}} - \frac{d_{u^2}}{m_{11}}|u| & 0 & \frac{m_{22}}{m_{11}}v \\ 0 & -\frac{d_{22}}{m_{22}} - \frac{d_{v^2}}{m_{22}}|v| & -\frac{m_{11}}{m_{22}}u - \frac{d_{23}}{m_{22}} - \frac{d_{vr}}{m_{22}}|v| \\ \frac{m_{11}-m_{22}}{m_{33}}v & -\frac{d_{32}}{m_{33}} - \frac{d_{rv}}{m_{33}}|r| & -\frac{d_{33}}{m_{33}} - \frac{d_{r^2}}{m_{33}}|r| \end{bmatrix} \\ \mathbf{b} &= \begin{bmatrix} \frac{1}{m_{11}} & 0 & 0 \\ 0 & \frac{1}{m_{22}} & 0 \\ 0 & 0 & \frac{1}{m_{33}} \end{bmatrix} \end{aligned} \tag{3.5}$$

The state transition matrix $\mathbf{a}(\boldsymbol{\nu})$ contains drag and Coriolis effects, and the input matrix $\mathbf{b}$ contains the gains for the control inputs $\boldsymbol{\tau}$. The mean environmental dis-

turbance in body-fixed coordinates is $\mathbf{f}_{\boldsymbol{\nu}}(\psi)$. The noise vector $\mathbf{w}_{\boldsymbol{\nu}}$ is a random vector sampled from a zero-mean multinormal distribution with a covariance matrix $\mathbf{W}_{\boldsymbol{\nu}}$; in other words, the total noise in body-fixed coordinates is $N(\mathbf{f}_{\boldsymbol{\nu}}(\psi), \mathbf{W}_{\boldsymbol{\nu}})$. For small velocities, $\mathbf{a}(\boldsymbol{\nu})$ can be linearized around $\boldsymbol{\nu} = \mathbf{0}$, as shown below. In general, $\mathbf{a}(\boldsymbol{\nu})$ can be linearized around any reference velocity $\boldsymbol{\nu}_{\mathbf{r}}$.

$$\mathbf{a}|_{\boldsymbol{\nu}=\mathbf{0}} = \begin{bmatrix} -\frac{d_{11}}{m_{11}} & 0 & 0 \\ 0 & -\frac{d_{22}}{m_{22}} & -\frac{d_{23}}{m_{22}} \\ 0 & -\frac{d_{32}}{m_{33}} & -\frac{d_{33}}{m_{33}} \end{bmatrix} \tag{3.6}$$

3.2.2 Kinematics

To derive the kinematics of a vehicle moving in the plane, we first consider a point in the local coordinate system $\mathbf{p}$ expressed in global coordinates $\mathbf{P}$ through the following transformation, where the origin of the local coordinate system is (X, Y) in $\mathbf{P}$ coordinates with an orientation ψ:

$$\begin{bmatrix} P_x \\ P_y \end{bmatrix} = \begin{bmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{bmatrix} \begin{bmatrix} p_x \\ p_y \end{bmatrix} + \begin{bmatrix} X \\ Y \end{bmatrix} \tag{3.7}$$

When a vehicle starting at the origin of the $\mathbf{P}$ coordinate system is moving at a velocity $\boldsymbol{\nu}$, the position of the vehicle after a time δt is $\boldsymbol{\eta}(\delta t)$, derived below using (3.7).

$$\begin{bmatrix} X(\delta t) \\ Y(\delta t) \\ \psi(\delta t) \end{bmatrix} = \begin{bmatrix} u\delta t\cos\psi - v\delta t\sin\psi \\ u\delta t\sin\psi + v\delta t\cos\psi \\ r\delta t \end{bmatrix} + \begin{bmatrix} X \\ Y \\ \psi \end{bmatrix} \tag{3.8}$$

Using the definition of the derivative, we have:

$$\begin{aligned} \dot{\boldsymbol{\eta}} &= \lim_{\delta t \to 0} \frac{\boldsymbol{\eta}(\delta t) - \boldsymbol{\eta}(0)}{\delta t} \\ &= \mathbf{J}(\psi)\boldsymbol{\nu} \end{aligned} \tag{3.9}$$

$$\text{where} \quad \mathbf{J}(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

The rate of change of $\mathbf{J}$ is derived below.

$$\begin{aligned} \dot{\mathbf{J}}(\psi) &= \begin{bmatrix} -r\sin\psi & -r\cos\psi & 0 \\ r\cos\psi & -r\sin\psi & 0 \\ 0 & 0 & 0 \end{bmatrix} \\ &= \mathbf{S}(r)\mathbf{J}(\psi) \end{aligned} \tag{3.11}$$

$$\text{where} \;\; \mathbf{S}(r) \equiv \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} r \equiv \mathbf{S}_r r \tag{3.12}$$

The matrices $\mathbf{S}(r)$ and $\mathbf{S}_r$ will be used in Section 3.6 to derive the path-following error dynamics.

3.2.3 Full System

The state space equations (3.4) and (3.9) can be combined into a single nonlinear state space equation. The full state $\mathbf{x}$ contains both the velocity vector $\boldsymbol{\nu}$ and the position/orientation vector $\boldsymbol{\eta}$.

$$\mathbf{x} = \begin{bmatrix} \boldsymbol{\nu} \\ \boldsymbol{\eta} \end{bmatrix} \tag{3.13}$$

The evolution of (3.13) is shown below.

$$\begin{aligned}
\begin{bmatrix} \dot{\boldsymbol{\nu}} \\ \dot{\boldsymbol{\eta}} \end{bmatrix} &= \underbrace{\begin{bmatrix} \mathbf{a}(\boldsymbol{\nu}) & \mathbf{0} \\ \mathbf{J}(\psi) & \mathbf{0} \end{bmatrix}}_{=\mathbf{A}(\mathbf{x})} \begin{bmatrix} \boldsymbol{\nu} \\ \boldsymbol{\eta} \end{bmatrix} + \underbrace{\begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix}}_{=\mathbf{B}} \boldsymbol{\tau} + \underbrace{\begin{bmatrix} \mathbf{f}_{\boldsymbol{\nu}}(\psi) \\ \mathbf{0} \end{bmatrix}}_{=\mathbf{F}_{\boldsymbol{\nu}}(\psi)} + \underbrace{\begin{bmatrix} \mathbf{w}_{\boldsymbol{\nu}} \\ \mathbf{0} \end{bmatrix}}_{=\mathbf{w}} \\
\dot{\mathbf{x}} &= \mathbf{A}(\mathbf{x})\mathbf{x} + \mathbf{B}\boldsymbol{\tau} + \mathbf{F}_{\boldsymbol{\nu}}(\psi) + \mathbf{w}
\end{aligned} \tag{3.14}$$

Equation (3.14) is a nonlinear state space system even when the velocities are small ($\mathbf{a}(\boldsymbol{\nu}) = \mathbf{a}$) due to the rotation matrix $\mathbf{J}(\psi)$ and the body-frame steady disturbance $\mathbf{f}_{\boldsymbol{\nu}}(\psi)$.

3.3 Motion Planning Framework

The motion planning framework used to construct and describe motion plans is based on Frazzoli's Maneuver Automaton (MA) [29]. The MA framework builds motion plans from two types of maneuvers: motion primitives and trims. We ignore this distinction and represent motion primitives and trims in the same way; we also add a third type of maneuver, driving to a waypoint, and represent it as a general maneuver as well. Maneuvers can be described mathematically using motion groups, which are a type of Lie group particularly useful for building motion plans. See [55] for a useful overview of Lie groups applied to motion plans. This section defines motion groups and studies their properties, and in the next section we use motion groups to build motion plans for planar vehicles.

3.3.1 Motion Groups

A convenient representation for the position and orientation of the vehicle is a *motion group*. The motion group $\mathbf{M}$ is a rotation matrix $\mathbf{J}(\psi)$ augmented with the global

position states. In 3 DOF, the motion group is:

$$\mathbf{M} = \begin{bmatrix} \cos\psi & -\sin\psi & X \\ \sin\psi & \cos\psi & Y \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.15}$$

Motion groups can also be used to represent maneuvers; these are known as *local motion groups*. The local motion group $\mathbf{m}$ contains the local position and orientation changes Δx, Δy and $\Delta\psi$.

$$\mathbf{m} = \begin{bmatrix} \cos\Delta\psi & -\sin\Delta\psi & \Delta x \\ \sin\Delta\psi & \cos\Delta\psi & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.16}$$

Note that by their construction, global and local motion groups are always invertible.

A trim is a cruising condition for the vehicle in which the velocities $\boldsymbol{\nu}$ remain constant. To predict the evolution of the local motion group $\mathbf{m}(t)$ during a trim, we take the derivative of $\mathbf{m}$ with respect to time evaluated at $(\Delta x, \Delta y, \Delta\psi) = 0$, that is, $\mathbf{m} = \mathbf{I}_{3\times 3}$. This derivative, $\boldsymbol{\xi}$, is derived below.

$$\begin{aligned} \boldsymbol{\xi} \equiv \left.\frac{\partial \mathbf{m}}{\partial t}\right|_{\mathbf{m}=\mathbf{I}} &= \left.\begin{bmatrix} -r\sin\Delta\psi & -r\cos\Delta\psi & u \\ r\cos\Delta\psi & -r\sin\Delta\psi & v \\ 0 & 0 & 0 \end{bmatrix}\right|_{\Delta\psi=0} \\ &= \begin{bmatrix} 0 & -r & u \\ r & 0 & v \\ 0 & 0 & 0 \end{bmatrix} \end{aligned} \tag{3.17}$$

The derivative $\boldsymbol{\xi}$ is used in the following first-order linear matrix differential equation:

$$\dot{\mathbf{m}} = \mathbf{m}\,\boldsymbol{\xi} \tag{3.18}$$

Equation (3.18) is solved using the matrix exponential. The solution from an initial

position at the origin ($\mathbf{m}(0) = \mathbf{I}_{3\times 3}$) is shown below.

$$\begin{aligned} \mathbf{m} &= \mathbf{m}(0)e^{\boldsymbol{\xi} t} \\ &= e^{\boldsymbol{\xi} t} \end{aligned} \tag{3.19}$$

Equation (3.19) has an explicit form in 3 DOF: $\mathbf{m}$ has the form of (3.16) with:

$$\left.\begin{aligned} \Delta x &= (u\sin(rt) + v(\cos(rt) - 1))/r \\ \Delta y &= (u(1 - \cos(rt)) + v\sin(rt))/r \\ \Delta\psi &= rt \end{aligned}\right\} \quad \text{for} \quad r \neq 0$$
$$\Delta x = ut,\ \Delta y = vt,\ \Delta\psi = 0 \qquad \text{for} \quad r = 0 \tag{3.20}$$

3.3.2 Combining Motion Groups Through Concatenation

Motion groups are a convenient representation for the maneuver automaton framework because local motion groups can be concatenated through multiplication. Starting from an initial configuration $\mathbf{M}_0$, the configuration after i maneuvers with corresponding local motion groups $\mathbf{m}_1$, $\mathbf{m}_2$, etc., is:

$$\mathbf{M}_i = \underbrace{\underbrace{\underbrace{\mathbf{M}_0\mathbf{m}_1}_{\mathbf{M}_1}\mathbf{m}_2}_{\mathbf{M}_2}\ldots\mathbf{m}_{i-1}}_{\mathbf{M}_{i-1}}\mathbf{m}_i \tag{3.21}$$

Another way of expressing (3.21) is:

$$\mathbf{m}_i : \mathbf{M}_{i-1} \rightarrow \mathbf{M}_i \qquad \forall i > 0 \tag{3.22}$$

The next section describes how to use this property of local and global motion groups to easily construct motion plans.

3.4 Motion Plans for Planar Vehicles

Through the use of (3.21), entire motion plans can be constructed by concatenating maneuvers. As long as the maneuvers are feasible, then the motion plan is feasible. A motion plan p consists of $|p|$ maneuvers or *path legs*, where $|p|$ is the *plan length*. There are $|p|+1$ *nodes*, including the initial node. Each node can be represented by a global motion group $\mathbf{M}$.

The motion plan serves as a reference trajectory for the vehicle. Because of disturbances, modeling error and underactuation, the actual vehicle trajectory may differ from the reference. Nonetheless, it is desirable to have a continuous and smooth reference trajectory so that path errors and overshoot are minimized. Because the position at the end of maneuver $\mathbf{m}_i$ matches the position at the beginning of the next maneuver $\mathbf{m}_{i+1}$ through the concatenation procedure, the continuity of the reference trajectory is guaranteed. Furthermore, because the orientation of the vehicle at the end of maneuver $\mathbf{m}_i$ matches the orientation at the beginning of maneuver $\mathbf{m}_{i+1}$, the are no heading discontinuities either. The various maneuvers that form the motion plans are discussed below.

3.4.1 Trim Library

Trims represent discrete points in the velocity ($\boldsymbol{\nu}$) space. For an underactuated vehicle with two control inputs (the primary vehicle model considered in this thesis), we actively control only the surge and yaw velocities u and r. The set of discrete trims forms a trim library called $\mathcal{T}$. One of the simplest sets of speed controller setpoints is a 3×3 grid of points in the (u, r) space using nominal surge and yaw velocities U and R respectively. Each of these setpoints is assigned a letter code as follows:

$$\begin{bmatrix} (U,R) & (U,0) & (U,-R) \\ (0,R) & (0,0) & (0,-R) \\ (-U,R) & (-U,0) & (-U,-R) \end{bmatrix} \Leftrightarrow \begin{bmatrix} \mathtt{a} & \mathtt{b} & \mathtt{c} \\ \mathtt{d} & \mathtt{e} & \mathtt{f} \\ \mathtt{g} & \mathtt{h} & \mathtt{i} \end{bmatrix} \tag{3.23}$$

A richer library of setpoints could be used, at the price of higher computation

costs for the planner. Each trim remains at one speed controller setpoint for some amount of time; for example, trim `c` keeps the vehicle driving at its nominal forward speed U and a constant turn rate $-R$.

When u and r alone are being regulated by the speed controller, the sideslip velocity v may be nonzero, depending on the speed controller setpoint and the vehicle dynamics. While v is unregulated, it can be calculated for each setpoint given the plant model $\{\mathbf{a}(\boldsymbol{\nu}), \mathbf{b}\}$.

Each maneuver has an associated reference velocity vector $\boldsymbol{\nu}_\mathbf{r}$. Unless two consecutive maneuvers are the same, there is a discontinuity in the reference velocity at the transition. In the original Maneuver Automaton framework, motion primitives are used to smoothly connect the trim points in the velocity space between one maneuver and the next. In our framework, we instead handle the trim velocity transitions by predicting the transient behavior; this prediction of the mean error is derived in Section 3.6.

Figure 3-3 shows a motion plan with $|p| = 9$ using the trims defined in (3.23). The plan in the figure can be written using the speed controller codes `ebabbccfbb`, where `e` refers to the speed at the start of the plan. This representation is convenient but it does not capture the trim times, so it does not completely describe the plan. A complete plan description would also include the durations of each maneuver; for example, the motion plan shown in Figure 3-3 is described by a 1×9 vector of fives, as each maneuver is five seconds long.

3.4.2 Waypoint Maneuvers

Waypoints are fixed points in space that are used to enrich the planning domain. If all trim maneuvers are constrained to a fixed duration, then the set of reachable configuration space is finite using trim maneuvers alone. Waypoints expand the reachable configuration space and provide shortcuts (in a planning sense) across large tracts of the configuration space, as shown in Figure 3-4. For example, a waypoint can be used to traverse a large open region of the space with one maneuver instead of many fixed-duration trims concatenated together.

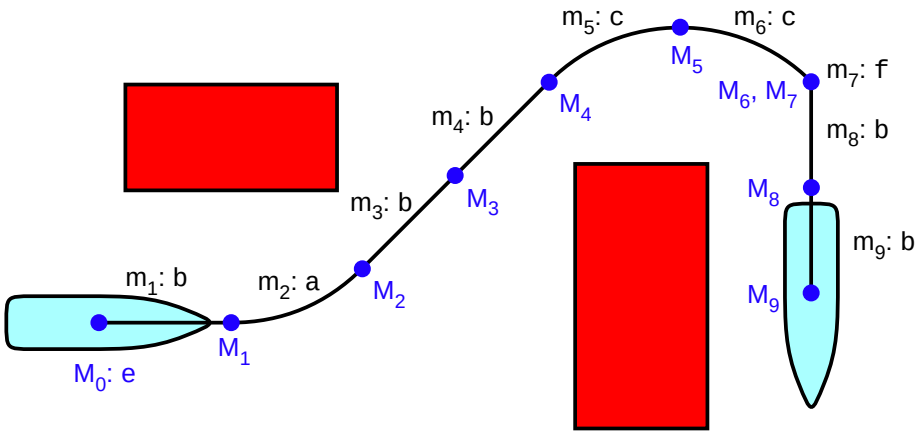

Figure 3-3: The motion plan `ebabbccfbb` uses 9 trims, each 5 seconds long. The red rectangles represent obstacles.

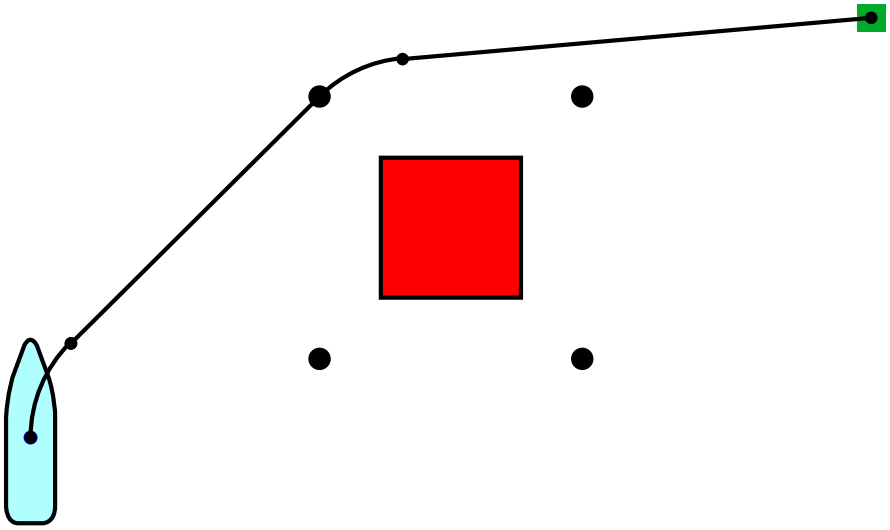

Figure 3-4: Waypoints (large circles) are automatically generated around the obstacle to enrich the planning domain. A waypoint is also placed at the goal.

Waypoint locations can be chosen manually or they can be automatically generated based on the map. For example, waypoints could be placed at the nodes of a Voronoi graph of the free space, or they could be placed near the corners of the obstacles. The latter approach is generally used in this thesis. The motivation for placing waypoints near the corners of obstacles is that the shortest path through a cluttered environment consists of line segments connecting the corners of the obstacles; by placing waypoints at the corners (with a safe buffer distance), then the best path given the constraints of the motion planning framework is likely to be very close to the true best path.

To drive to a waypoint from any position in the free space, the vehicle uses two maneuvers. The first maneuver aligns the heading of the vehicle with the waypoint, and the second maneuver is used to drive straight ahead to the waypoint. This approach is used to maintain the heading continuity throughout the motion plan. These maneuvers can be seen in Figure 3-4, and their construction is described below.

Waypoint Motion Groups

Because waypoints are global position references, they correspond to global motion groups $\mathbf{M}$. However, to be used in the maneuver automaton framework, and (3.21) in particular, driving to a waypoint must be represented as a local motion group $\mathbf{m}$. The waypoint specifies only the Cartesian coordinates (X_w, Y_w), but the corresponding global motion group $\mathbf{M}_w$ also includes an orientation angle. If we assume that the approach to the waypoint will be a straight line, then the angle is $\psi_w = \tan^{-1} \frac{Y_w - Y_0}{X_w - X_0}$ where the coordinates of the end of the previous maneuver are (X_0, Y_0). Knowing the initial global motion group $\mathbf{M}_0$ and the final global motion group $\mathbf{M}_w$, we can derive the local motion group for the maneuver, $\mathbf{m}_w$. Recall that motion groups are always invertible.

$$\begin{aligned} \mathbf{M}_0 \mathbf{m}_w &= \mathbf{M}_w \\ \mathbf{m}_w &= {\mathbf{M}_0}^{-1} \mathbf{M}_w \end{aligned} \tag{3.24}$$

The position continuity of the reference trajectory is preserved when using the waypoint maneuver $\mathbf{m}_w$, but unless the vehicle at $\mathbf{M}_0$ is pointing directly at the waypoint ($\psi_0 = \psi_w$) then the heading continuity will not be preserved. To enforce the heading continuity, a trim maneuver must be inserted into the motion plan to steer the vehicle toward the waypoint. There are several ways to align the vehicle with a waypoint: the vehicle may enter a forward turn in the direction of the waypoint (Figure 3-5), the vehicle may come to a stop and turn in place (Figure 3-6), or other strategies may be employed. If the former strategy is used, the vehicle may never be able to line up the waypoint if the waypoint is within a circle on either side of the vehicle whose radius equals the vehicle's turning radius. On the other hand, turning in place may be a difficult maneuver for the vehicle to perform. In our planning framework, the planner first tries the former approach; if the waypoint is unreachable then it tries the second approach. The trim durations for each approach are derived below. In the derivations, the initial vehicle position and orientation is (X_1, Y_1, ψ_1) and the waypoint is located at (X_4, Y_4).

Forward Turn

If the alignment trim is a forward turn (trim `a` or `c`), then the vehicle will drive around an arc of a circle during the trim (Figure 3-5). The radius of the circle is $R_2 = |u/r|$. During the turn the vehicle will be oriented at an angle $\theta = -\tan^{-1} v/r \approx -v/r$ with respect to the tangent of the circle due to sideslip. Using trigonometric addition formulas, the center of the circle is derived below.

$$\begin{aligned} X_2 &= X_1 - \frac{u}{r}\sin\psi_1 - \frac{v}{r}\cos\psi_1 \\ Y_2 &= Y_1 + \frac{u}{r}\cos\psi_1 - \frac{v}{r}\sin\psi_1 \end{aligned} \tag{3.25}$$

The angle of the tangent to the circle at the vehicle's initial position is $\psi_2 = \psi_1 - \theta$. The distance from the center of the circle to the waypoint is R_4, and the angle of the connecting segment is ψ_4. If the waypoint is inside the circle ($R_4 < R_2$) then it is inaccessible using a forward turn. However, if it is outside the circle then it can be

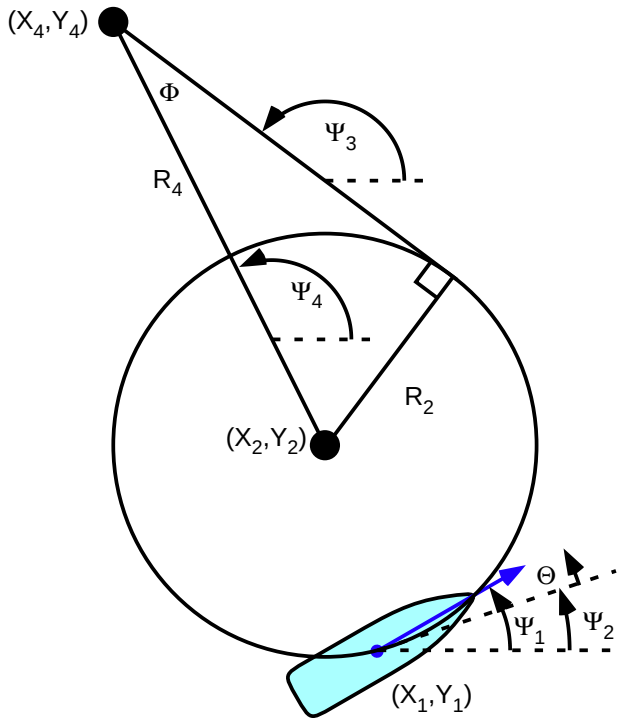

Figure 3-5: Derivation of the trim time using a forward turn. This approach is only valid when the waypoint (X_4, Y_4) is outside the circle defined by the radius R_2.

reached after a finite amount of time in the trim. The angle between the tangent to the circle passing through the waypoint (ψ_3) and the line to the center of the circle is $\phi = \psi_3 - \psi_4 = \sin^{-1} R_2/R_4$. That means the pull-out angle neglecting sideslip is $\psi_3 = \psi_4 + \phi$. The actual pull-out angle is earlier, at $\psi_3 - \theta$, but the initial pull-in angle is also earlier, at $\psi_2 = \psi_1 - \theta$. Therefore the trim duration is $t = (\psi_3 - \psi_1)/r$.

Turn In Place

If the waypoint is inaccessible using a forward turn ($R_4 < R_2$ in the derivation above) then it can be reached by stopping and turning in place (trim `d` or `f`, Figure 3-6). This analysis is limited to a non-minimum phase underactuated vehicle whose source of yaw moment causes a negative sway motion; the analysis for different vehicle configurations is slightly different.

A pure yaw trim may still cause sideslip, resulting in a circular vehicle motion with a radius $R = |v/r|$. The center of the circle is shown below.

$$\begin{aligned} X_2 &= X_1 + R\cos\psi_1 \\ Y_2 &= Y_1 + R\sin\psi_1 \end{aligned} \tag{3.26}$$

The path to the waypoint passes through the center of the circle, so the pull-out heading ψ_4 is easily computed. Finally, the trim time is $t = (\psi_4 - \psi_1)/r$.

Waypoint Motion Plans

A motion plan using waypoints is shown in Figure 3-7. This motion plan uses forward turns to line up with two of the waypoints, and a turn in place to line up with the third. By comparing Figure 3-7 with Figure 3-3, we see that using waypoints can reduce the total number of maneuvers in the plan; this results in faster searches using the planning algorithm. Furthermore, all but one of the trims in Figure 3-7 are associated with a waypoint, meaning that they are added in the planning same step as that waypoint; this further reduces the effective plan length. The plan in the figure could be written as `ebawcwfw`; to capture the association between certain trims

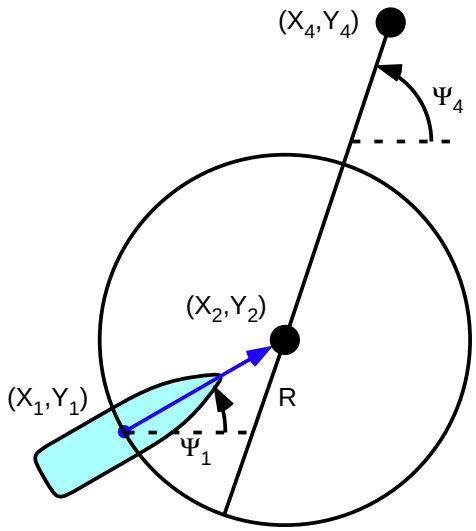

Figure 3-6: Derivation of the trim time when turning in place.

and the waypoints, it could also be written as `eb(aw)(cw)(fw)`. Here we can see that the effective plan length, from the planner's perspective, is four. However, as before, these representations do not capture the variable trim length or the locations of the waypoints; these pieces of data are necessary to fully describe the motion plan.

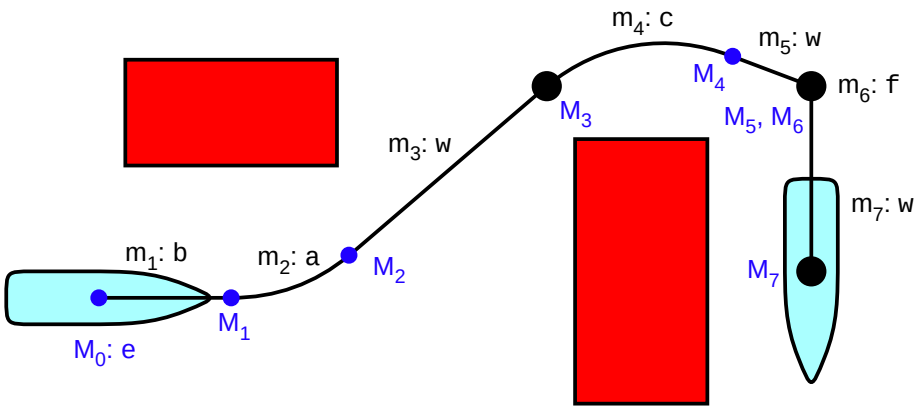

Figure 3-7: The motion plan `ebawcwfw` uses 4 trims and 3 waypoints (indicated by large circles). The trims preceding each waypoint are variable-length trims used to align the reference trajectory with the waypoint.

The motion plans generated by the planner represent the reference trajectory that the vehicle is asked to follow. In practice the vehicle may not be able to follow the

motion plan exactly, due to predictable effects (transients, reference velocity discontinuities, etc.) and unpredictable effects (stochastic disturbances and modeling error). The next section describes the control law used by the vehicle to track the reference trajectory and reject unpredictable disturbances.

3.5 Control Law

The vehicle controller is used to track the reference position and velocity as the vehicle executes a motion plan. There are many different types of controllers that could be used to regulate the system (see Chapter 2). However, the planner needs to be able to predict the error statistics associated with following each motion plan, and the error prediction can only be solved analytically if the control law is relatively simple. For this reason we use linear state feedback control: it can be easily and quickly designed on-line from an estimate of the vehicle model parameters, and its effect on the path-following error dynamics can be predicted analytically. In this section we derive the control law and describe how the controller changes for different maneuvers.

3.5.1 State Feedback Control

The goal of state feedback control is to drive the full state $\mathbf{x}$ (3.13) to the reference state $\mathbf{r}$. The control law, shown in Figure 3-8, includes a feedforward term $\mathbf{K_r r}$ and a feedback term $-\mathbf{K_x x}$. The feedback term alone drives the state to $\mathbf{0}$, and the feedforward term shifts the target from $\mathbf{0}$ to the reference $\mathbf{r}$.

$$\boldsymbol{\tau} = -\mathbf{K_x x} + \mathbf{K_r r} \tag{3.27}$$

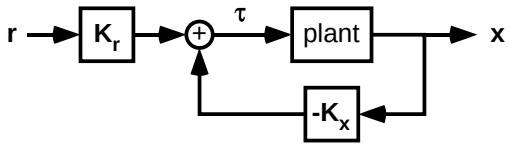

Figure 3-8: Feedforward/feedback control law.

The reference can be split into a reference speed $\boldsymbol{\nu}_\mathbf{r}$ and a reference position/orientation $\boldsymbol{\eta}_\mathbf{r}$. The error between the state and the reference, $\mathbf{e} = \mathbf{x} - \mathbf{r}$, can similarly be split into speed error $\boldsymbol{\nu}_\mathbf{e}$ and position/orientation error $\boldsymbol{\eta}_\mathbf{e}$.

$$\begin{aligned} \mathbf{e} &= \mathbf{r} - \mathbf{x} \\ \begin{bmatrix} \boldsymbol{\nu}_\mathbf{e} \\ \boldsymbol{\eta}_\mathbf{e} \end{bmatrix} &= \begin{bmatrix} \boldsymbol{\nu}_\mathbf{r} \\ \boldsymbol{\eta}_\mathbf{r} \end{bmatrix} - \begin{bmatrix} \boldsymbol{\nu} \\ \boldsymbol{\eta} \end{bmatrix} \end{aligned} \tag{3.28}$$

Note that because the reference state vector $\mathbf{r}$ contains both velocities and positions, it must be self-consistent, that is, its two components must satisfy the following differential equation:

$$\dot{\boldsymbol{\eta}}_\mathbf{r} = \mathbf{J}(\psi_r)\boldsymbol{\nu}_\mathbf{r} \tag{3.29}$$

Furthermore, we assume that the reference velocity is constant over each maneuver, that is, $\dot{\boldsymbol{\nu}}_\mathbf{r} = \mathbf{0}$. This is always true when using the motion plans constructed in the previous section. The feedback matrix $\mathbf{K}_\mathbf{x}$ has a part that operates on the velocity vector, $\mathbf{K}_{\boldsymbol{\nu}}$, and a part that operates on the position vector, $\mathbf{K}_{\boldsymbol{\eta}}$. The feedforward matrix $\mathbf{K}_\mathbf{r}$ has a similar structure.

$$\mathbf{K}_\mathbf{x} = \begin{bmatrix} \mathbf{K}_{\boldsymbol{\nu}} & \mathbf{K}_{\boldsymbol{\eta}} \end{bmatrix} \tag{3.30}$$

$$\mathbf{K}_\mathbf{r}(\boldsymbol{\nu}) = \begin{bmatrix} \mathbf{K}_{\boldsymbol{\nu}} - \mathbf{b}^{-1}\mathbf{a}(\boldsymbol{\nu}) & \mathbf{K}_{\boldsymbol{\eta}} \end{bmatrix} \tag{3.31}$$

The term $\mathbf{K}_{\boldsymbol{\nu}} - \mathbf{b}^{-1}\mathbf{a}(\boldsymbol{\nu})$ counteracts the damping in the plant. The inverse of $\mathbf{b}$ exists if the vehicle is fully actuated (the rank of $\mathbf{b}$ is 3). For now we assume that the vehicle is fully actuated and $\mathbf{a}(\boldsymbol{\nu})$ and $\mathbf{b}$ are known exactly.

3.5.2 Control Law Derivation

Plugging the control law (3.27) into the nonlinear state space system (3.14) results in the following closed-loop system.

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A}(\mathbf{x})\mathbf{x} - \mathbf{B}\mathbf{K}_{\mathbf{x}}\mathbf{x} + \mathbf{B}\mathbf{K}_{\mathbf{r}}(\boldsymbol{\nu})\mathbf{r} + \mathbf{F}_{\boldsymbol{\nu}}(\psi) + \mathbf{w} \\ &= \mathbf{A}_{\mathbf{cl}}(\mathbf{x})\mathbf{x} + \mathbf{B}\mathbf{K}_{\mathbf{r}}(\boldsymbol{\nu})\mathbf{r} + \mathbf{F}_{\boldsymbol{\nu}}(\psi) + \mathbf{w} \qquad (3.32) \\ \text{where} \quad \mathbf{A}_{\mathbf{cl}}(\mathbf{x}) &= \mathbf{A}(\mathbf{x}) - \mathbf{B}\mathbf{K}_{\mathbf{x}} = \begin{bmatrix} \mathbf{a}(\boldsymbol{\nu}) - \mathbf{b}\mathbf{K}_{\boldsymbol{\nu}} & -\mathbf{b}\mathbf{K}_{\boldsymbol{\eta}} \\ \mathbf{J}(\psi) & \mathbf{0} \end{bmatrix} \qquad (3.33) \end{aligned}$$

The control law can be written in terms of the control matrix definitions (3.30-3.31) and the error vectors.

$$\begin{aligned} \boldsymbol{\tau} &= -\mathbf{K}_{\boldsymbol{\nu}}\boldsymbol{\nu} - \mathbf{K}_{\boldsymbol{\eta}}\boldsymbol{\eta} + \left(\mathbf{K}_{\boldsymbol{\nu}} - \mathbf{b}^{-1}\mathbf{a}(\boldsymbol{\nu})\right)\boldsymbol{\nu}_{\mathbf{r}} + \mathbf{K}_{\boldsymbol{\eta}}\boldsymbol{\eta}_{\mathbf{r}} \\ &= -\mathbf{K}_{\boldsymbol{\nu}}\boldsymbol{\nu}_{\mathbf{e}} - \mathbf{K}_{\boldsymbol{\eta}}\boldsymbol{\eta}_{\mathbf{e}} - \mathbf{b}^{-1}\mathbf{a}(\boldsymbol{\nu})\,\boldsymbol{\nu}_{\mathbf{r}} \qquad (3.34) \end{aligned}$$

The control law (3.34) is invariant to the orientation of the reference trajectory ψ_r only if $\mathbf{K}_{\boldsymbol{\eta}}\boldsymbol{\eta}_{\mathbf{e}}$ is not a function of ψ_r. The position error $\boldsymbol{\eta}_{\mathbf{e}}$ can be mapped to reference-local coordinates (a coordinate system aligned with the ψ_r, Figure 3-9):

$$\boldsymbol{\eta}_{\mathbf{e0}} = \mathbf{J}(\psi_r)^T\boldsymbol{\eta}_{\mathbf{e}}. \qquad (3.35)$$

The full error vector $\mathbf{e}$ can be rotated to reference-local coordinates with the 6×6 matrix $\mathbf{J_2}$. The rotated error vector is $\mathbf{e_0}$.

$$\begin{aligned} \mathbf{J_2} &\equiv \begin{bmatrix} \mathbf{I}_{3\times3} & \mathbf{0} \\ \mathbf{0} & \mathbf{J}(\psi_r) \end{bmatrix} \qquad (3.36) \\ \mathbf{e_0} &\equiv \begin{bmatrix} \boldsymbol{\nu}_{\mathbf{e}} \\ \boldsymbol{\eta}_{\mathbf{e0}} \end{bmatrix} = \mathbf{J_2}^T\mathbf{e} \qquad (3.37) \end{aligned}$$

Now consider the feedback matrix designed for the full state system (3.14) when $\psi = 0$. In this case, the lower left quadrant of $\mathbf{A}$ is $\mathbf{I}_{3\times3}$. We denote that feedback

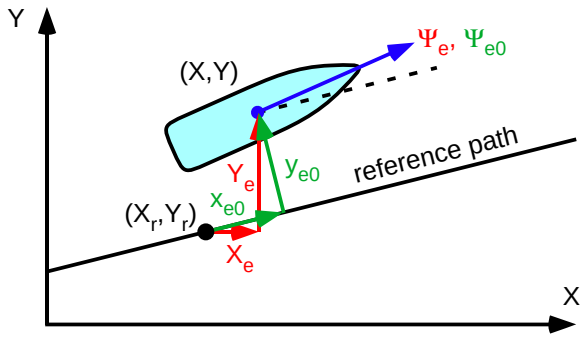

Figure 3-9: The vehicle position (X, Y, ψ) relative to the reference position (X_r, Y_r, ψ_r) expressed in global coordinates (X_e, Y_e, ψ_e) and reference-local coordinates $(x_{e0}, y_{e0}, \psi_{e0})$.

matrix as $\mathbf{K_{x0}}$, which is partitioned as follows:

$$\mathbf{K_{x0}} = \begin{bmatrix} \mathbf{K}_\nu & \mathbf{K}_{\eta 0} \end{bmatrix} \tag{3.38}$$

As ψ changes and the lower left quadrant of $\mathbf{A}$ becomes $\mathbf{J} \neq \mathbf{I}$, then the position feedback matrix $\mathbf{K}_\eta$ changes as well. It is related to $\mathbf{K}_{\eta 0}$ as follows:

$$\begin{aligned} \mathbf{K}_\eta \boldsymbol{\eta}_\mathbf{e} &= \mathbf{K}_{\eta 0} \boldsymbol{\eta}_{\mathbf{e0}} \\ &= \mathbf{K}_{\eta 0} \mathbf{J}(\psi_r)^T \boldsymbol{\eta}_\mathbf{e} \\ \mathbf{K}_\eta &= \mathbf{K}_{\eta 0} \mathbf{J}(\psi_r)^T \end{aligned} \tag{3.39}$$

Rewriting (3.34), we can now write the control law in terms of reference-local coordinates only.

$$\boldsymbol{\tau} = -\mathbf{K}_\nu \boldsymbol{\nu}_\mathbf{e} - \mathbf{K}_{\eta 0} \boldsymbol{\eta}_{\mathbf{e0}} - \mathbf{b}^{-1} \mathbf{a}(\boldsymbol{\nu})\, \boldsymbol{\nu}_\mathbf{r} \tag{3.40}$$

With these coordinate transformations, it is only necessary to design a feedback controller (3.38) once for the $\psi = 0$ system, and the control law can be applied in the reference-local coordinates. This approach is much easier than redesigning the controller whenever the heading changes.

3.5.3 Controller Structure and LQR Design

In the motion planning framework, there is no need to specify whether a particular maneuver is completely open-loop (playing back a sequence of pre-recorded control actions), completely closed-loop (position, orientation and velocity feedback) or somewhere in between. The trim library $\mathcal{T}$ is defined as a set of speed control setpoints to regulate the vehicle's surge velocity u and yaw rate r. The speed controller is closed-loop for velocity, but open-loop for position; the speed controller alone cannot correct for disturbances that push the vehicle off the reference trajectory. Alternatively, the controller could add position feedback during the maneuver. While this is feasible for fully actuated vehicles, it may be impossible for underactuated vehicles. For example, with an underactuated surface vessel it is impossible to hold a fixed position and orientation (underactuated point stabilization) with a fixed feedback control law [10, 38]. However, the kinematics of forward and backward motion mean that position *can* be regulated for certain trims. For the underactuated vehicle that we consider, position feedback is possible for the trims in which $u \neq 0$: `a`, `b`, `c`, `g`, `h` and `i`. However, in this example we restrict position feedback to forward maneuvers, `a`, `b` and `c`. Because driving to a waypoint (`w`) is a special case of `b`, the waypoint maneuver has position feedback as well.

The reference-local feedback control law has the following form in the speed control and speed/position control cases.

$$\mathbf{K_{x0}} = \begin{cases} [\, \mathbf{K_\nu} \quad \mathbf{0} \quad] & \text{for} \quad \texttt{defghi} \\ [\, \mathbf{K_\nu} \quad \mathbf{K_{\eta 0}} \,] & \text{for} \quad \texttt{abcw} \end{cases} \tag{3.41}$$

The state feedback controller is a linear quadratic regulator (LQR) designed around the vehicle's $\mathbf{A}(\boldsymbol{\nu})$ and $\mathbf{B}$ matrices (3.14) with $\psi = 0$. The LQR controller is tuned with a state cost matrix $\mathbf{Q}_{lqr}$ and a control cost matrix $\mathbf{R}_{lqr}$. For simplicity, we can set $\mathbf{Q}_{lqr} = \mathbf{I}$ and $\mathbf{R}_{lqr} = \rho\,\mathbf{I}$, so that the only tuning parameter is the scalar value ρ. For the maneuvers for which there is no position feedback (`defghi`), the state cost on the position and orientation states is zero and $\mathbf{Q}_{lqr} = \mathrm{diag}(\mathbf{I}, \mathbf{0})$.

Because the $\mathbf{A}$ matrix depends on the vehicle's velocity, the controller is designed around each maneuver and its associated reference velocity $\boldsymbol{\nu}_\mathbf{r}$. A complete motion plan can therefore store a controller $\{\mathbf{K_{x0}}, \mathbf{K_r}\}$ with each maneuver. During the execution of the motion plan, the vehicle simply loads the reference velocity $\boldsymbol{\nu}_\mathbf{r}$ for the maneuver, computes the reference position $\boldsymbol{\eta}_\mathbf{r}(t)$ from (3.29), and loads the controller associated with the maneuver.

Once the motion plan is specified and a linear state feedback controller is designed for each maneuver in the plan, then it is possible to predict the mean trajectory of the vehicle and the variance of the path-following error. These calculations are needed to compute the predicted collision probability for the vehicle; the planner uses this prediction to select the optimal plan. The next section covers the prediction of the path-following error statistics for a motion plan.

3.6 Error Dynamics

In this section we derive the dynamics of the path-following error for a motion plan, focusing on the error evolution through each maneuver. The state error is used to predict the probability of hitting obstacles and to predict the amount of information available to the online parameter learning algorithm described in Chapter 4.

Given the assumption that the disturbances acting on the vehicle are Gaussian, the vehicle state is normally distributed around a mean state at every moment in time. We derive analytic expressions for the mean state error (3.57) and the state error variance (3.65), and in Section 3.7 we develop an analytic approximation for the collision probability. These analytic expressions allow the planner to evaluate each motion plan very quickly so that the planner can run nearly in real-time.

First, in Section 3.6.1, the error dynamics are expressed in global coordinates as a simple nonlinear differential equation. Next they are transformed to a reference-local coordinate system (Section 3.6.2) where the mean and variance of the error are solved analytically (Sections 3.6.3 and 3.6.4). Finally, error predictions for underactuated vehicles are discussed in Section 3.6.5 and error propagation across maneuver

transitions is described in Section 3.6.6.

3.6.1 Global Error Dynamics

The first step is to express the error $\mathbf{e}(t)$ as a differential equation by finding a simple expression for $\dot{\mathbf{e}}(t)$. By the definition of $\mathbf{e}$ (3.28), the derivative of the error is:

$$\begin{aligned}
\dot{\mathbf{e}} = \dot{\mathbf{x}} - \dot{\mathbf{r}} &= \mathbf{A_{cl}}(\mathbf{x})\mathbf{x} + \mathbf{B}\mathbf{K_r}(\boldsymbol{\nu})\mathbf{r} + \mathbf{F}_{\boldsymbol{\nu}}(\psi) + \mathbf{w} - \dot{\mathbf{r}} \\
&= \mathbf{A_{cl}}(\mathbf{x})\mathbf{e} + \mathbf{A_{cl}}(\mathbf{x})\mathbf{r} + \mathbf{B}\mathbf{K_r}(\boldsymbol{\nu})\mathbf{r} - \dot{\mathbf{r}} + \mathbf{F}_{\boldsymbol{\nu}}(\psi) + \mathbf{w} \\
&= \mathbf{A_{cl}}(\mathbf{x})\mathbf{e} + \mathbf{A_r}(\psi)\mathbf{r} - \dot{\mathbf{r}} + \mathbf{F}_{\boldsymbol{\nu}}(\psi) + \mathbf{w}
\end{aligned} \tag{3.42}$$

The matrix $\mathbf{A_r}(\psi)$ used in (3.42) is defined below.

$$\begin{aligned}
\mathbf{A_r}(\psi) &\equiv \mathbf{A_{cl}}(\mathbf{x}) + \mathbf{B}\mathbf{K_r}(\boldsymbol{\nu}) \\
&= \begin{bmatrix} \mathbf{a}(\boldsymbol{\nu}) - \mathbf{b}\mathbf{K}_{\boldsymbol{\nu}} & -\mathbf{b}\mathbf{K}_{\boldsymbol{\eta}} \\ \mathbf{J}(\psi) & \mathbf{0} \end{bmatrix} + \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{K}_{\boldsymbol{\nu}} - \mathbf{b}^{-1}\mathbf{a}(\boldsymbol{\nu}) & \mathbf{K}_{\boldsymbol{\eta}} \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{J}(\psi) & \mathbf{0} \end{bmatrix}
\end{aligned} \tag{3.43}$$

The second and third terms of (3.42) can be condensed by recalling the self-consistency of $\mathbf{r}$ (3.29).

$$\begin{aligned}
\mathbf{A_r}(\psi)\mathbf{r} - \dot{\mathbf{r}} &= \begin{bmatrix} \mathbf{0} \\ \mathbf{J}(\psi)\boldsymbol{\nu}_\mathbf{r} \end{bmatrix} - \begin{bmatrix} \dot{\boldsymbol{\nu}}_\mathbf{r} = \mathbf{0} \\ \dot{\boldsymbol{\eta}}_\mathbf{r} = \mathbf{J}(\psi_r)\boldsymbol{\nu}_\mathbf{r} \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{0} \\ (\mathbf{J}(\psi_e) - \mathbf{I})\,\mathbf{J}(\psi_r)\boldsymbol{\nu}_\mathbf{r} \end{bmatrix}
\end{aligned} \tag{3.44}$$

In (3.44) we used the relation $\mathbf{J}(\psi) = \mathbf{J}(\psi_e)\mathbf{J}(\psi_r)$, which arises from $\psi = \psi_e + \psi_r$. Assuming the heading error is small, $\mathbf{J}(\psi_e) - \mathbf{I}$ can be replaced with $\mathbf{S}_r\psi_e$, where $\mathbf{S}_r$

was defined in (3.12). With this simplification, the lower submatrix of (3.44) becomes:

$$\begin{aligned} (\mathbf{J}(\psi_e) - \mathbf{I})\,\mathbf{J}(\psi_r)\boldsymbol{\nu}_\mathbf{r} &= \mathbf{S}_r\psi_e\mathbf{J}(\psi_r)\boldsymbol{\nu}_\mathbf{r} \\ &= \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{S}_r\mathbf{J}(\psi_r)\boldsymbol{\nu}_\mathbf{r} \end{bmatrix}\boldsymbol{\eta}_\mathbf{e} \\ \text{and we define} \quad \mathbf{A_{22}}(\mathbf{r}) &\equiv \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{S}_r\mathbf{J}(\psi_r)\boldsymbol{\nu}_\mathbf{r} \end{bmatrix}. \end{aligned} \tag{3.45}$$

Using $\mathbf{A_{22}}(\mathbf{r})$ as defined above, (3.44) reduces to:

$$\mathbf{A}_\mathbf{r}(\psi)\mathbf{r} - \dot{\mathbf{r}} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A_{22}}(\mathbf{r}) \end{bmatrix}\mathbf{e}. \tag{3.46}$$

This new matrix (3.46) can be combined with $\mathbf{A_{cl}}(\mathbf{x})$ to form a new nonlinear state transition matrix that incorporates the reference information.

$$\mathbf{A_{clr}}(\mathbf{x},\mathbf{r}) = \begin{bmatrix} \mathbf{a}(\boldsymbol{\nu}) - \mathbf{b}\mathbf{K}_{\boldsymbol{\nu}} & -\mathbf{b}\mathbf{K}_{\boldsymbol{\eta}} \\ \mathbf{J}(\psi) & \mathbf{A_{22}}(\mathbf{r}) \end{bmatrix} \tag{3.47}$$

When $\mathbf{A_{clr}}$ is inserted into the error dynamics equation (3.42), the result is a nonlinear time-varying differential equation.

$$\dot{\mathbf{e}} = \mathbf{A_{clr}}(\mathbf{x},\mathbf{r})\,\mathbf{e} + \mathbf{F}_{\boldsymbol{\nu}}(\psi) + \mathbf{w} \tag{3.48}$$

3.6.2 Reference-Local Error Dynamics

The error dynamics $\mathbf{e}(t)$ cannot be solved directly from (3.48), because $\mathbf{A_{clr}}(\mathbf{x},\mathbf{r})$ is a function of the vehicle state. Even when the dynamics equation (3.4) is linearized ($\mathbf{a}(\boldsymbol{\nu}) = \mathbf{a}$) and the heading error is small ($\mathbf{J}(\psi) \approx \mathbf{J}(\psi_r)$ and $\mathbf{F}_{\boldsymbol{\nu}}(\psi) \approx \mathbf{F}_{\boldsymbol{\nu}}(\psi_r)$), the differential equation is still time-varying if the yaw rate reference is nonzero ($r_r \neq 0$), since $\psi_r = \psi_r(t)$. However, the evolution of the reference-local error $\mathbf{e_0}(t)$ is more favorable to computation. This quantity was computed in (3.37) using the matrix $\mathbf{J_2}$. Similarly, the body-reference wind force vector is written in terms of the global force

vector and another rotation matrix $\mathbf{J_1}$:

$$\mathbf{F_\nu}(\psi_r) = \mathbf{J_1}^T\mathbf{F} \quad \text{where} \quad \mathbf{J_1} = \begin{bmatrix} \mathbf{J}(\psi_r) & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}. \tag{3.49}$$

Using these definitions, the evolution of $\mathbf{e_0}$ is shown below.

$$\begin{aligned} \dot{\mathbf{e}}_\mathbf{0} &= \mathbf{J_2}^T\dot{\mathbf{e}} + \dot{\mathbf{J}}_\mathbf{2}^T\mathbf{e} \\ &= \mathbf{J_2}^T\mathbf{A_{clr}}(\mathbf{x},\mathbf{r})\mathbf{e} + \mathbf{J_2}^T\mathbf{J_1}^T\mathbf{F} + \mathbf{J_2}^T\mathbf{w} + \mathbf{J_2}^T\mathbf{S_2}^T\mathbf{e} \\ &= \mathbf{J_2}^T\left(\mathbf{A_{clr}}(\mathbf{x},\mathbf{r}) + \mathbf{S_2}^T\right)\mathbf{J_2}\mathbf{e_0} + \mathbf{J_1}^T\mathbf{F} + \mathbf{w} \end{aligned} \tag{3.50}$$

$$\text{where} \quad \mathbf{S_2} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{S}(r_r) \end{bmatrix} \quad \text{and} \quad \dot{\mathbf{J}}_\mathbf{2} = \mathbf{S_2}\mathbf{J_2} \tag{3.51}$$

The quantity $\mathbf{J_2}^T\left(\mathbf{A_{clr}}(\mathbf{x},\mathbf{r}) + \mathbf{S_2}^T\right)\mathbf{J_2}$ is simplified and renamed $\mathbf{A_0}$,

$$\begin{aligned} \mathbf{A_0} &= \mathbf{J_2}^T\left(\mathbf{A_{clr}}(\mathbf{x},\mathbf{r}) + \mathbf{S_2}^T\right)\mathbf{J_2} \\ &= \begin{bmatrix} \mathbf{a}(\boldsymbol{\nu}) - \mathbf{bK_\nu} & -\mathbf{bK_\eta}\mathbf{J}(\psi_r) \\ \mathbf{I} & \mathbf{J}(\psi_r)^T\left(\mathbf{A_{22}}(\mathbf{r}) + \mathbf{S}(r)^T\right)\mathbf{J}(\psi_r) \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{A_{11}}(\boldsymbol{\nu}) & \mathbf{A_{12}} \\ \mathbf{I} & \mathbf{S_\nu}(\boldsymbol{\nu_r})^T \end{bmatrix} \end{aligned} \tag{3.52}$$

where

$$\mathbf{A_{11}}(\boldsymbol{\nu}) \equiv \mathbf{a}(\boldsymbol{\nu}) - \mathbf{bK_\nu} \tag{3.53}$$

$$\mathbf{A_{12}} \equiv -\mathbf{bK_\eta}\mathbf{J}(\psi_r) = -\mathbf{bK_{\eta 0}} \tag{3.54}$$

$$\begin{aligned} \mathbf{S_\nu}(\boldsymbol{\nu_r}) &\equiv \mathbf{J}(\psi_r)^T\left(\mathbf{A_{22}}(\mathbf{r})^T + \mathbf{S}(r)\right)\mathbf{J}(\psi_r) \\ &= \begin{bmatrix} 0 & -r_r & 0 \\ r_r & 0 & 0 \\ -v_r & u_r & 0 \end{bmatrix}. \end{aligned} \tag{3.55}$$

If the velocity errors are small, then $\mathbf{a}(\boldsymbol{\nu}) \approx \mathbf{a}(\boldsymbol{\nu_r})$ and $\mathbf{A_0}(\boldsymbol{\nu_r})$ is constant for each

maneuver. The simplification of $\mathbf{S}_{\boldsymbol{\nu}}(\boldsymbol{\nu}_\mathbf{r})$ in (3.55) is only valid for 3 DOF vehicles. For 3 DOF vehicles, the reference-local error dynamics equation, shown below, is a linear time-varying differential equation.

$$\dot{\mathbf{e}}_\mathbf{0} = \mathbf{A_0}(\boldsymbol{\nu}_\mathbf{r})\mathbf{e_0} + \mathbf{J_1}^T\mathbf{F} + \mathbf{w} \tag{3.56}$$

This differential equation can be solved for the mean value and the covariance matrix for $\mathbf{e_0}$. These solutions are described below.

3.6.3 Mean Error Evolution

The mean value of (3.56) can be solved analytically using the matrix exponential to find the mean reference-local error throughout the maneuver. First we note that $\mathbf{J_1}(t) = \mathbf{J_1}(0)e^{\mathbf{S_1}t}$. Next we insert this into the explicit solution to the mean ($\mathbf{w} = \mathbf{0}$) differential equation,

$$\begin{aligned}
\dot{\bar{\mathbf{e}}}_\mathbf{0} &= \mathbf{A_0}\bar{\mathbf{e}}_\mathbf{0} + \mathbf{J_1}^T\mathbf{F} \\
\bar{\mathbf{e}}_\mathbf{0}(t) &= e^{\mathbf{A_0}t}\bar{\mathbf{e}}_\mathbf{0}(0) + \int_0^t e^{\mathbf{A_0}(t-\tau)}\mathbf{J_1}^T(\tau)\mathbf{F}d\tau \\
&= e^{\mathbf{A_0}t}\bar{\mathbf{e}}_\mathbf{0}(0) + \int_0^t e^{(\mathbf{A_0}t - \mathbf{A_0}\tau + \mathbf{S_1}^T\tau)}d\tau \times \mathbf{J_1}(0)^T\mathbf{F} \\
&= e^{\mathbf{A_0}t}\bar{\mathbf{e}}_\mathbf{0}(0) + \left(\mathbf{A_0} - \mathbf{S_1}^T\right)^{-1}\left(e^{\mathbf{A_0}t} - e^{\mathbf{S_1}^Tt}\right)\mathbf{J_1}(0)^T\mathbf{F}.
\end{aligned} \tag{3.57}$$

The mean error in global coordinates, $\bar{\mathbf{e}}$, is computed using the following transformation,

$$\bar{\mathbf{e}}(t) = \mathbf{J_2}\left(\psi_r(t)\right)\,\bar{\mathbf{e}}_\mathbf{0}(t). \tag{3.58}$$

3.6.4 Error Covariance Evolution

The covariance of the error vector $\mathbf{e}$ is:

$$\boldsymbol{\Sigma} = E\left[(\mathbf{e} - \bar{\mathbf{e}})(\mathbf{e} - \bar{\mathbf{e}})^T\right] \tag{3.59}$$

where $E[\cdot]$ is the expectation operator over all possible noise sequences $\mathbf{w}(t)$. Similarly, the covariance of the reference-local error vector $\mathbf{e_0}$ is:

$$\mathbf{\Sigma_0} = E\left[(\mathbf{e_0} - \bar{\mathbf{e}}_\mathbf{0})(\mathbf{e_0} - \bar{\mathbf{e}}_\mathbf{0})^T\right] = \mathbf{J_2}^T\mathbf{\Sigma}\mathbf{J_2} \tag{3.60}$$

The covariance matrix of the noise vector is:

$$\mathbf{W} = E\left[\mathbf{w}\mathbf{w}^T\right] = \begin{bmatrix} \mathbf{W}_{\boldsymbol{\nu}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{3.61}$$

and the evolution of $\mathbf{\Sigma_0}$ is the variance evolution equation [35] for the reference-local error dynamics equation (3.56).

$$\dot{\mathbf{\Sigma}}_\mathbf{0} = \mathbf{A_0}\mathbf{\Sigma_0} + \mathbf{\Sigma_0}\mathbf{A_0}^T + \mathbf{W} \tag{3.62}$$

Equation (3.62) is a Riccati equation that can be solved analytically using the method described in [35] or [34]; the latter method is presented here. First a block matrix $\mathbf{M}$ is constructed as follows:

$$\mathbf{M} = \begin{bmatrix} -\mathbf{A_0}^T & \mathbf{0} \\ \mathbf{W} & \mathbf{A_0} \end{bmatrix}. \tag{3.63}$$

Next a block matrix is defined using two square matrices $\mathbf{Y_1}$ and $\mathbf{Y_2}$, each the same size as $\mathbf{\Sigma_0}$.

$$\mathbf{Y} \equiv \begin{bmatrix} \mathbf{Y_1} \\ \mathbf{Y_2} \end{bmatrix} \qquad \mathbf{Y}(0) = \begin{bmatrix} \mathbf{I} \\ \mathbf{\Sigma_0}(0) \end{bmatrix} \tag{3.64}$$

Equation (3.62) is equivalent to the differential equation $\dot{\mathbf{Y}} = \mathbf{MY}$ with the initial condition shown above. The equation is solved using the matrix exponential.

$$\begin{aligned} \mathbf{Y}(t) &= e^{\mathbf{M}t}\mathbf{Y}(0) \\ \mathbf{\Sigma_0}(t) &= \mathbf{Y_2}\mathbf{Y_1}^{-1} \end{aligned} \tag{3.65}$$

The variance of the global error vector is computed using a rotation from the reference-local coordinates back to the global coordinates.

$$\mathbf{\Sigma}(t) = \mathbf{J_2}(t)\mathbf{\Sigma_0}(t)\mathbf{J_2}(t)^T \tag{3.66}$$

There may be numerical stability issues when computing $\mathbf{Y_1}^{-1}$ if $t - t_0$ is too large; this happens when $\mathbf{\Sigma_0}(t)$ approaches the steady state. In that case, the problem is a continuous algebraic Riccati equation,

$$\mathbf{0} = \mathbf{A_0}\mathbf{\Sigma_0}(\infty) + \mathbf{\Sigma_0}(\infty)\mathbf{A_0}^T + \mathbf{W}. \tag{3.67}$$

The solution to this equation is shown in Section B.1, taken from [2].

3.6.5 Error Dynamics for Underactuated Vehicles

The prediction of the mean error (3.57) and the error variance (3.65) applies only to fully-actuated vehicles. In particular, the feedforward term $\mathbf{K_r}(\boldsymbol{\nu})$ (3.31) requires that $\mathbf{b}^{-1}$ exists. For underactuated vehicles this is not the case, either because $\mathbf{b}$ is singular or it is non-square (only two control inputs, for example). Exact cancellation of the damping term is therefore impossible for underactuated vehicles. One workaround is to use the pseudoinverse $\mathbf{b}^*$, but it may be more desirable to apply feedforward to certain states and neglect other states entirely. For generality we define $\mathbf{b}^{(inv)}$, which is used below to construct $\mathbf{K_r}(\boldsymbol{\nu})$. For a fully actuated vehicle, $\mathbf{b}^{(inv)} = \mathbf{b}^{-1}$. For an underactuated vehicle, $\mathbf{b}^{(inv)}$ is the same size as $\mathbf{b}^T$.

$$\mathbf{K_r}(\boldsymbol{\nu}) = \begin{bmatrix} \mathbf{K_\nu} - \mathbf{b}^{(inv)}\mathbf{a}(\boldsymbol{\nu}) & \mathbf{K_\eta} \end{bmatrix} \tag{3.68}$$

The choice of the structure of $\mathbf{b}^{(inv)}$ is left to the control designer. In our example, where the trim library defines u and r setpoints, a reasonable option is to take the inverse of the u and r rows of $\mathbf{b}$ and distribute them over the u and r columns of

$\mathbf{b}^{(inv)}$:

$$\mathbf{b} = \begin{bmatrix} b_1 & 0 \\ 0 & b_2 \\ 0 & b_3 \end{bmatrix} \qquad \mathbf{b}^{(inv)} = \begin{bmatrix} 1/b_1 & 0 & 0 \\ 0 & 0 & 1/b_3 \end{bmatrix}. \tag{3.69}$$

Once $\mathbf{b}^{(inv)}$ is defined, $\mathbf{A_r}$ is updated; for the underactuated vehicle, there is incomplete cancelation of the damping matrix.

$$\begin{aligned} \mathbf{A_r}(\mathbf{x}) &\equiv \mathbf{A_{cl}}(\mathbf{x}) + \mathbf{B}\mathbf{K_r}(\boldsymbol{\nu}) \\ &= \begin{bmatrix} \mathbf{a}(\boldsymbol{\nu}) - \mathbf{b}\mathbf{b}^{(inv)}\mathbf{a}(\boldsymbol{\nu}) & \mathbf{0} \\ \mathbf{J}(\psi) & \mathbf{0} \end{bmatrix} \end{aligned} \tag{3.70}$$

The quantity $\mathbf{A_r r} - \dot{\mathbf{r}}$ then becomes:

$$\mathbf{A_r r} - \dot{\mathbf{r}} = \begin{bmatrix} \left(\mathbf{a}(\boldsymbol{\nu}) - \mathbf{b}\mathbf{b}^{(inv)}\mathbf{a}(\boldsymbol{\nu})\right)\boldsymbol{\nu_r} \\ \mathbf{A_{22}}(\mathbf{r})\boldsymbol{\eta_e} \end{bmatrix}. \tag{3.71}$$

The lower half of (3.71) is merged into $\mathbf{A_{clr}}$ and $\mathbf{A_0}$. The remainder is defined as $\mathbf{d}(\boldsymbol{\nu_r})$. As before, we assume that the velocity errors are small and $\mathbf{a}(\boldsymbol{\nu}) \approx \mathbf{a}(\boldsymbol{\nu_r})$.

$$\mathbf{d}(\boldsymbol{\nu_r}) \equiv \begin{bmatrix} \left(\mathbf{a}(\boldsymbol{\nu_r}) - \mathbf{b}\mathbf{b}^{(inv)}\mathbf{a}(\boldsymbol{\nu_r})\right)\boldsymbol{\nu_r} \\ \mathbf{0} \end{bmatrix} \tag{3.72}$$

The dynamics equations are updated to account for this additive constant term:

$$\begin{aligned} \dot{\mathbf{e}} &= \mathbf{A_{clr}}(\mathbf{x}, \mathbf{r})\,\mathbf{e} + \mathbf{d} + \mathbf{J_1}^T\mathbf{F} + \mathbf{w} \\ \dot{\mathbf{e}}_\mathbf{0} &= \mathbf{A_0}(\boldsymbol{\nu_r})\mathbf{e_0} + \mathbf{d} + \mathbf{J_1}^T\mathbf{F} + \mathbf{w} \end{aligned} \tag{3.73}$$

The updated mean error solution is shown below. The new $\mathbf{d}$ term is an additive constant, so the error variance is not affected by underactuation.

$$\begin{aligned} \bar{\mathbf{e}}_\mathbf{0}(t) &= e^{\mathbf{A_0}t}\bar{\mathbf{e}}_\mathbf{0}(0) + \mathbf{A_0}^{-1}\left(e^{\mathbf{A_0}t} - \mathbf{I}\right)\mathbf{d} \\ &\quad + \left(\mathbf{A_0} - \mathbf{S_1}^T\right)^{-1}\left(e^{\mathbf{A_0}t} - e^{\mathbf{S_1}^T t}\right)\mathbf{J_1}(0)^T\mathbf{F} \end{aligned} \tag{3.74}$$

3.6.6 Concatenating Maneuvers

So far the analysis of the mean error and the error variance has been restricted to single maneuvers in which the reference velocity $\boldsymbol{\nu}_\mathbf{r}$ is constant. The next step is to generalize the result to maneuver sequences by studying how the error is propagated through maneuver transitions. At the transitions, the initial conditions for the error statistics of the new maneuver must match the statistics at the end of the previous maneuver. The actual vehicle state $\mathbf{x}$ is constant from the moment before (+) to the moment after (−) the transition. The position reference $\boldsymbol{\eta}_\mathbf{r}$ is constant over the transition as well, and $\mathbf{J}_{\mathbf{2}_+} = \mathbf{J}_{\mathbf{2}_-}$. The reference-local initial conditions are derived below.

$$\begin{aligned}
\mathbf{x} \;=\; \mathbf{r}_+ + \mathbf{e}_+ &= \mathbf{r}_- + \mathbf{e}_- \\
\therefore\quad \mathbf{e}_+ &= \mathbf{e}_- - (\mathbf{r}_+ - \mathbf{r}_-) \\
\mathbf{e}_{\mathbf{0}+} &= \mathbf{J}_\mathbf{r}{}^T \mathbf{e}_+ \\
&= \mathbf{J}_\mathbf{r}{}^T \mathbf{e}_- - \mathbf{J}_\mathbf{r}{}^T \begin{bmatrix} \boldsymbol{\nu}_{\mathbf{r}+} - \boldsymbol{\nu}_{\mathbf{r}-} \\ \mathbf{0} \end{bmatrix} \\
&= \mathbf{e}_{\mathbf{0}-} - (\mathbf{r}_+ - \mathbf{r}_-) \\
\bar{\mathbf{e}}_{\mathbf{0}+} &= \bar{\mathbf{e}}_{\mathbf{0}-} - (\mathbf{r}_+ - \mathbf{r}_-)
\end{aligned} \tag{3.75}$$

Next we consider the reference-local error covariance matrix before and after the transition.

$$\begin{aligned}
\boldsymbol{\Sigma}_{\mathbf{0}-} &= \mathbf{J}_\mathbf{r}{}^T E\left[(\mathbf{x} - \mathbf{r}_- - \bar{\mathbf{e}}_-)(\mathbf{x} - \mathbf{r}_- - \bar{\mathbf{e}}_-)^T\right] \mathbf{J}_\mathbf{r} \\
\boldsymbol{\Sigma}_{\mathbf{0}+} &= \mathbf{J}_\mathbf{r}{}^T E\left[(\mathbf{x} - \mathbf{r}_+ - \bar{\mathbf{e}}_+)(\mathbf{x} - \mathbf{r}_+ - \bar{\mathbf{e}}_+)^T\right] \mathbf{J}_\mathbf{r}
\end{aligned} \tag{3.76}$$

Plugging in the result from (3.75) reveals that $\boldsymbol{\Sigma}_{\mathbf{0}+} = \boldsymbol{\Sigma}_{\mathbf{0}-}$, meaning that the error covariance matrix does not change across the transition.

In the execution of the motion plan, the maneuver transitions need not be based purely on the expected duration of the maneuver. For example, when driving to a waypoint it makes sense to wait until the vehicle actually reaches the waypoint

before switching to the next maneuver, in case disturbances or modeling error cause the vehicle to arrive at the waypoint earlier or later than expected. This is simple to implement on the vehicle. Maneuvers `a`, `b`, `c` and `w` end when the along-track position error with respect to the planned end of the maneuver is zero, and maneuvers `d` and `f` end when the heading error with respect to the planned end of the maneuver is zero. Due to these termination criteria, the along-track position error is exactly zero at the end of maneuvers `a`, `b`, `c` and `w`, and the heading error is exactly zero at the end of maneuvers `d` and `f`. To account for this effect, the fourth row of $\bar{\mathbf{e}}_{\mathbf{0}}$ and the fourth row and column of $\boldsymbol{\Sigma}_{\mathbf{0}}$ (corresponding to the along-track position state) must be set to zero at the end of the maneuver in the former case, and the sixth row of $\bar{\mathbf{e}}_{\mathbf{0}}$ and the sixth row and column of $\boldsymbol{\Sigma}_{\mathbf{0}}$ (corresponding to the heading state) must be set to zero at the end of the maneuver in the latter case. Subsequently, the initial mean error at the beginning of the next maneuver is computed according to (3.75).

The procedure for evaluating the mean error and the error variance at t is computed using the procedure shown in Algorithm 5. Assume that t is in the m'th maneuver and the preceding maneuvers end at $t_0, t_1, \ldots, t_{m-1}$.

Algorithm 5 Procedure for computing the mean and variance of the state error.

Initialize $\bar{\mathbf{e}}_{\mathbf{0}}(t_0) = \mathbf{0}$ and $\boldsymbol{\Sigma}_{\mathbf{0}}(t_0) = \mathbf{0}$.
for $i = 1 : (m-1)$ **do**
  Compute $\bar{\mathbf{e}}_{\mathbf{0}}(t_i)$ from $\bar{\mathbf{e}}_{\mathbf{0}}(t_{i-1})$ using $\boldsymbol{\nu}_{\mathbf{r},i}$, $\mathbf{J}(\psi_r(t_i))$, and $\Delta t = t_i - t_{i-1}$ (3.74).
  Compute $\boldsymbol{\Sigma}_{\mathbf{0}}(t_i)$ from $\boldsymbol{\Sigma}_{\mathbf{0}}(t_{i-1})$ using $\boldsymbol{\nu}_{\mathbf{r},i}$, $\mathbf{J}(\psi_r(t_i))$, and $\Delta t = t_i - t_{i-1}$ (3.65).
  Modify $\bar{\mathbf{e}}_{\mathbf{0}}(t_i)$ according to the maneuver's termination criterion.
  Modify $\boldsymbol{\Sigma}_{\mathbf{0}}(t_i)$ according to the maneuver's termination criterion.
end for
Compute $\bar{\mathbf{e}}_{\mathbf{0}}(t)$ from $\bar{\mathbf{e}}_{\mathbf{0}}(t_{m-1})$ using $\boldsymbol{\nu}_{\mathbf{r},m}$, $\mathbf{J}(\psi_r(t))$, and $\Delta t = t - t_{m-1}$ (3.74).
Compute $\boldsymbol{\Sigma}_{\mathbf{0}}(t)$ from $\boldsymbol{\Sigma}_{\mathbf{0}}(t_{m-1})$ using $\boldsymbol{\nu}_{\mathbf{r},m}$, $\mathbf{J}(\psi_r(t))$, and $\Delta t = t - t_{m-1}$ (3.65).
Compute $\bar{\mathbf{e}}(t)$ from $\bar{\mathbf{e}}_{\mathbf{0}}(t)$ using $\mathbf{J}(\psi_r(t))$ (3.58).
Compute $\boldsymbol{\Sigma}(t)$ from $\boldsymbol{\Sigma}_{\mathbf{0}}(t)$ using $\mathbf{J}(\psi_r(t))$ (3.66).

3.6.7 Simple Example

A simple demonstration of the prediction of the mean and variance of the path-following error is shown in Figure 3-10. The predicted growth and contraction of the

error distribution matches the result of a 1,000-point Monte Carlo simulation.

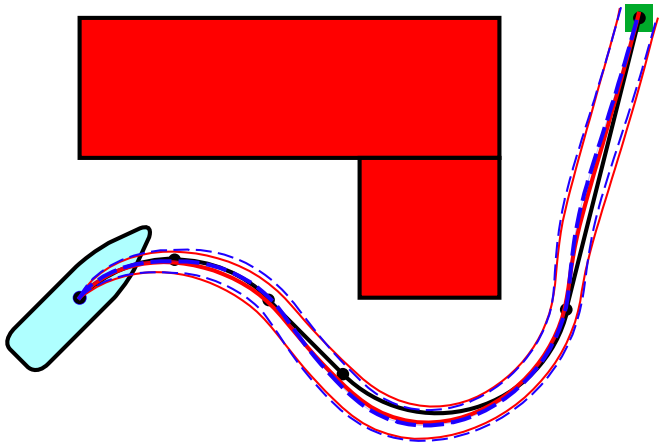

Figure 3-10: The mean error (thick lines) and standard deviation (thin lines) for the motion plan `eccbaw`. The nominal plan is shown in black; it brings the vehicle around the obstacles to the goal (the green square). The predicted mean and standard deviation are shown as dashed blue lines, and the results of a 1,000-point Monte Carlo simulation are shown as solid red lines.

Now that we can predict the error distribution as a function of time for any motion plan, we must consider how that error distribution affects the probability of hitting nearby obstacles. The next section covers how to predict the collision probability for each motion plan.

3.7 Collision Probability and Particle Absorption

The planner developed in this chapter constructs motion plans in the free configuration space, so that in the absence of disturbances or errors the vehicle is guaranteed to avoid all known obstacles. However, the actual vehicle trajectory may deviate from the nominal motion plan, as discussed in the previous section. If the nominal trajectory comes close to an obstacle, then disturbances may cause the vehicle to hit the obstacle. In this section we develop a prediction of the probability that these collisions will occur for any motion plan.

Predicting the collision probability for the vehicle given the mean error and the error variance is non-trivial. Consider a vehicle modeled as a particle driving next to

a wall. Because of correlations through time, if the particle has not hit the wall at time t, then it is nearly certain that it has not hit the wall in infinitesimal time later, $t + \epsilon$. However, as time progresses the probability of the particle hitting the wall becomes non-negligible due to stochastic disturbances. The evolution of the collision probability as the particle moves down the length of the wall is known as the particle absorption problem. In this section we attempt to solve this problem analytically.

Previous efforts to solve the particle absorption problem focus on specific cases that do not apply to a holonomic vehicle. One approach uses the Fokker-Planck equation to describe the evolution of the trajectory distribution, and the steady solution is found by solving four moment equations simultaneously [42]. Other solutions involve random walks on lattices [59, 3]. The closest match to our problem is found in [12], in which the particle absorption rate is computed for a randomly accelerated particle, but in that work the particle has no other dynamics. Our approach is described below.

In the particle absorption problem, the stochastic process may be a random walk, in which velocity is sampled from a normal distribution, or the output of a closed-loop control system that is affected by Gaussian disturbances. Because the driving disturbance is Gaussian and the system dynamics are locally linear, the resulting state distribution is also Gaussian. The evolution of the state variance can be obtained analytically by solving a Riccati equation, as in Section 3.6.

If the state vector for the system is $\mathbf{x}$ and the scalar output is $y = \mathbf{C}\mathbf{x}$, then the output follows a time-varying normal distribution, $y \sim N(\bar{y}(t), \Sigma_y(t))$. We consider the situation in which there is a constraint on the output space such that the system fails if $y(t) \geq d(t)$ for any t, or equivalently the system succeeds only if $\forall t,\ y(t) < d(t)$. Without a loss of generality, we assume that the constraint is always positive: $\forall t,\ d(t) > 0$. The constraint $d(t)$ may be thought of as the position of an obstacle relative to the nominal trajectory of a vehicle moving in a physical space. If the output $y(t)$ exceeds $d(t)$, then the vehicle has collided with the obstacle.

We are interested in predicting the probability that the system will succeed given the constraint; that is, we would like to know the fraction of trajectories that do

not violate the constraint, $P_a(t)$. Because the probability of success monotonically decreases as t increases ($\dot{P}_a \leq 0$), and trajectories that violate the constraint at t_1 do not need to be considered for $t > t_1$, this problem is known as *particle absorption*. In essence, the constraint absorbs the trajectories (particles) that violate it, and the success probability refers to the fraction of the original particles that have not yet been absorbed.

In Section 3.7.1 we introduce the state-space model that describes the system. The variance evolution is derived in Section 3.7.2. The PDF for the output with no constraints is presented in Section 3.7.4. In Section 3.7.5 we derive the PDF when the constraint is first applied. It is useful to consider the special case when the constraint is only applied at one point in time. In that case the constraint is called a *gate*. The PDF of the output a short time after passing the gate is studied in Section 3.7.6. A continuous constraint, a *wall*, can be modeled as a series of gates spaced infinitely close together. The wall constraint is studied in Section 3.7.7 for first-order systems and higher-order systems. We have developed an analytic solution for the higher-order case, but not the first-order case.

3.7.1 System Model

To ensure that the solution applies to as many different systems as possible, we use a very general system representation. The state vector $\mathbf{x}$ evolves through time as follows:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{G}\mathbf{w} \tag{3.77}$$

where $\mathbf{w}$ is a disturbance vector sampled from a zero-mean multinormal distribution whose covariance matrix is $\mathbf{W}$, $\mathbf{A}$ is the closed-loop dynamics matrix, and $\mathbf{G}$ maps the disturbance vector to the states. We are interested in only one output y, which is selected from $\mathbf{x}$ through the row vector $\mathbf{C}$.

$$y = \mathbf{C}\mathbf{x} \tag{3.78}$$

The noise vector **Gw** may have elements that are always zero, depending on the system. If **CGw** is nonzero, then the noise directly affects the output y. In that case, the lowest order of the system is first-order. However, if **CGw** is always zero, then the system is second-order or higher. In this section we define two example systems:

- **System 1**: A first-order system with $\mathbf{A} = -1$, $\mathbf{G} = 1$, $\mathbf{W} = 0.015$ and $\mathbf{C} = 1$. This system represents Brownian motion with feedback. The initial condition is $x(0) = 0$.

- **System 2**: A third-order system with:

$$\mathbf{A} = \begin{bmatrix} 1.04 & 1.20 & 0.50 & 0.85 \\ -4.18 & -4.01 & -1.56 & -2.66 \\ 1 & 0 & 0 & 0.25 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

  and $\mathbf{G} = \mathbf{I}_{4\times4}$, $\mathbf{W} = \text{diag}[0.001, 0.001, 0, 0]$ and $\mathbf{C} = [0, 0, 1, 0]$. This system represents a holonomic vehicle following a straight reference path under closed-loop control, with $x = [v, r, y, \psi]^T$. The eigenvalues of the closed-loop system are at -0.24, -0.31, and -1.21 $\pm$ 0.34, so the system is stable. Note that in this system **CGw** $= 0$. The disturbance affects the output state through two channels: a second-order channel (a lateral acceleration disturbance) and a third-order channel (a yaw acceleration disturbance).

3.7.2 Variance Evolution

We have assumed that the state and output vectors are shifted so that the expected value of the output is zero, $E[y(t)] = 0\ \forall t$. The state variance is $\mathbf{\Sigma} = E[\mathbf{x}\mathbf{x}^T]$, and it evolves through time according to the linear variance equation.

$$\dot{\mathbf{\Sigma}} = \mathbf{A}\mathbf{\Sigma} + \mathbf{\Sigma}\mathbf{A}^T + \mathbf{G}\mathbf{W}\mathbf{G}^T \tag{3.79}$$

Equation (3.79) can be solved analytically when **A** is constant using the solution

described in Section 3.6.4. If $\mathbf{A}$ is time-varying, then (3.79) must be integrated directly to obtain $\mathbf{\Sigma}(t)$. The variance of the output is:

$$\Sigma_y(t) = \mathbf{C}\mathbf{\Sigma}(t)\mathbf{C}^T. \tag{3.80}$$

Figure 3-11 shows a simulation of System 2 and its variance evolution. The predicted variance evolution from (3.80) matches the results from a 10,000-point Monte Carlo simulation. The convergence of the Monte Carlo simulation is also shown in Figure 3-11.

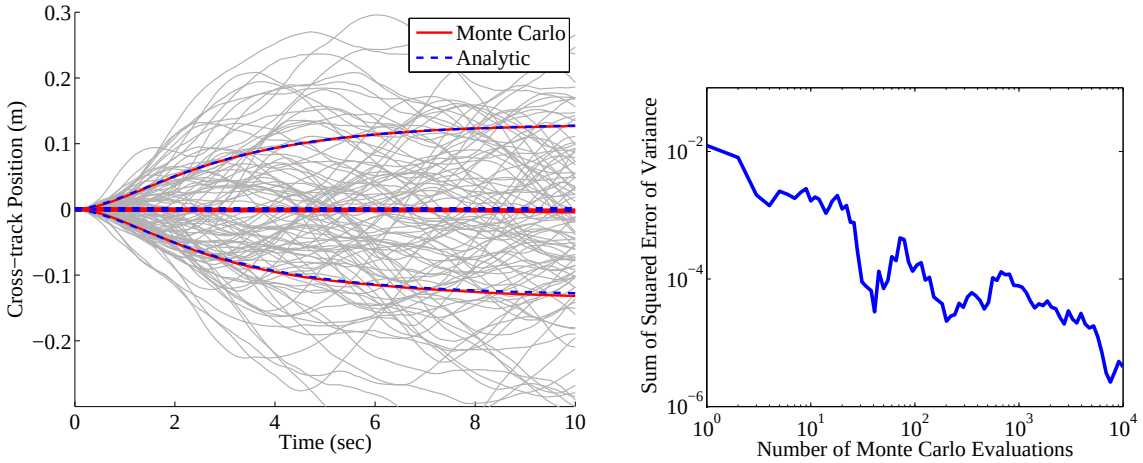

Figure 3-11: 100 trajectories from System 2. The predicted standard deviation (blue dashed lines) matches the measured standard deviation from 10,000 trajectories (red lines). The convergence of the sum of the squared cross-track variance error is shown on the right as a function of the number of Monte Carlo evaluations.

3.7.3 Summary of the Particle Absorption Solution

Here we summarize the prediction of the particle absorption rate that is developed over the remainder of this section. This result is only valid when $\mathbf{C}\mathbf{G}\mathbf{W}\mathbf{G}^T\mathbf{C}^T = 0$. The probability that the particle has not violated the constraint after t seconds is $P_a(t)$. This quantity evolves according to the following time-varying differential

equation:

$$\dot{P}_a(t) = -C(t)P_a(t) \tag{3.81}$$

with the initial condition $P_a(0) = 1$. This equation can be solved using Runge-Kutta integration, Euler integration, or any other method. The time-varying decay constant $C(t)$ is evaluated using the following equation. It is a function of the constraint $d(t)$ and its derivative $\dot{d}(t)$, as well as a number of system parameters.

$$C(t) = \frac{1}{n_0(t)\sqrt{2\pi\Sigma_y(t)}} \exp\left(-\frac{d(t)^2}{2\Sigma_y(t)}\right)\left[\sqrt{\frac{\Sigma_c(t)}{2\pi}} - \frac{\dot{d}(t)}{2}\left(1 - \operatorname{erf}\left(\frac{\dot{d}(t)}{\sqrt{\Sigma_c(t)}}\right)\right)\right] \tag{3.82}$$

The unconstrained output variance $\Sigma_y(t)$ is evaluated using (3.80), and $n_0(t)$ is the probability that $y(t) < d(t)$ if the output were unconstrained:

$$n_0(t) = \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{d(t)}{\sqrt{2\Sigma_y(t)}}\right)\right). \tag{3.83}$$

The remaining quantity in (3.82) is $\Sigma_c(t)$, which is evaluated as follows:

$$\Sigma_c(t) = \mathbf{C}\mathbf{A}\boldsymbol{\Sigma}(t)\mathbf{A}^T\mathbf{C}^T - \frac{\left(\mathbf{C}\boldsymbol{\Sigma}(t)\mathbf{A}^T\mathbf{C}^T\right)^2}{\Sigma_y(t)} \tag{3.84}$$

The overall procedure for evaluating the particle absorption evolution is listed in Algorithm 6. The remainder of this section is devoted to the derivation of this result.

Algorithm 6 Procedure for computing the particle absorption evolution.

1: Compute the state variance evolution $\boldsymbol{\Sigma}(t)$.
2: Compute the output variance evolution $\Sigma_y(t)$ using (3.80).
3: Compute $n_0(t)$ using (3.83).
4: Compute $\Sigma_c(t)$ using (3.84).
5: Compute $C(t)$ using (3.82).
6: Solve the time-varying differential equation (3.81) for $P_a(t)$, with the initial condition $P_a(0) = 1$.

3.7.4 Output PDF with No Constraints

In the absence of constraints ($\forall t,\ d(t) = \infty$), the state variance $\mathbf{\Sigma}(t)$ can be predicted, as a function of time, from the solution to (3.79) from arbitrary initial conditions $\mathbf{\Sigma}(0)$. The PDF for the cross-track position is $p(y)_{open}$, which is a zero-mean normal distribution whose variance is $\Sigma_y(t)$. For notational simplicity, we will only explicitly indicate the time dependence of the parameters where it is necessary to do so.

$$p(y)_{open} = \frac{1}{\sqrt{2\pi\Sigma_y}} \exp\left(-\frac{y^2}{2\Sigma_y}\right) \tag{3.85}$$

This PDF is compared with a histogram from the Monte Carlo simulation for System 2 in Figure 3-12.

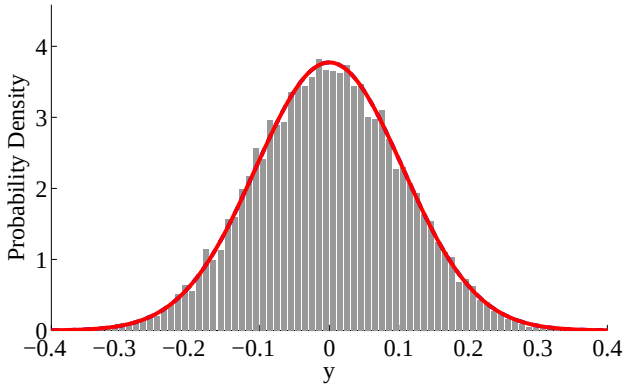

Figure 3-12: $p(y)_{open}$ for System 2 (Figure 3-11), evaluated at $t = 5$. The histogram represents a 10,000-point Monte Carlo simulation.

3.7.5 First Constraint Encounter

Consider a constraint that is only active after $t = t_0$: that is, $\forall t < t_0,\ d(t) = \infty$. Therefore $p(y)_{t<t_0} = p(y)_{open}$ from (3.85). At $t = t_0$, the constraint becomes active; at this moment, there is a finite probability that $y(t_0) \geq d(t_0)$, which violates the constraint. This failure probability is 1 minus the probability of success $P_a(t_0)$. The

probability of success is the CDF of (3.85) evaluated at $y(t_0) = d_0$, where $d_0 \equiv d(t_0)$:

$$\begin{aligned} P_a(t_0) \equiv n_0 &= P(y(t_0) < d_0) \\ &= \int_{-\infty}^{d_0} p(y)_{open} dy \Big|_{t=t_0} \\ &= \frac{1}{2}\left(1 + \text{erf}\left(\frac{d_0}{\sqrt{2\Sigma_y(t_0)}}\right)\right). \end{aligned} \tag{3.86}$$

At t_0, the distribution of particles that did not violate the constraint remains unchanged; however, their PDF is rescaled by the normalization factor $1/n_0$ so that its integral is 1.

$$p(y)_{t=t_0} = \begin{cases} \frac{1}{n_0\sqrt{2\pi\Sigma_y}} \exp\left(-\frac{y^2}{2\Sigma_y}\right) & \text{for } y < d_0 \\ 0 & \text{otherwise} \end{cases} \tag{3.87}$$

This distribution is plotted in Figure 3-13 for System 2 with $d_0 = 0.1$. In this example, the probability of not violating the constraint is $P_a(t_0) = n_0 = 0.828$.

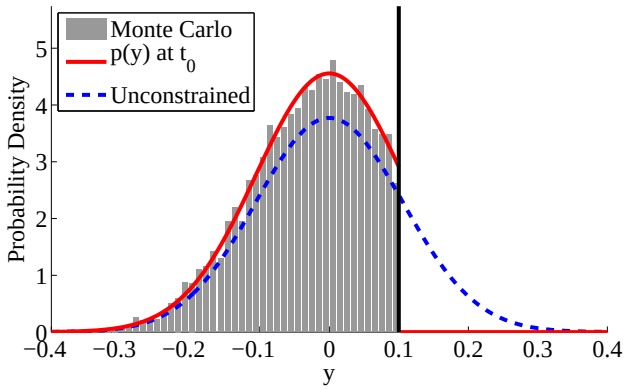

Figure 3-13: $p(y)_{t=t_0}$ for System 2 at $t_0 = 5$ with $d_0 = 0.1$. The success probability is $P_a(t_0) = 0.828$. The result is validated with a 10,000-point Monte Carlo simulation. The PDF just before t_0, $p(y)_{open}$, is the dashed blue line.

3.7.6 A Gate: An Infinitesimal-Duration Constraint

Consider the situation in which $d(t)$ has the following profile:

$$d(t) = \begin{cases} \infty & \text{for } t < t_0 \\ d_0 & \text{for } t = t_0 \\ \infty & \text{for } t > t_0 \end{cases} \tag{3.88}$$

In this case, the constraint can be thought of as a *gate* with an infinitesimal duration. An example of this situation is shown in Figures 3-14 and 3-15.

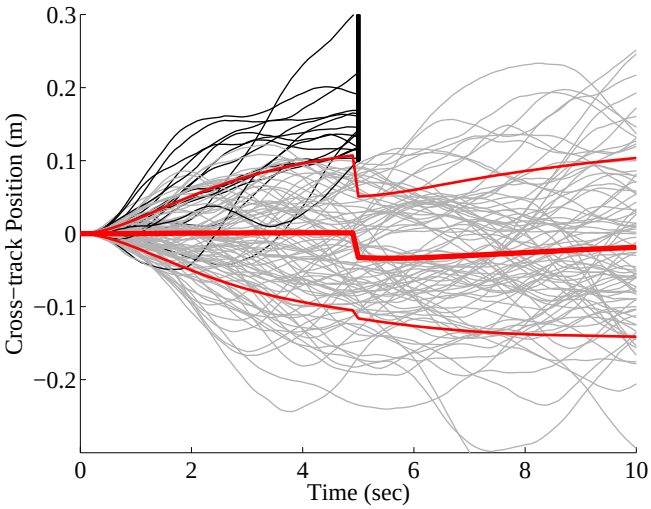

Figure 3-14: 100 Monte Carlo trajectories of System 2 encountering a gate with $d_0 = 0.1$ and an infinitesimal width (thick black line) at $t_0 = 5$. The dark trajectories hit the gate, while the others pass by successfully. The red curves show the measured mean and standard deviation of 10,000 successful trajectories.

For this constraint, $P_a(t)$ is a step function reducing from 1 to n_0 at $t = t_0$. We are interested in how the PDF of the successful particles (3.87) evolves a short time after passing the gate. The y dynamics are:

$$\dot{y} = \mathbf{C}\dot{\mathbf{x}} = \mathbf{CAx} + \mathbf{CGw} \equiv \delta \tag{3.89}$$

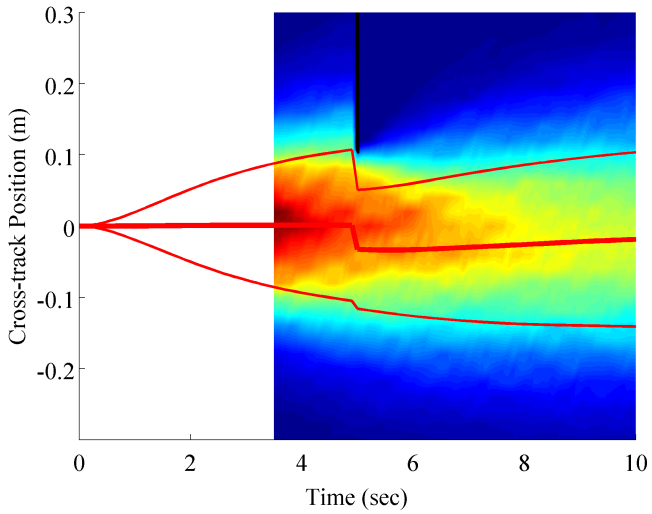

Figure 3-15: The same data as Figure 3-14, plotted as a histogram to illustrate the diffusion of the trajectories past d_0 after t_0.

The linear approximation to the value of $y(t)$ for $t = t_0 + \Delta t$ when Δt is small is:

$$\begin{aligned} y(t_0 + \Delta t) &= \mathbf{C}e^{\mathbf{A}\Delta t}\mathbf{x}(t_0) + \mathbf{CG}\mathbf{w}_{\Delta t} \\ &\approx \mathbf{C}(\mathbf{I} + \mathbf{A}\Delta t)\mathbf{x}(t_0) + \mathbf{CG}\mathbf{w}_{\Delta t} \\ &= y(t_0) + \mathbf{CA}\mathbf{x}(t_0)\Delta t + \mathbf{CG}\mathbf{w}_{\Delta t} \end{aligned} \tag{3.90}$$

where $\mathbf{w}_{\Delta t} \sim N(\mathbf{0}, \mathbf{W}\Delta t)$ is a Wiener process. For notational convenience, we define $\gamma \equiv y(t_0 + \Delta t)$, $\delta_t \equiv \mathbf{CA}\mathbf{x}(t_0)\Delta t + \mathbf{CG}\mathbf{w}_{\Delta t}$, and $y \equiv y(t_0)$. With these definitions, (3.90) can be rewritten as follows:

$$\gamma = y + \delta_t \tag{3.91}$$

The new variable γ is the linear prediction of the output Δt seconds after successfully passing the gate. We are interested in computing the PDF for γ, as that distribution is the key to understanding the particle absorption rate for a continuous constraint. First we define a new state vector $\mathbf{z} = [y, \delta_t]^T$. The mean value of the

new state is $\bar{\mathbf{z}} = \mathbf{0}$ when $d_0 = \infty$ and the variance is $\mathbf{\Sigma_z}$, which is derived below:

$$\mathbf{z} = \begin{bmatrix} \mathbf{Cx} \\ \mathbf{CAx}\Delta t + \mathbf{CGw}_{\Delta t} \end{bmatrix} \tag{3.92}$$

$$\begin{aligned} \mathbf{\Sigma_z} = E[\mathbf{z}\mathbf{z}^T] &= \begin{bmatrix} \mathbf{C\Sigma C}^T & \mathbf{C\Sigma A}^T\mathbf{C}^T\Delta t \\ \mathbf{CA\Sigma C}^T\Delta t & \mathbf{CA\Sigma A}^T\mathbf{C}^T\Delta t^2 + \mathbf{CGWG}^T\mathbf{C}^T\Delta t \end{bmatrix} \\ &\equiv \begin{bmatrix} \Sigma_y & \rho_t\sqrt{\Sigma_y\Sigma_{\delta t}} \\ \rho_t\sqrt{\Sigma_y\Sigma_{\delta t}} & \Sigma_{\delta t} \end{bmatrix} \end{aligned} \tag{3.93}$$

$$\text{where} \quad \Sigma_{\delta t} \equiv \mathbf{CA\Sigma A}^T\mathbf{C}^T\Delta t^2 + \mathbf{CGWG}^T\mathbf{C}^T\Delta t$$

$$\text{and} \quad \rho_t = \frac{\mathbf{C\Sigma A}^T\mathbf{C}^T\Delta t}{\sqrt{\Sigma_y\Sigma_{\delta t}}}$$

The correlation coefficient between the output y and the deviation after Δt seconds, δ_t, is ρ_t. Next we can make the following definitions:

$$\Sigma_{\delta x} \equiv \mathbf{CA\Sigma A}^T\mathbf{C}^T \tag{3.94}$$

$$\text{and} \quad \Sigma_{\delta w} \equiv \mathbf{CGWG}^T\mathbf{C}^T \tag{3.95}$$

$$\text{so} \quad \Sigma_{\delta t} = \Sigma_{\delta x}\Delta t^2 + \Sigma_{\delta w}\Delta t \tag{3.96}$$

$$\Sigma_{\delta\rho} \equiv \mathbf{C\Sigma A}^T\mathbf{C}^T \tag{3.97}$$

$$\text{so} \quad \rho_t = \frac{\Sigma_{\delta\rho}\Delta t}{\sqrt{\Sigma_y\Sigma_{\delta t}}} \tag{3.98}$$

For systems with feedback, $\Sigma_{\delta x} \neq 0$. For first-order systems such as Brownian motion, $\Sigma_{\delta w} \neq 0$, while for higher-order systems $\Sigma_{\delta w} = 0$. This distinction is important in the particle absorption analysis. The three system parameters $\Sigma_{\delta x}$, $\Sigma_{\delta w}$ and $\Sigma_{\delta\rho}$ may change through time, but they can be predicted from the solution to (3.79).

To evaluate the PDF for γ, we need to know the joint distribution $p(y, \delta_t)_{t=t_0}$. To calculate this PDF we first need to know the joint probability distribution between y

and δ_t when $d_0 = \infty$, $p(y, \delta_t)_{open}$:

$$\begin{aligned} p(y,\delta_t)_{open} &= \frac{1}{2\pi\sqrt{\Sigma_y\Sigma_{\delta t}}\sqrt{1-{\rho_t}^2}} \exp\left(-\frac{1}{2(1-{\rho_t}^2)}\left(\frac{y^2}{\Sigma_y}+\frac{{\delta_t}^2}{\Sigma_{\delta t}}-\frac{2\rho_t y\delta_t}{\sqrt{\Sigma_y\Sigma_{\delta t}}}\right)\right) \\ &= \frac{1}{2\pi\sqrt{\Sigma_y\Sigma_{\delta t}}\sqrt{1-{\rho_t}^2}} \exp\left(-\frac{\left(y-\rho_t\delta_t\sqrt{\Sigma_y/\Sigma_{\delta t}}\right)^2}{2\Sigma_y(1-{\rho_t}^2)}-\frac{{\delta_t}^2}{2\Sigma_{\delta t}}\right) \\ &= \frac{1}{2\pi\sqrt{\Sigma_y\Sigma_{\delta t}}\sqrt{1-{\rho_t}^2}} \exp\left(-\frac{{\delta_t}^2}{2\Sigma_{\delta t}}\right) \exp\left(-\frac{\left(y-\rho_t\delta_t\sqrt{\Sigma_y/\Sigma_{\delta t}}\right)^2}{2\Sigma_y(1-{\rho_t}^2)}\right) \end{aligned} \tag{3.99}$$

Equation (3.99) could be rearranged to isolate y instead of δ_t, but isolating δ_t makes the following analysis easier. When $d_0 \neq \infty$, this distribution is truncated to $y < d_0$ with the normalization constant n_0.

$$p(y,\delta_t)_{t=t_0} = \begin{cases} \frac{1}{n_0\cdot 2\pi\sqrt{\Sigma_y\Sigma_{\delta t}}\sqrt{1-{\rho_t}^2}} \exp\left(-\frac{{\delta_t}^2}{2\Sigma_{\delta t}}\right) \exp\left(-\frac{\left(y-\rho_t\delta_t\sqrt{\Sigma_y/\Sigma_{\delta t}}\right)^2}{2\Sigma_y(1-{\rho_t}^2)}\right) & \text{for } y < d_0 \\ 0 & \text{otherwise} \end{cases} \tag{3.100}$$

From the joint PDF (3.100), we can compute the marginal distribution for δ_t by integrating (3.100) over y. Because $p(y, \delta_t)_{t=t_0}$ is truncated to $y < d_0$, we only need to integrate up to d_0.

$$\begin{aligned} p(\delta_t)_{t=t_0} &= \int_{-\infty}^{d_0} p(y,\delta_t)dy \\ &= \frac{1}{2n_0\sqrt{2\pi\Sigma_{\delta t}}} \exp\left(-\frac{{\delta_t}^2}{2\Sigma_{\delta t}}\right)\left(1+\operatorname{erf}\left(\frac{d_0-\rho_t\delta_t\sqrt{\Sigma_y/\Sigma_{\delta t}}}{\sqrt{2\Sigma_y(1-{\rho_t}^2)}}\right)\right) \end{aligned} \tag{3.101}$$

The PDF for δ_t is plotted in Figure 3-16 for System 1 with $\Delta t = 0.1$ seconds and $\rho_t = -0.218$. The PDF is nearly a zero-mean normal distribution despite the significant correlation between the states, so for simplicity we assume that δ_t is normally distributed. This is a necessary assumption to complete the derivation of the PDF

for γ.

$$p(\delta_t)_{t=t_0} \approx \frac{1}{n_0\sqrt{2\pi\Sigma_{\delta t}}} \exp\left(-\frac{{\delta_t}^2}{2\Sigma_{\delta t}}\right) \tag{3.102}$$

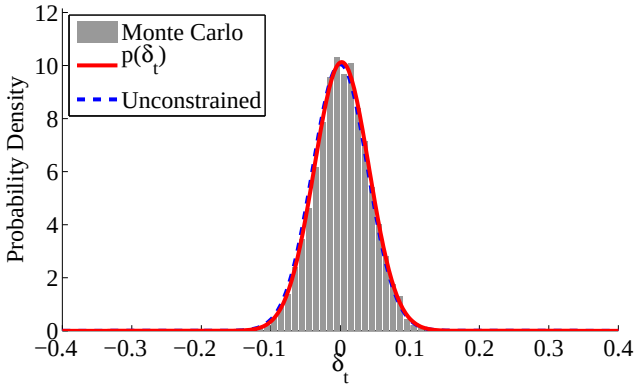

Figure 3-16: $p(\delta_t)_{t=t_0}$ for System 1 with $d_0 = 0.1$. The correlation coefficient is $\rho_t = -0.218$. The result is validated with a 10,000-point Monte Carlo simulation. The PDF just before t_0, $p(\delta_t)_{open}$, is the dashed blue line.

Using our knowledge of the distributions of $y(t_0)$ (3.87) and $\delta(t_0)$ (3.102), we can now evaluate the probability that $\gamma = y + \delta_t$ is less than some arbitrary value c, that is, we would like to compute the CDF $P(\gamma < c)$. The domain D defined by the inequality $\gamma < c$ is equivalent to the inequality $y < c - \delta_t$. $P(\gamma < c)$ is evaluated by integrating (3.100) over D.

$$P(\gamma < c) = \int_{D:\delta_t<c-y} p(y,\delta_t)dyd\delta_t = \int_{-\infty}^{\infty}\int_{-\infty}^{c-\delta_t} p(y,\delta_t)dyd\delta_t \tag{3.103}$$

It is actually easier to deal with the truncation of the normal distribution due to the gate (3.100) in the limits of integration, rather than in the integrand; in other words, integrating $\frac{1}{n_0}\,p(y,\delta_t)_{open}$ over the domain $y < d_0$ is equivalent to integrating $p(y,\delta_t)$ over the whole domain. Therefore, the upper limit of integration for y in (3.103) is the minimum of $c - \delta_t$ and d_0. The integral can be broken into two parts, A and B:

$$P(\gamma < c) = \underbrace{\int_{-\infty}^{c-d_0}\int_{-\infty}^{d_0} \frac{1}{n_0}\,p(y,\delta_t)_{open}dyd\delta_t}_{A\text{: if } c-\delta_t > d_0 \rightarrow \delta_t < c-d_0} + \underbrace{\int_{c-d_0}^{\infty}\int_{-\infty}^{c-\delta_t} \frac{1}{n_0}\,p(y,\delta_t)_{open}dyd\delta_t}_{B\text{: if } c-\delta_t < d_0 \rightarrow \delta_t > c-d_0} \tag{3.104}$$

We are interested in finding the PDF for γ, which is defined as follows:

$$\begin{aligned} p(\gamma = c) = \frac{\partial}{\partial c} P(\gamma < c) &= \frac{\partial}{\partial c} P(\gamma < c)_A + \frac{\partial}{\partial c} P(\gamma < c)_B \\ &\equiv p(\gamma = c)_A + p(\gamma = c)_B \end{aligned} \tag{3.105}$$

We consider the two parts of (3.105) separately, recalling the rules about taking the derivative of an integral with limits (an extension of the first fundamental theorem of calculus):

$$\begin{aligned} \text{If} \qquad F(x) &= \int_{a(x)}^{b(x)} f(x,z)dz \\ \text{then} \quad \frac{d}{dx}F(x) &= f(x,b(x))\frac{db}{dx} - f(x,a(x))\frac{da}{dx} + \int_{a(x)}^{b(x)} \frac{\partial}{\partial x} f(x,z)dz \end{aligned} \tag{3.106}$$

First we consider the A integral.

$$\begin{aligned} P(\gamma < c)_A &= \int_{-\infty}^{c-d_0} \int_{-\infty}^{d_0} \frac{1}{n_0}\, p(y,\delta_t)_{open} dy d\delta_t \\ &= \int_{-\infty}^{c-d_0} \int_{-\infty}^{d_0} \frac{1}{n_0 \cdot 2\pi\sqrt{\Sigma_y \Sigma_{\delta t}}\sqrt{1-{\rho_t}^2}} \exp\left(-\frac{{\delta_t}^2}{2\Sigma_{\delta t}}\right) \\ &\quad \times \quad \exp\left(-\frac{\left(y - \rho_t \delta_t \sqrt{\Sigma_y/\Sigma_{\delta_t}}\right)^2}{2\Sigma_y(1-{\rho_t}^2)}\right) dy d\delta_t \\ &= \int_{-\infty}^{c-d_0} \frac{1}{2n_0\sqrt{2\pi\Sigma_{\delta t}}} \exp\left(-\frac{{\delta_t}^2}{2\Sigma_{\delta t}}\right) \left(1 + \text{erf}\left(\frac{d_0 - \rho_t \delta_t \sqrt{\Sigma_y/\Sigma_{\delta t}}}{\sqrt{2\Sigma_y(1-{\rho_t}^2)}}\right)\right) d\delta_t \end{aligned} \tag{3.107}$$

In this case the integrand is not a function of c, so we use the unmodified first fundamental theorem of calculus to evaluate $p(\gamma = c)_A$:

$$p(\gamma = c)_A = \frac{1}{2n_0\sqrt{2\pi\Sigma_{\delta t}}} \exp\left(-\frac{(c-d_0)^2}{2\Sigma_{\delta t}}\right) \left(1 + \text{erf}\left(\frac{d_0 - (c-d_0)\rho_t \sqrt{\Sigma_y/\Sigma_{\delta t}}}{\sqrt{2\Sigma_y(1-{\rho_t}^2)}}\right)\right) \tag{3.108}$$

Next we consider the B integral.

$$\begin{aligned}
P(\gamma < c)_B &= \int_{c-d_0}^{\infty} \int_{-\infty}^{c-\delta_t} \frac{1}{n_0}\, p(y, \delta_t)_{open} dy d\delta_t \\
&= \int_{c-d_0}^{\infty} \int_{-\infty}^{c-\delta_t} \frac{1}{n_0 \cdot 2\pi\sqrt{\Sigma_y \Sigma_{\delta t}}\sqrt{1-{\rho_t}^2}} \exp\left(-\frac{{\delta_t}^2}{2\Sigma_{\delta t}}\right) \\
&\quad \times \quad \exp\left(-\frac{\left(y - \rho_t \delta_t \sqrt{\Sigma_y/\Sigma_{\delta_t}}\right)^2}{2\Sigma_y(1-{\rho_t}^2)}\right) dy d\delta_t \\
&= \int_{c-d_0}^{\infty} \frac{1}{2n_0\sqrt{2\pi\Sigma_{\delta_t}}} \exp\left(-\frac{{\delta_t}^2}{2\Sigma_{\delta t}}\right) \left(1 + \operatorname{erf}\left(\frac{c - \delta_t\left(1 + \rho_t\sqrt{\Sigma_y/\Sigma_{\delta t}}\right)}{\sqrt{2\Sigma_y(1-{\rho_t}^2)}}\right)\right) d\delta_t
\end{aligned} \tag{3.109}$$

In this case the integrand is a function of c, so we use (3.106) to evaluate $p(\gamma = c)_B$.

$$\begin{aligned}
p(\gamma = c)_B &= f(c, \infty)\cancelto{0}{\frac{d\infty}{dc}} - f(c, c-d_0)\cancelto{1}{\frac{d(c-d_0)}{dc}} + \int_{c-d_0}^{\infty} \frac{\partial}{\partial c} f(c, \delta_t) d\delta_t \\
&= -\frac{1}{2n_0\sqrt{2\pi\Sigma_{\delta t}}} \exp\left(-\frac{(c-d_0)^2}{2\Sigma_{\delta t}}\right) \left(1 + \operatorname{erf}\left(\frac{c - (c-d_0)\left(1 + \rho_t\sqrt{\Sigma_y/\Sigma_{\delta t}}\right)}{\sqrt{2\Sigma_y(1-{\rho_t}^2)}}\right)\right) \\
&\quad + \int_{c-d_0}^{\infty} \frac{1}{n_0\sqrt{2\pi\Sigma_{\delta t}}} \exp\left(-\frac{{\delta_t}^2}{2\Sigma_{\delta t}}\right) \frac{1}{\sqrt{2\pi\Sigma_y(1-{\rho_t}^2)}} \\
&\quad \times \quad \exp\left(-\frac{\left(c - \delta_t\left(1 + \rho_t\sqrt{\Sigma_y/\Sigma_{\delta t}}\right)\right)^2}{2\Sigma_y(1-{\rho_t}^2)}\right) d\delta_t
\end{aligned} \tag{3.110}$$

Now we note that $p(\gamma = c)_A$ and the first term of $p(\gamma = c)_B$ cancel each other out. That leaves us with the second term of $p(\gamma = c)_B$, which can be simplified to the following, with $r \equiv 1 + \rho_t\sqrt{\Sigma_y/\Sigma_{\delta t}}$.

$$\begin{aligned}
p(\gamma = c) &= \int_{c-d_0}^{\infty} \frac{1}{n_0|r|} \frac{1}{\sqrt{2\pi\Sigma_{\delta t}}} \exp\left(-\frac{{\delta_t}^2}{2\Sigma_{\delta t}}\right) \\
&\quad \times \quad \frac{1}{\sqrt{2\pi\Sigma_y(1-{\rho_t}^2)/r^2}} \exp\left(-\frac{(\delta_t - c/r)^2}{2\Sigma_y(1-{\rho_t}^2)/r^2}\right) d\delta_t
\end{aligned} \tag{3.111}$$

To further simplify (3.111), we note that the product of two Gaussians is proportional to another Gaussian:

$$\begin{aligned} N(a,A)\times N(b,B) &\iff \frac{1}{\sqrt{2\pi A}}\exp\left(-\frac{(x-a)^2}{2A}\right)\cdot\frac{1}{\sqrt{2\pi B}}\exp\left(-\frac{(x-b)^2}{2B}\right) \\ &= \frac{1}{2\pi\sqrt{AB}}\exp\left(-\frac{\left(x-\frac{Ba+Ab}{A+B}\right)^2}{2\frac{AB}{A+B}}\right)\exp\left(-\frac{(a-b)^2}{2(A+B)}\right) \end{aligned} \tag{3.112}$$

We apply this property to our problem with $a = 0$, $A = \Sigma_{\delta t}$, $b = c/r$ and $B = \Sigma_y(1-{\rho_t}^2)/r^2$. We also introduce a new quantity $\Sigma_\gamma \equiv \Sigma_{\delta t}r^2+\Sigma_y(1-{\rho_t}^2)$. Using these substitutions, we can write (3.111) as a single Gaussian distribution involving the variable of integration δ_t. Then it is a simple step to evaluate the integral.

$$\begin{aligned} p(\gamma=c) &= \int_{c-d_0}^{\infty}\frac{1}{n_0|r|\cdot 2\pi\sqrt{\Sigma_{\delta_t}\Sigma_y(1-{\rho_t}^2)/r^2}}\exp\left(-\frac{(c/r)^2}{2\left(\Sigma_{\delta t}+\Sigma_y(1-{\rho_t}^2)/r^2\right)}\right) \\ &\quad\times\quad \exp\left(-\frac{\left(\delta_t-\frac{\Sigma_{\delta_t}c/r}{\Sigma_{\delta t}+\Sigma_y(1-{\rho_t}^2)/r^2}\right)^2}{2\frac{\Sigma_{\delta t}\Sigma_y(1-{\rho_t}^2)/r^2}{\Sigma_{\delta t}+\Sigma_y(1-{\rho_t}^2)/r^2}}\right)d\delta \\ &= \int_{c-d_0}^{\infty}\frac{1}{n_0\cdot 2\pi\sqrt{\Sigma_{\delta t}\Sigma_y(1-{\rho_t}^2)}}\exp\left(-\frac{c^2}{2\Sigma_\gamma}\right)\exp\left(-\frac{(\delta_t-cr\Sigma_{\delta t}/\Sigma_\gamma)^2}{2\Sigma_{\delta t}\Sigma_y(1-{\rho_t}^2)/\Sigma_\gamma}\right)d\delta_t \\ &= \frac{1}{2n_0\sqrt{2\pi\Sigma_\gamma}}\exp\left(-\frac{c^2}{2\Sigma_\gamma}\right)\left(1-\mathrm{erf}\left(\frac{c-d_0-cr\Sigma_{\delta t}/\Sigma_\gamma}{\sqrt{2\Sigma_{\delta t}\Sigma_y(1-{\rho_t}^2)/\Sigma_\gamma}}\right)\right) \end{aligned} \tag{3.113}$$

Next we clean up (3.113) and replace c with γ to arrive at the final result.

$$p(\gamma) = \frac{1}{2n_0\sqrt{2\pi\Sigma_\gamma}}\exp\left(-\frac{\gamma^2}{2\Sigma_\gamma}\right)\left(1-\mathrm{erf}\left(\frac{\gamma\left(1-r\Sigma_{\delta t}/\Sigma_\gamma\right)-d_0}{\sqrt{2\Sigma_{\delta t}\Sigma_y(1-{\rho_t}^2)/\Sigma_\gamma}}\right)\right) \tag{3.114}$$

Finally, we can simplify the expression for Σ_γ:

$$\begin{aligned}\Sigma_\gamma &= \Sigma_{\delta t} r^2 + \Sigma_y (1 - {\rho_t}^2) \\ &= \Sigma_{\delta t} + 2\rho_t \sqrt{\Sigma_y \Sigma_{\delta t}} + \Sigma_y \\ &= \Sigma_y + (\Sigma_{\delta w} + 2\Sigma_{\delta \rho})\Delta t + \Sigma_{\delta x} \Delta t^2 \end{aligned} \tag{3.115}$$

The PDF for γ (3.114) is plotted in Figure 3-17 for System 1, $\Delta t = 0.1$ seconds after passing the gate. The PDF is verified by a Monte Carlo simulation of 10,000 trajectories. The area of the PDF that exceeds the constraint increases as the distribution is siphoned from the original truncated normal distribution; the amount of area lost to the left of the constraint always equals the amount of area gained to the right of the constraint.

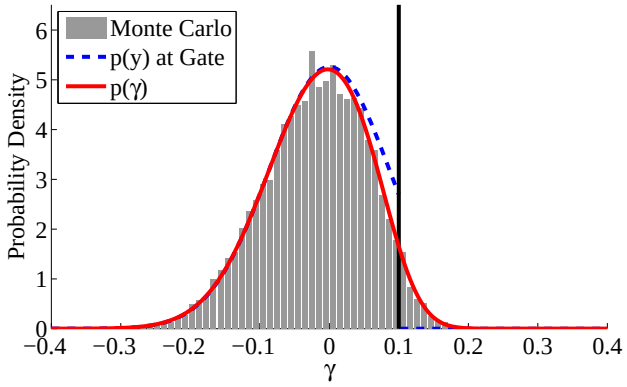

Figure 3-17: $p(\gamma)$ after $\Delta t = 0.1$ seconds for System 1 with $d_0 = 0.1$. The result is validated with a 10,000-point Monte Carlo simulation of $y(t_0 + \Delta t)$. The PDF at t_0, $p(y)_{t=t_0}$, is the dashed blue line.

3.7.7 Wall Diffusion

We now know how to compute the PDF for the trajectories a short time Δt after passing a gate, which is a constraint that only acts for a moment in time. Next we extend this result to a continuous constraint known as a *wall*. After Δt, the trajectories that violate the constraint $d(t + \Delta t)$ are absorbed into the wall. At any

time t, the probability that the vehicle has not yet hit the wall is $P_a(t)$. A short time later, at $t + \Delta t$, the survival probability is $P_a(t + \Delta t)$.

$$\begin{aligned} P_a(t + \Delta t) &= P_a(t) \times P(y(t + \Delta t) < d(t + \Delta t)) \\ &= P_a(t) \times P(\gamma(\Delta t) < d(t + \Delta t)) \end{aligned}$$

We are assuming that the distribution at any time is a truncated normal distribution. This assumption is required for the derivation of $p(\gamma)$ in Section 3.7.6. As time goes on, and more trajectories are absorbed by the wall, this assumption may falter. Nonetheless, it is a necessary assumption for this analysis.

The survival rate (the opposite of the collision rate) is $\dot{P}_a$:

$$\begin{aligned} \dot{P}_a &= \lim_{\Delta t \to 0} \frac{P_a(t + \Delta t) - P_a(t)}{\Delta t} \\ &= P_a \lim_{\Delta t \to 0} \frac{P(\gamma(\Delta t) < d(t + \Delta t)) - 1}{\Delta t} \\ &= -P_a \lim_{\Delta t \to 0} \frac{P(\gamma(\Delta t) \geq d(t + \Delta t))}{\Delta t} \\ &\equiv -CP_a \end{aligned} \tag{3.116}$$

For clarity we now define $F(t, \Delta t) = P(\gamma(\Delta t) \geq d(t + \Delta t))$. When $\Delta t = 0$, this expression is the probability that the particle is exceeding the constraint at t, so $F(t, 0) = 0$. Using the definition of the derivative, we can define C as follows:

$$\begin{aligned} C &= \lim_{\Delta t \to 0} \frac{F(t, \Delta t) - F(t, 0)}{\Delta t} \\ &= \frac{d}{d\Delta t} F(t, \Delta t)\Big|_{\Delta t = 0} \end{aligned} \tag{3.117}$$

The constant C is the diffusion rate of the probability distribution past the constraint at t, so it is evaluated at $\Delta t = 0$. For clarity we will not explicitly state that the evaluation occurs at $\Delta t = 0$, nor will we write the dependence of the parameters

on t, until it is necessary. Defining $d_t = d(t + \Delta t)$, we have:

$$\begin{aligned} C &= \frac{d}{d\Delta t}\int_{d_t}^{\infty} p(\gamma = c)\, dc \\ &= \frac{d}{d\Delta t}\int_{d_t}^{\infty} \frac{1}{2n_0\sqrt{2\pi\Sigma_\gamma}} \exp\left(-\frac{c^2}{2\Sigma_\gamma}\right)\left(1 - \mathrm{erf}\left(\frac{c(1 - r\Sigma_{\delta t}/\Sigma_\gamma) - d_0}{\sqrt{2\Sigma_{\delta t}\Sigma_y(1-{\rho_t}^2)/\Sigma_\gamma}}\right)\right) dc \\ &= \frac{d}{d\Delta t}\int_{d_t}^{\infty} f(\Delta t, c) dc \qquad (3.118) \end{aligned}$$

$$\text{where} \quad f(\Delta t, c) = p(\gamma(\Delta t) = c)$$

Using the definitions (3.96) and (3.98), we can rewrite r and some terms from (3.118) to highlight the dependence on Δt; this will be useful in understanding the limiting behavior of the equations.

$$\begin{aligned} r &= 1 + \rho_t\sqrt{\Sigma_y/\Sigma_{\delta t}} \\ &= 1 + \frac{\Sigma_{\delta\rho}\Delta t}{\Sigma_{\delta t}} \\ r\Sigma_{\delta t} &= \Sigma_{\delta x}\Delta t^2 + (\Sigma_{\delta w} + \Sigma_{\delta\rho})\Delta t \\ \Sigma_{\delta t}(1 - {\rho_t}^2) &= \Sigma_{\delta t}\left(1 - \frac{{\Sigma_{\delta\rho}}^2\Delta t^2}{\Sigma_y\Sigma_{\delta t}}\right) \\ &= \Sigma_{\delta x}\Delta t^2 + \Sigma_{\delta w}\Delta t - {\Sigma_{\delta\rho}}^2\Delta t^2/\Sigma_y \\ \text{and recall} \quad \Sigma_\gamma &= \Sigma_y + (\Sigma_{\delta w} + 2\Sigma_{\delta\rho})\Delta t + \Sigma_{\delta x}\Delta t^2 \end{aligned}$$

Equation (3.118) can be evaluated using the method described in (3.106).

$$\begin{aligned} C &= -f(\Delta t, d_t)\frac{dd_t}{d\Delta t} + \int_{d_t}^{\infty} \frac{d}{d\Delta t} f(\Delta t, c) dc \\ &\equiv C_1 + C_2 \qquad (3.119) \end{aligned}$$

The first term of (3.119) is easily evaluated at $\Delta t = 0$. We note that when $\Delta t \to 0$,

then $r\Sigma_{\delta t}/\Sigma_\gamma \to 0$ and $\Sigma_y/\Sigma_\gamma \to 1$.

$$\begin{aligned}
C_1 &= -f(\Delta t, d_t)\frac{dd_t}{d\Delta t}\bigg|_{\Delta t=0} \\
&= -\frac{\dot{d}}{2n_0\sqrt{2\pi\Sigma_y}}\exp\left(-\frac{{d_0}^2}{2\Sigma_y}\right)\left(1-\mathrm{erf}\left(\frac{d_t-d_0}{\sqrt{2\Sigma_{\delta t}(1-{\rho_t}^2)}}\right)\right)\bigg|_{\Delta t=0} \\
&= -\frac{\dot{d}}{2n_0\sqrt{2\pi\Sigma_y}}\exp\left(-\frac{{d_0}^2}{2\Sigma_y}\right)\left(1-\mathrm{erf}\left(\frac{\dot{d}\Delta t}{\sqrt{2\left((\Sigma_{\delta x}-{\Sigma_{\delta\rho}}^2/\Sigma_y)\Delta t^2+\Sigma_{\delta w}\Delta t\right)}}\right)\right)\bigg|_{\Delta t=0} \\
&= -\frac{\dot{d}}{2n_0\sqrt{2\pi\Sigma_y}}\exp\left(-\frac{{d_0}^2}{2\Sigma_y}\right)\left(1-\mathrm{erf}\left(\frac{\dot{d}}{\sqrt{2\left(\Sigma_{\delta x}-{\Sigma_{\delta\rho}}^2/\Sigma_y+\Sigma_{\delta w}/\Delta t\right)}}\right)\right)\bigg|_{\Delta t=0} \\
&= \begin{cases} -\frac{\dot{d}}{2n_0\sqrt{2\pi\Sigma_y}}\exp\left(-\frac{{d_0}^2}{2\Sigma_y}\right)\left(1-\mathrm{erf}\left(\frac{\dot{d}}{\sqrt{2\left(\Sigma_{\delta x}-{\Sigma_{\delta\rho}}^2/\Sigma_y\right)}}\right)\right) & \text{for } \Sigma_{\delta w}=0 \\ -\frac{\dot{d}}{2n_0\sqrt{2\pi\Sigma_y}}\exp\left(-\frac{{d_0}^2}{2\Sigma_y}\right) & \text{otherwise} \end{cases} \qquad (3.120)
\end{aligned}$$

(3.121)

Next we examine the second term in (3.119), using the rule for the derivative of a product.

$$\begin{aligned}
C_2 &= \int_{d_t}^{\infty}\frac{d}{d\Delta t}f(\Delta t, c)dc\bigg|_{\Delta t=0} \\
&= \int_{d_t}^{\infty}\frac{d}{d\Delta t}\left(\frac{1}{2n_0\sqrt{2\pi\Sigma_\gamma}}\exp\left(-\frac{c^2}{2\Sigma_\gamma}\right)\right)\times\left(1-\mathrm{erf}\left(\frac{c(1-r\Sigma_{\delta t}/\Sigma_\gamma)-d_0}{\sqrt{2\Sigma_{\delta t}\Sigma_y(1-{\rho_t}^2)/\Sigma_\gamma}}\right)\right)dc\bigg|_{\Delta t=0} \\
&\quad -\int_{d_t}^{\infty}\frac{1}{2n_0\sqrt{2\pi\Sigma_\gamma}}\exp\left(-\frac{c^2}{2\Sigma_\gamma}\right)\times\frac{d}{d\Delta t}\,\mathrm{erf}\left(\frac{c(1-r\Sigma_{\delta t}/\Sigma_\gamma)-d_0}{\sqrt{2\Sigma_{\delta t}\Sigma_y(1-{\rho_t}^2)/\Sigma_\gamma}}\right)dc\bigg|_{\Delta t=0} \qquad (3.122)
\end{aligned}$$

We call the first integral C_{2a} and the second integral C_{2b}. To evaluate the first integral, we first look at the derivative of a normal distribution with respect to its variance:

$$\frac{d}{dA}\left(\frac{1}{\sqrt{2\pi A}}\exp\left(-\frac{x^2}{2A}\right)\right)=\frac{1}{2A\sqrt{2\pi A}}\exp\left(-\frac{x^2}{2A}\right)\left(\frac{x^2}{A}-1\right) \qquad (3.123)$$

Next, the derivative of Σ_γ with respect to Δt is:

$$\frac{d\Sigma_\gamma}{d\Delta t} = \Sigma_{\delta w} + 2\Sigma_{\delta\rho} + 2\Sigma_{\delta x}\Delta t \tag{3.124}$$

Using (3.123) and (3.124), we can write C_{2a} as follows:

$$\begin{aligned} C_{2a} &= \int_{d_t}^{\infty} \frac{1}{4n_0\Sigma_\gamma\sqrt{2\pi\Sigma_\gamma}} \exp\left(-\frac{c^2}{2\Sigma_\gamma}\right)\left(\frac{c^2}{\Sigma_\gamma} - 1\right)(\Sigma_{\delta w} + 2\Sigma_{\delta\rho} + 2\Sigma_{\delta x}\Delta t) \\ &\quad \times \left(1 - \operatorname{erf}\left(\frac{c(1 - r\Sigma_{\delta t}/\Sigma_\gamma) - d_0}{\sqrt{2\Sigma_{\delta t}\Sigma_y(1-{\rho_t}^2)/\Sigma_\gamma}}\right)\right) dc \Bigg|_{\Delta t=0} \end{aligned} \tag{3.125}$$

Because we have already taken the derivative with respect to Δt, we can now evaluate (3.125) at $\Delta t = 0$.

$$C_{2a} = \int_{d_0}^{\infty} \frac{\Sigma_{\delta w} + 2\Sigma_{\delta\rho}}{4n_0\Sigma_y\sqrt{2\pi\Sigma_y}} \exp\left(-\frac{c^2}{2\Sigma_y}\right)\left(\frac{c^2}{\Sigma_y} - 1\right)\left(1 - \operatorname{erf}\left(\frac{c - d_0}{0^+}\right)\right) dc \tag{3.126}$$

The last term of (3.126) is equal to 1 when $c = d_0$ and 0 whenever $c > d_0$. Because the integral of a function whose value is nonzero at only one point is zero, we know that $C_{2a} = 0$.

We now turn to the second integral in (3.122), C_{2b}. We shift the limits of integration by introducing a new variable $c' = c - d_0$. It is now possible to make the first-order approximation $\Sigma_\gamma \approx \Sigma_y$; this is an acceptable assumption when evaluating the derivative with respect to Δt at $\Delta t = 0$.

$$\begin{aligned} C_{2b} &= -\int_0^{\infty} \frac{1}{2n_0\sqrt{2\pi\Sigma_y}} \exp\left(-\frac{(c'+d_0)^2}{2\Sigma_y}\right) \times \frac{d}{d\Delta t} \operatorname{erf}\left(\frac{c' - (c'+d_0)r\Sigma_{\delta t}/\Sigma_y}{\sqrt{2\Sigma_{\delta t}(1-{\rho_t}^2)}}\right) dc' \Bigg|_{\Delta t=0} \\ &= -\int_0^{\infty} \frac{1}{2n_0\sqrt{2\pi\Sigma_y}} \exp\left(-\frac{(c'+d_0)^2}{2\Sigma_y}\right) \times \frac{d}{dz}\left(\operatorname{erf}\left(\frac{z}{\sqrt{2}}\right)\right) \frac{dz}{d\Delta t}\, dc' \Bigg|_{\Delta t=0} \\ &= -\int_0^{\infty} \frac{1}{n_0\sqrt{2\pi\Sigma_y}} \exp\left(-\frac{(c'+d_0)^2}{2\Sigma_y}\right) \times \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{z^2}{2}\right) \frac{dz}{d\Delta t}\, dc' \Bigg|_{\Delta t=0} \end{aligned} \tag{3.127}$$

In (3.127), we have introduced the following change of variables:

$$\begin{aligned} z &= \frac{c' - (c' + d_0) r \Sigma_{\delta t} / \Sigma_y}{\sqrt{\Sigma_{\delta t}(1 - {\rho_t}^2)}} \\ &\equiv \frac{A_z}{\sqrt{B_z}} \end{aligned} \tag{3.128}$$

A_z and B_z can be expanded to explicitly show the time dependence:

$$\begin{aligned} A_z &= c' - (c' + d_0) \left((\Sigma_{\delta w} + \Sigma_{\delta \rho}) \Delta t + \Sigma_{\delta x} \Delta t^2 \right) / \Sigma_y \\ B_z &= \Sigma_{\delta w} \Delta t + (\Sigma_{\delta x} - {\Sigma_{\delta \rho}}^2 / \Sigma_y) \Delta t^2 \end{aligned}$$

with:

$$\begin{aligned} \dot{A}_z &= -(c' + d_0) \left((\Sigma_{\delta w} + \Sigma_{\delta \rho}) + 2 \Sigma_{\delta x} \Delta t \right) / \Sigma_y \\ &= A_z / \Delta t - c' / \Delta t - (c' + d_0) \Sigma_{\delta x} \Delta t / \Sigma_y \\ \dot{B}_z &= \Sigma_{\delta w} + 2 (\Sigma_{\delta x} - {\Sigma_{\delta \rho}}^2 / \Sigma_y) \Delta t \\ &= 2 B_z / \Delta t - \Sigma_{\delta w} \end{aligned}$$

Note that when $\Delta t \to 0$, $A_z \to c'$ and $B_z \to 0$. That means that $z = \infty$ when $\Delta t = 0$ and $c' \neq 0$, but we need to be more careful as c' approaches zero. The derivative of z with respect to Δt is:

$$\begin{aligned} \frac{dz}{d\Delta t} &= \frac{\dot{A}_z}{\sqrt{B_z}} - \frac{A_z \dot{B}_z}{2 {B_z}^{3/2}} \\ &= \frac{A_z / \Delta t - c' / \Delta t - (c' + d_0) \Sigma_{\delta x} \Delta t / \Sigma_y}{\sqrt{B_z}} - \frac{2 A_z B_z / \Delta t - A_z \Sigma_{\delta w}}{2 B_z \sqrt{B_z}} \\ &= -\frac{c' / \Delta t + (c' + d_0) \Sigma_{\delta x} \Delta t / \Sigma_y}{\sqrt{B_z}} + \frac{A_z \Sigma_{\delta w}}{2 B_z \sqrt{B_z}} \\ &= \frac{-\left(c' / \Delta t + (c' + d_0) \Sigma_{\delta x} \Delta t / \Sigma_y \right) \sqrt{B_z} + \frac{1}{2} A_z \Sigma_{\delta w} / \sqrt{B_z}}{B_z} \end{aligned} \tag{3.129}$$

We are now forced to consider (3.129) in two separate cases: when $\Sigma_{\delta w} = 0$ (higher-order systems, with respect to $\mathbf{w}$) and when it is nonzero (first-order systems). In both cases we focus on the limit when Δt is small, because we will eventually evaluate

the result at $\Delta t = 0$. We take the first case first.

$$\begin{aligned}\left.\frac{dz}{d\Delta t}\right|_{\Sigma_{\delta w}=0} &\approx -\frac{(c'/\Delta t)\sqrt{(\Sigma_{\delta x} - {\Sigma_{\delta\rho}}^2/\Sigma_y)\Delta t^2}}{B_z} \\ &= -\frac{c'}{B_z}\sqrt{\Sigma_{\delta x} - {\Sigma_{\delta\rho}}^2/\Sigma_y} \\ &\equiv -\frac{c'}{B_z}\sqrt{\Sigma_c} \end{aligned} \tag{3.130}$$

In the second case, the Δt^2 terms go to zero faster than the Δt terms, so $A_z \to c'$ and $B_z \to \Sigma_{\delta w}\Delta t$.

$$\begin{aligned}\left.\frac{dz}{d\Delta t}\right|_{\Sigma_{\delta w}\neq 0} &\approx \frac{-(c'/\Delta t)\sqrt{\Sigma_{\delta w}\Delta t} + \frac{1}{2}c'\Sigma_{\delta w}/\sqrt{\Sigma_{\delta w}\Delta t}}{B_z} \\ &= -\frac{\frac{1}{2}c'\sqrt{\Sigma_{\delta w}}/\sqrt{\Delta t}}{B_z} \end{aligned} \tag{3.131}$$

Next we take note of the following identity.

$$\begin{aligned}\int \frac{x}{\sqrt{2\pi A}}\exp\left(-\frac{(x-a)^2}{2A}\right)dx &= -\frac{A}{\sqrt{2\pi A}}\exp\left(-\frac{(x-a)^2}{2A}\right) + \frac{a}{2}\operatorname{erf}\left(x - a\sqrt{2A}\right) \\ \text{so}\quad \int_0^\infty \frac{c'}{B_z}\exp\left(-\frac{(c')^2}{2B_z}\right)dc' &= 1 \end{aligned} \tag{3.132}$$

The integral in (3.132) approaches a Dirac delta function centered at 0, $D(c')$, as $\Delta t \to 0$ (causing $B_z \to 0$). With this substitution, we can write C_{2b} as:

$$C_{2b} = \begin{cases} \int_0^\infty \frac{1}{n_0\sqrt{2\pi\Sigma_y}}\exp\left(-\frac{(c'+d_0)^2}{2\Sigma_y}\right)\frac{\sqrt{\Sigma_c}}{\sqrt{2\pi}}D(c')\,dc' & \text{for } \Sigma_{\delta w} = 0 \\ \int_0^\infty \frac{1}{n_0\sqrt{2\pi\Sigma_y}}\exp\left(-\frac{(c'+d_0)^2}{2\Sigma_y}\right)\frac{\sqrt{\Sigma_{\delta w}}}{2\sqrt{2\pi\Delta t}}D(c')\,dc'\Big|_{\Delta t=0} & \text{otherwise} \end{cases} \tag{3.133}$$

Using the sifting property of the Dirac delta function, the integrals in (3.133) are equal to the value of the integrand evaluated at $c' = 0$.

$$C_{2b} = \begin{cases} \frac{1}{n_0\sqrt{2\pi\Sigma_y}}\exp\left(-\frac{{d_0}^2}{2\Sigma_y}\right)\frac{\sqrt{\Sigma_c}}{\sqrt{2\pi}} & \text{for } \Sigma_{\delta w} = 0 \\ \frac{1}{n_0\sqrt{2\pi\Sigma_y}}\exp\left(-\frac{{d_0}^2}{2\Sigma_y}\right)\frac{\sqrt{\Sigma_{\delta w}}}{2\sqrt{2\pi\Delta t}}\Big|_{\Delta t=0} & \text{otherwise} \end{cases} \tag{3.134}$$

The final value for C is the sum of C_1, C_{2a} (which is 0) and C_{2b}.

$$C = \begin{cases} \frac{1}{n_0\sqrt{2\pi\Sigma_y}} \exp\left(-\frac{{d_0}^2}{2\Sigma_y}\right)\left[\sqrt{\frac{\Sigma_c}{2\pi}} - \frac{\dot{d}}{2}\left(1 - \operatorname{erf}\left(\frac{\dot{d}}{\sqrt{\Sigma_c}}\right)\right)\right] & \text{for } \Sigma_{\delta w} = 0 \\ \frac{1}{n_0\sqrt{2\pi\Sigma_y}} \exp\left(-\frac{{d_0}^2}{2\Sigma_y}\right)\left[\frac{\sqrt{\Sigma_{\delta w}}}{2\sqrt{2\pi\Delta t}} - \frac{\dot{d}}{2}\right] & \text{otherwise} \end{cases} \tag{3.135}$$

Recall that $\Sigma_c = \Sigma_{\delta x} - {\Sigma_{\delta\rho}}^2/\Sigma_y$ as defined in (3.130). Clearly when $\Sigma_{\delta w} \neq 0$, which is the case when the disturbance directly affects the output channel, the solution for C is invalid because it involves the fraction $1/\sqrt{\Delta t}$ evaluated at $\Delta t \to 0$. As such, this method only applies to the higher-order systems in which the disturbance does not directly affect the output. The cause of this issue is that in the Wiener process, the state variance scales linearly with time, rather than quadratically; as $\Delta t \to 0$, the standard deviation of the noise diminishes as a square root function, which as an infinite slope at the origin.

For higher-order systems ($\Sigma_{\delta w} = 0$) the value of C from (3.135) is used in (3.116) to evaluate the collision probability as a function of time. C itself is a function of time, due to its dependence on $\mathbf{\Sigma}(t)$ and the wall distance $d(t)$. However, both of those quantities can be computed analytically and (3.116) can be integrated through time very quickly. Figure 3-18 shows a batch of trajectories moving past a sinusoidal constraint. The predicted and measured collision probabilities are shown in Figure 3-19. The prediction is accurate for a significant portion of the wall traversal. The reason for the growing error is the violation of the assumption that the PDF outside of the wall is a truncated normal distribution; as trajectories diffuse into the wall, the distribution of the surviving trajectories distorts away from a Gaussian. Nonetheless, this analytic prediction of the collision probability is a useful tool for estimating the risk associated with a motion plan.

3.7.8 Handling Multiple Obstacles with an Optimal Sample Time

For a general motion planning problem, there may be multiple obstacles that encroach on the reference path, and the distances to those obstacles are not likely described by

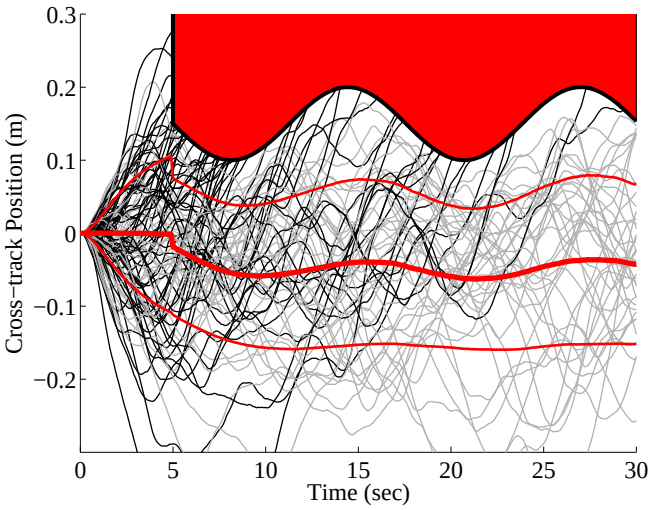

Figure 3-18: 100 Monte Carlo trajectories of System 2 encountering a sinusoidal constraint at $t_0 = 5$. The dark trajectories violate the constraint, while the others do not. The red curves show the measured mean and standard deviation of 10,000 successful trajectories.

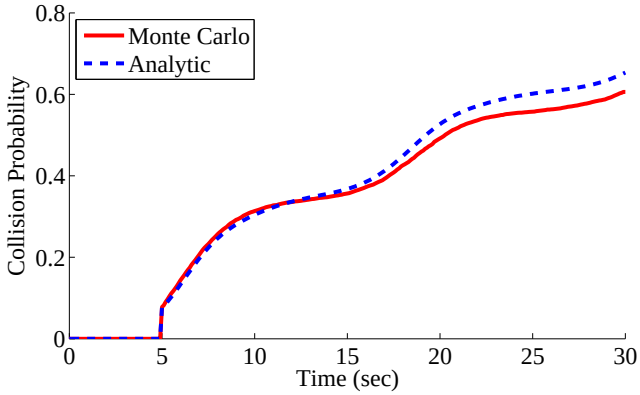

Figure 3-19: The predicted (dashed blue) and measured (red) collision probability for the example shown in Figure 3-18. The trends are accurate throughout the simulation, although some error eventually builds up as the normal distribution assumption breaks down.

analytic functions. To deal with this issue, we instead sample the obstacle locations every Δt_{obs} seconds throughout the motion plan and combine the effects of all of the obstacles found in each sample. That time period is the sample rate for which the method described below matches the analytic solution presented above for a fixed wall distance. Because the optimal sample rate depends on the wall distance, it is calculated for a wall distance d_0 equal to twice the cross-track error standard deviation when in the `b` trim. Motion plans that bring the vehicle closer than that distance to the obstacles would likely be rejected anyway, and obstacles farther than that distance have very small effects on the collision probability.

Recall what happens to the probability distribution just after passing a gate. The normal distribution is truncated at $y = d$; the fraction of the distribution that is truncated is the collision probability, P_{hit}. Afterwards, the remaining distribution spreads out past $y = d$ on the other side of the gate until it eventually forms a new normal distribution.

The variance prediction algorithm from Section 3.6.4 only works with normal distributions. The truncated distribution at the gate is not normal; however, we can approximate it as a normal distribution whose mean and variance are the same. The truncated distribution $p_d(y)$ is not strictly a probability density function because its total area is $P_{nohit} < 1$.

$$p_d(y) = \begin{cases} \frac{1}{\sqrt{2\pi V}} \exp\left(-\frac{(y-\bar{y})^2}{2V}\right) & \text{for } y \leq d \\ 0 & \text{otherwise} \end{cases} \tag{3.136}$$

The mean value $\bar{y}_d$ and variance V_d of this distribution are calculated as follows.

$$\begin{aligned} \bar{y}_d &= \int_{-\infty}^{\infty} p_d(y) y \, dy \\ &= \int_{-\infty}^{d} p(y) y \, dy \\ &= P_{nohit} \, \bar{y} - p(d) V \end{aligned} \tag{3.137}$$

$$
\begin{aligned}
V_d &= \int_{-\infty}^{\infty} p_d(y) y^2 \, dy \\
&= \int_{-\infty}^{d} p(y) y^2 \, dy \\
&= P_{nohit} \left(V + \bar{y}^2 \right) - p(d) V (d + \bar{y}) \qquad (3.138)
\end{aligned}
$$

The normal approximation to the truncated distribution, $p_{d,n}(y)$, is shown below and plotted in Figure 3-20.

$$
p_{d,n}(y) = \frac{1}{\sqrt{2\pi V_d}} \exp\left(-\frac{(y - \bar{y}_d)^2}{2V_d} \right) \qquad (3.139)
$$

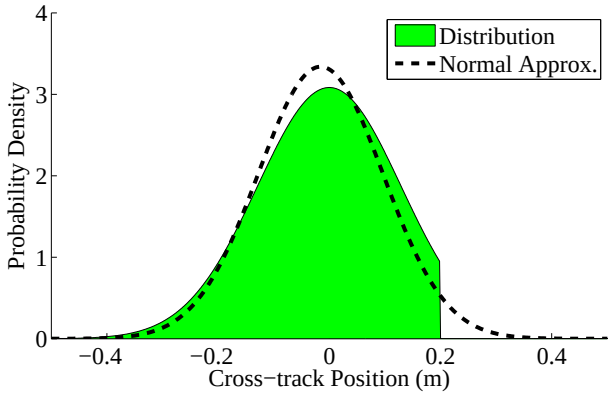

Figure 3-20: The probability distribution when the vehicle passes the gate, and the normal approximation using the same mean and variance.

When the vehicle passes an obstacle with a non-zero width, the collision probability increases with time (Figure 3-21). The obstacle can be represented as a series of gates with a spacing $\Delta t_{obs} U$. After passing each gate, the collision probability is computed using (3.86). Then the resulting distribution approximated as a normal distribution using (3.137-3.139). The new values for the mean error and the error variance are replaced in the appropriate matrices: $\bar{y}_d \rightarrow \bar{\mathbf{e}}_\mathbf{0}(5)$ and $V_d \rightarrow \mathbf{\Sigma_0}(5,5)$. The off-diagonal terms in the fifth row and column are also scaled by $\sqrt{V_d/V}$. Then the mean error and error variance prediction algorithms from Section 3.6 are run from one gate to the next.

Finally, if the probability of passing each gate k is $P_{nohit,k}$, then the overall collision

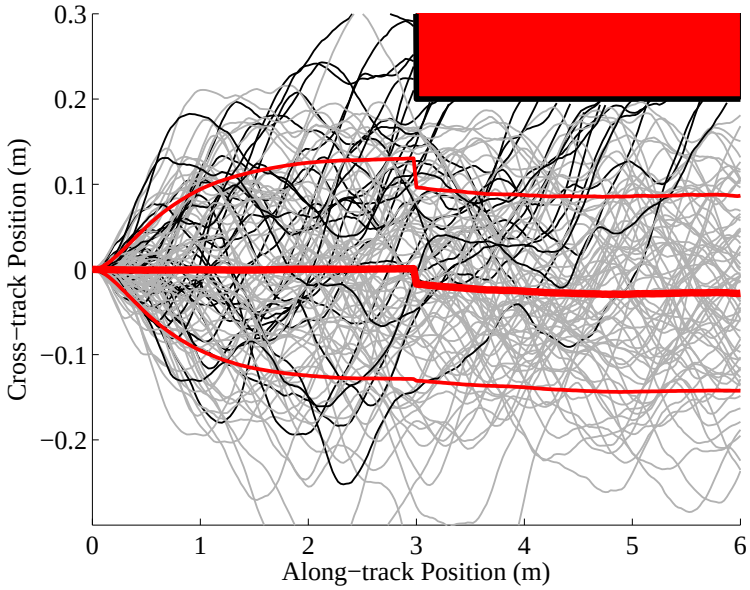

Figure 3-21: 100 trajectories from a 10,000-point Monte Carlo simulation of a path-following task near an obstacle. The trajectories that hit the obstacle are drawn as black lines. The red curves show the mean and standard deviation of the Monte Carlo trajectories.

probability for the obstacle is:

$$P_{hit} = 1 - \prod_k P_{nohit,k} \tag{3.140}$$

The optimal obstacle sample time Δt_{obs} is the time for which the diffusion into a wall as predicted by the particle absorption solution matches the diffusion as modeled by two gates spaced Δt_{obs} seconds apart. First the diffusion rate C is calculated using $\Sigma_y = V$, $\bar{y} = 0$, and $d_0 = 2\sqrt{V}$. Assuming the mean and variance of the distribution do not change significantly after passing the first gate, the probability of remaining collision-free at the second gate is:

$$P_{a,gates} = \frac{1}{2}\left(1 + \text{erf}\,\frac{d_0 - \bar{y}_d}{\sqrt{2V_d}}\right) \tag{3.141}$$

Meanwhile, the survival probability after Δt seconds using the particle absorption solution is equal to $e^{-C\Delta t_{obs}}$. We can equate this value with $P_{a,gates}$ and solve for

Δt_{obs}:

$$\Delta t_{obs} = -\frac{1}{C}\log\left(\frac{1}{2}\left(1 + \text{erf}\,\frac{d_0 - \bar{y}_d}{\sqrt{2V_d}}\right)\right) \tag{3.142}$$

For the system shown in Figure 3-21, $\Delta t_{obs} = 1.87$ seconds. The equivalent gates are shown in Figure 3-22 along with the predicted mean and standard deviation of the trajectories using the sampled obstacle approach. The error statistics using the sampled approach match the statistics from a 10,000-point Monte Carlo simulation very well. The collision probability for various gate spacings is shown in Figure 3-23. While the optimal gate spacing does not perfectly match the Monte Carlo collision probability, it is a convenient way to approximate the diffusion rate using only the system parameters. The optimal gate spacing does not strongly depend on the distance to the obstacle d.

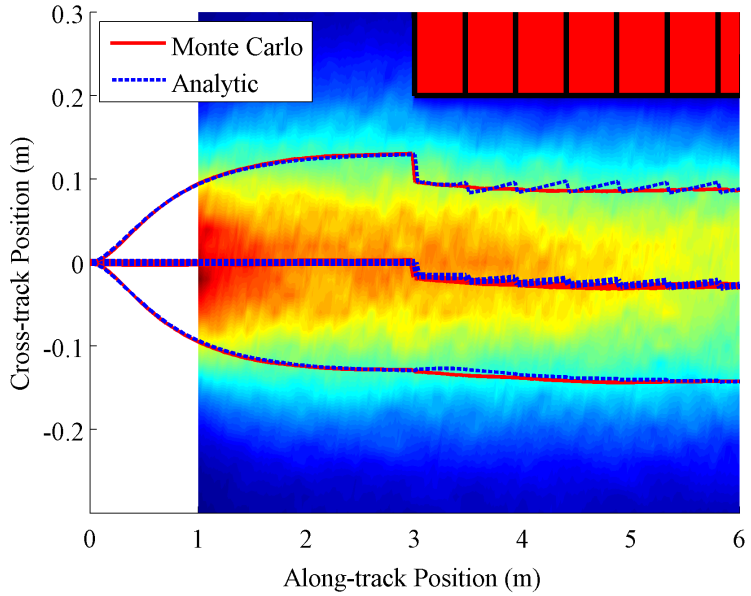

Figure 3-22: Probability density for the vehicle through time, as measured from a 10,000-point Monte Carlo simulation of a path-following task near an obstacle. The red curves show the mean and standard deviation from the Monte Carlo simulation, and the blue dashed curves show the mean and variance using the optimal gate spacing.

When there are multiple obstacles around the reference path, the effect of each obstacle must be taken into consideration at each sample time. Consider an obsta-

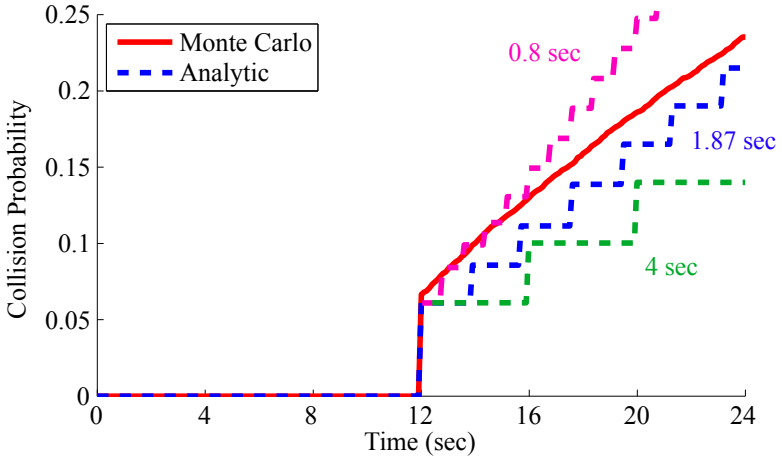

Figure 3-23: Collision probability for the simulation in Figure 3-21. The analytic solution approximates the collision probability best when the obstacle sample time is 1.87 seconds, which corresponds to a gate spacing of 0.467 meters. If the gate spacing is too close, then the predicted collision probability it too high, and if it is too loose than the predicted collision probability is too low.

cle whose nearest point is d meters away at an angle θ from the vehicle (in global coordinates). A rotation matrix for the line to the obstacle is:

$$\mathbf{J}_d = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{J}(\theta) \end{bmatrix} \tag{3.143}$$

The mean error vector in the direction of the obstacle is $\bar{\mathbf{e}}_d = {\mathbf{J}_d}^T \bar{\mathbf{e}}$ and the variance in that direction is $\mathbf{\Sigma}_d = {\mathbf{J}_d}^T \mathbf{\Sigma} \mathbf{J}_d$. The equivalent mean error from the earlier example is $\bar{y} = \bar{\mathbf{e}}_d(4)$ and the equivalent error variance is $V = \mathbf{\Sigma}_d(4,4)$. The collision probability P_{nohit}, the approximate mean error after the obstacle $\bar{y}_d$, and the approximate error variance after the obstacle V_d are computed from (3.137-3.138). The new values are replaced in the appropriate matrices: $\bar{y}_d \rightarrow \bar{\mathbf{e}}_d(4)$ and $V_d \rightarrow \mathbf{\Sigma}_d(4,4)$. The off-diagonal terms in the fourth row and column are also scaled by $\sqrt{V_d/V}$. Next the mean error and variance matrices are rotated back to global coordinates, $\mathbf{J}_d \bar{\mathbf{e}}_d \rightarrow \bar{\mathbf{e}}$ and $\mathbf{J}_d \mathbf{\Sigma}_d {\mathbf{J}_d}^T \rightarrow \mathbf{\Sigma}$. Finally, the error evolution algorithm from the previous section is run for $\Delta x/U$ seconds until the collision probabilities are computed again.

If there are N_{obs} obstacles and there are N_T time points Δt_{obs} apart, then the

overall collision probability is:

$$P_{hit} = 1 - \prod_{j=1}^{N_T} \prod_{k=1}^{N_{obs}} (1 - P_{nohit,j,k}) \tag{3.144}$$

where $P_{nohit,j,k}$ is the probability of passing the kth obstacle at the jth time point. The overall procedure for computing the collision probability for a motion plan is listed in Algorithm 7.

Algorithm 7 Procedure for computing the collision probability for a motion plan.

1: Calculate the diffusion rate C for the steady-state error statistics using a forward trim, with an effective obstacle distance equal to twice the standard deviation of the cross-track error (3.82).
2: Using C, compute the optimal sample time Δt_{obs} (3.142).
3: Discretize time throughout the motion plan t_j using Δt_{obs}, resulting in N_T time points.
4: **for** $j = 1 : N_T$ **do**
5: Compute the mean state error $\mathbf{e}(t_j)$ and the variance $\mathbf{\Sigma}(t_j)$.
6: **for** each obstacle k **do**
7: Calculate the distance d and the angle θ to the nearest point on the kth obstacle.
8: Calculate the mean and variance of the state in the θ direction, $\bar{y} = \bar{\mathbf{e}}_d(4)$ and $V = \mathbf{\Sigma}_d(4,4)$, respectively, using $\bar{\mathbf{e}}_d = {\mathbf{J}_d}^T \bar{\mathbf{e}}$ and $\mathbf{\Sigma}_d = {\mathbf{J}_d}^T \mathbf{\Sigma} \mathbf{J}_d$ with $\mathbf{J}_d$ defined in (3.143).
9: Compute the probability that the vehicle clears the kth obstacle at t_j, $P_{nohit,j,k}$, using d, $\bar{y}$, and V.
10: Compute the posterior mean $\bar{y}_d$ and variance V_d using (3.137-3.138).
11: Update the rotated mean and variance matrices $\bar{y}_d \rightarrow \bar{\mathbf{e}}_d(4)$ and $V_d \rightarrow \mathbf{\Sigma}_d(4,4)$, and scale the off-diagonal terms of the fourth row and column by $\sqrt{V_d/V}$.
12: Update the mean and variance matrices $\mathbf{J}_d \bar{\mathbf{e}}_d \rightarrow \bar{\mathbf{e}}$ and $\mathbf{J}_d \mathbf{\Sigma}_d {\mathbf{J}_d}^T \rightarrow \mathbf{\Sigma}$.
13: **end for**
14: **end for**
15: Evaluate the overall collision probability P_{hit} using (3.144).

3.8 Motion Planning with the A* Search Algorithm

In this chapter we have developed a motion planning framework and described how to predict the state error statistics (Algorithm 5) and the collision probability (Algorithm 7) for motion plans given stochastic disturbances. The final step is to find the optimal motion plan within that framework.

Motion plans can be generated using an expanding search algorithm such as the RRT algorithm [49] or the expanding A* search algorithm. The maneuvers included in the maneuver library $\mathcal{M}$ are the trim maneuvers, which hold the vehicle in a trim trajectory for a fixed duration. For example, $\mathcal{M}$ may be all of the trims in the trim library $\mathcal{T}$, held for 5 seconds each. The maneuvers $\mathbf{m}$ in $\mathcal{M}$ are evaluated using (3.20).

In this section we present an example of a motion planning task. The mission is to drive to the green circle in Figure 3-24 while avoiding the obstacles. Potential waypoints are automatically generated off the corners of the obstacles, with an X and Y offset equal to 1.5 times the width of the vehicle. With that offset, there is a full vehicle width between the obstacle and the edge of a vehicle positioned at the waypoint.

We use the expanding A* search algorithm to find the optimal motion plan. While the planning domain may be slightly more limited using the A* algorithm than an RRT with random speed control setpoints and trim durations, A* is deterministic and optimal within its domain. The A* algorithm requires a cost function g that is used to compare different plans, and a heuristic function h that is a prediction of the cost from the end of the plan to the goal. These two functions are described below.

3.8.1 Cost Function

We are interested in finding plans that are efficient, meaning that they get to the goal quickly, yet safe, meaning that the probability of colliding with an obstacle along the

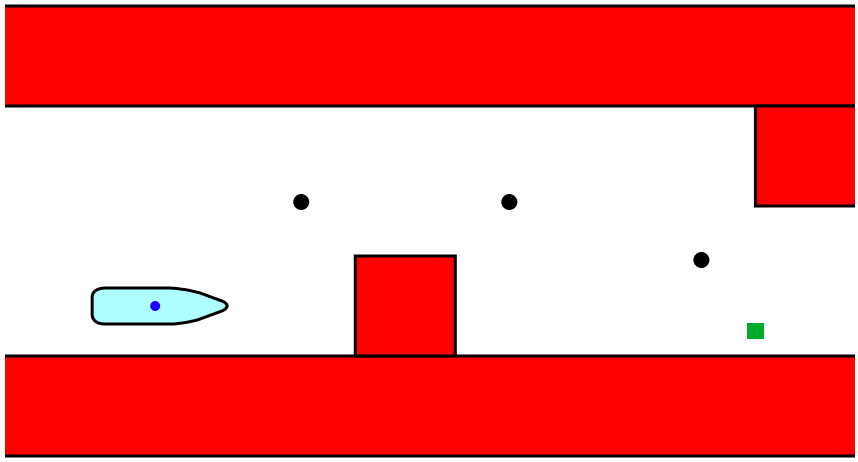

Figure 3-24: The planning space for a motion planning task. The mission is to drive to the goal point, indicated with a green square. The black circles are waypoints that may be used by the planner; they are automatically placed off the corners of the obstacles.

way is low. The cost function includes two components: the plan duration T and the collision probability P_{hit}.

Plan Duration

The total duration of the plan is calculated as follows. If the duration of maneuver i is t_i and there are $|p|$ maneuvers in the plan, then the duration of the plan is T:

$$T = \sum_{i=1}^{|p|} t_i \tag{3.145}$$

If the cost function includes T alone, then the planner will select the time-optimal plan. This plan will be guaranteed to be collision-free in the absence of all uncertainty and disturbances. However, while the plan will not intersect with obstacles, it may come dangerously close.

It may be desirable to encourage the planner to use waypoints. If a waypoint is 15 meters straight ahead of the vehicle, then the vehicle could either drive directly to the waypoint (one planning step, `w`) or it could use three 5-meter forward trims

(three planning steps, `bbb`). Using the waypoint increases the speed of the planning algorithm in this case. In addition, waypoints may be placed farther away from the obstacles in the free regions of the space; a path using the waypoint may no longer be time-optimal, but it may be safer. There is no difference in physical cost between driving to a waypoint and simply driving forward for the same length of time. If the effective cost of driving to a waypoint is scaled by a factor $c_w < 1$, then, all other factors being equal, the planner will choose to drive to a waypoint rather than using a forward trim. The modified duration is:

$$T = \sum_{i=1}^{|p|} \left\{ \begin{array}{ll} c_w t_i & \text{if } m_i = \texttt{w} \\ t_i & \text{otherwise} \end{array} \right\} \tag{3.146}$$

The modified duration (3.146) is identical to (3.145) when $c_w = 1$.

Collision Probability

The collision probability P_{hit} is calculated in Section 3.7. As a new maneuver is added to the motion plan, the gate collision probabilities only need to be calculated for the new leg, as opposed to the whole plan.

Combined Cost

The plan duration T (3.146) and the collision probability P_{hit} could be combined in a number of different ways. For example, the cost could be a linear combination of the two values, as shown below and in Figure 3-25.

$$g = T + K_{hit} P_{hit} \tag{3.147}$$

In (3.147), the constant K_{hit} scales the contribution of the collision probability; it is the additive time cost associated with a collision. However, this form of the cost function may not be desirable for two reasons. First, K_{hit} is a parameter that must be tuned; a certain amount of intuition is necessary to find an appropriate value for the parameter. Second, if the vehicle is nearly guaranteed to hit an obstacle (for

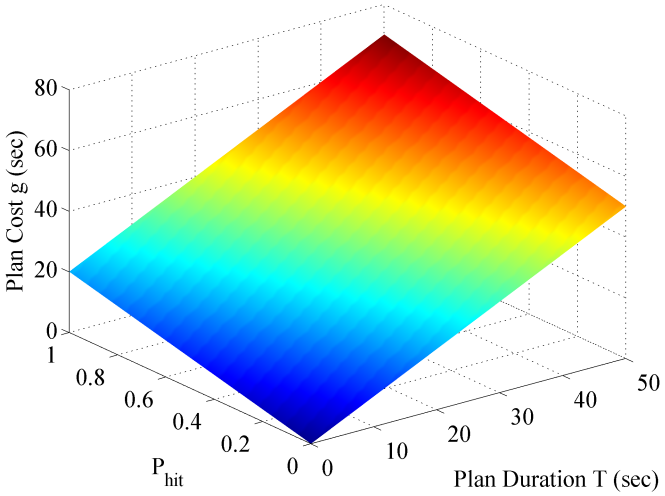

Figure 3-25: Cost function g (3.147), a linear combination of T and P_{hit}, with $K_{hit} = 20$.

example, the plan takes the vehicle through a corridor whose width is barely wider than the vehicle itself), then the plan cost is only increased by a finite amount (K_{hit}) over a plan of identical duration with $P_{hit} = 0$. This allows the planner to choose a very undesirable plan in certain situations.

An alternative formulation for g is based on the assumption that the vehicle must keep trying to execute the plan until it can do so without hitting any obstacles. If the vehicle continues to the end of the plan each time, despite the collision, then the expected time to complete the mission is:

$$g = \frac{T}{1 - P_{hit}} \tag{3.148}$$

This cost function (3.148) has the advantages that there are no tunable parameters and it exhibits nonlinear behavior, as shown in Figure 3-26. Plans with a high collision probability are strongly discouraged, and $g \to \infty$ as $P_{hit} \to 1$. Note that in both cases the units of the cost function are time (seconds).

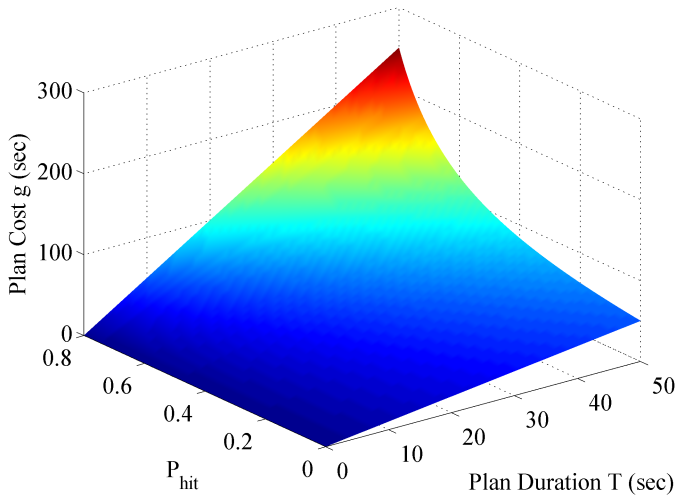

Figure 3-26: Cost function g (3.148): nonlinear combination of T and P_{hit}.

3.8.2 Heuristic Function

In order for the A* search algorithm to return the optimal plan from the motion planning framework (the plan with the lowest cost that ends at the goal), the heuristic function h must not overestimate the cost to get from the end of any plan to the goal. The search process takes less time (fewer plans evaluated) if h is very close to the actual cost to the goal, as opposed to a trivial heuristic such as $h = 0$.

A reasonable heuristic is the time it would take to drive straight to the goal, as the crow flies. This can be computed as the diagonal distance to the goal divided by the nominal forward speed of the vehicle. For example, if the goal is 10 meters away and the vehicle can drive at 0.25 m/sec, then $h = 40$ seconds.

However, the diagonal distance is not appropriate if there is an obstacle in the way, as the vehicle cannot drive through the obstacle. If there is an obstacle between the vehicle and the goal location, then the shortest path to the goal connects to obstacle corners with straight line segments. If a network graph connects the obstacle corners with the goal and the vehicle, then a better heuristic is the shortest path through the graph from the vehicle to the goal, divided by the forward speed of the vehicle.

Because the vehicle cannot drive exactly to the obstacle corners, the obstacles are

first expanded by half of the vehicle's width; the corners of these expanded obstacles are the nodes of the obstacle graph. The goal position is also added as a node. Straight edges are added between all of the nodes; edges that intersect with the expanded obstacles are discarded. The remaining set of nodes and edges forms the obstacle graph $\mathcal{D}$ (Figure 3-27). Dijkstra's algorithm [18] is run on this graph to find the minimum cost from every node n to the goal, C_n. This step is performed quickly offline before the planner runs. Online, when the planner is evaluating the predicted cost of a plan p, each node n of $\mathcal{D}$ is extended to the end of the plan with a straight line segment. The cost of each of these new edges c_n is the length of the edge divided by the forward speed of the vehicle. This cost is ∞ if the edge intersects an expanded obstacle. Finally, the heuristic is the minimum cost from the end of the plan to the goal, which is computed as follows:

$$h = \min_{n \in \mathcal{D}} (C_n + c_n) \tag{3.149}$$

Figure 3-27 shows the obstacle graph and the extension to the vehicle, along with the minimum-cost path through the graph. The obstacle graph for the example planning problem is shown in Figure 3-28.

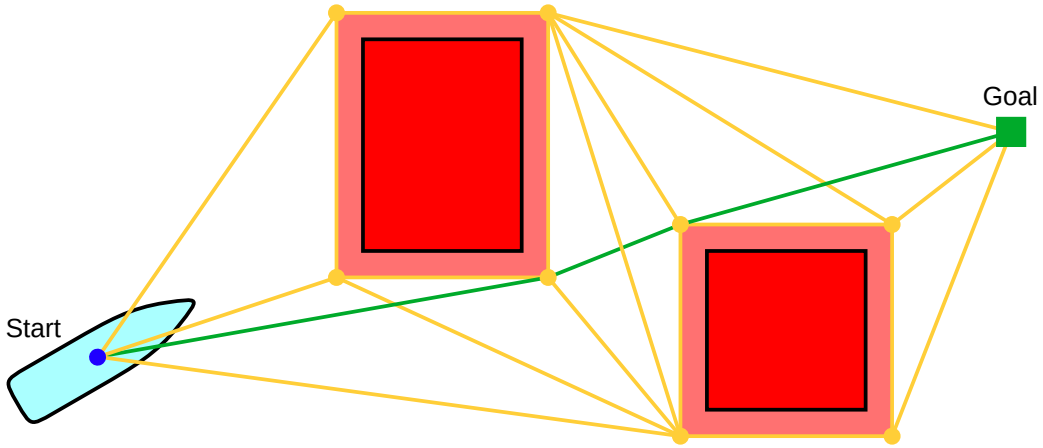

Figure 3-27: Obstacle graph $\mathcal{D}$ used to compute the A* heuristic h. The obstacles are expanded by half of the vehicle width, and the graph nodes include the corners of these expanded obstacles. The green path is the minimum-cost route through the graph; the cost of this path is h.

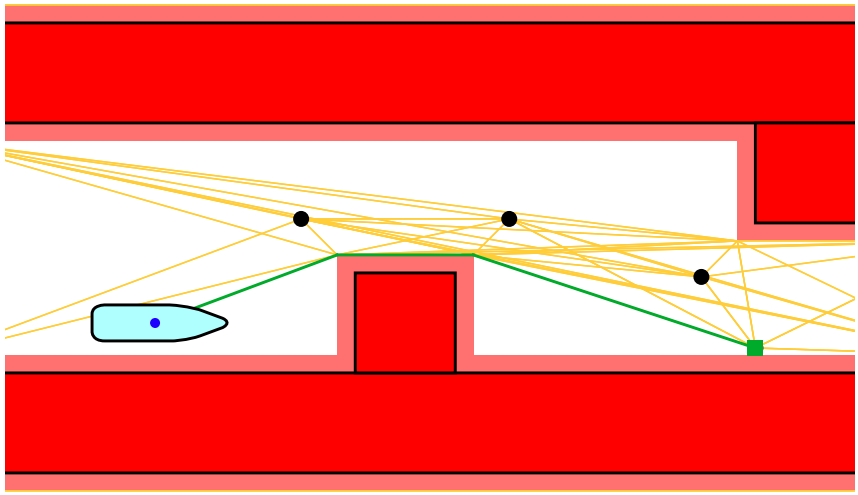

Figure 3-28: Obstacle graph for the example with $c_w = 0.9$. The green path is the minimum-cost route used for the heuristic.

If waypoints are used and $c_w < 1$, then the shortest path to the goal may include waypoints instead of obstacle corners. The cost used in the obstacle graph $\mathcal{D}$ is scaled by c_w for the legs that end at waypoints. In Figure 3-28, the waypoint cost factor is $c_w = 0.9$, which is not small enough to make the shortest path use the waypoints. In Figure 3-29, c_w is reduced to 0.5; in this case, she shortest path through the graph includes two of the waypoints.

3.8.3 Search Procedure

The expanding A* search algorithm finds the plan with the minimum cost g that ends at the goal (within a tolerance ϵ). The plans in the search queue are sorted by the predicted total cost $f = g + h$. The plan with the smallest f is expanded by adding all maneuvers and all waypoints that do not result in collisions.

The search procedure is listed in Algorithm 8. It differs from the basic A* algorithm (Algorithm 2) in that it generates the search tree on the fly and it sometimes adds multiple maneuvers at once, as when driving to a waypoint.

The search tree using Algorithm 8 is shown in Figure 3-30 along with the optimal plan `eidawcwca`. The vehicle backs away from the wall, turns in place, then uses two

Algorithm 8 A* Algorithm for motion planning.

Define the goal set $\mathcal{G} = \{\mathbf{X} \in \mathbb{R}^6 : |\mathbf{x} - \mathbf{x}_{goal}| < \epsilon\}$.
Load the obstacle file.
Load the trim library $\mathcal{T}$ and use it to construct the maneuver library $\mathcal{M}$.
Create a library of waypoints $\mathcal{W}$ around the obstacles, with one at the goal.
Create a network graph $\mathcal{D}$ around the obstacles. Use Dijkstra's algorithm to find the minimum collision-free distance from each node of the graph to the goal.
Create a plan p containing only the initial state $\mathbf{x}_0$.
Initialize the search queue $\mathcal{Q}$ with p.
while $|\mathcal{Q}| > 0$ **do**
  Remove the first plan from the queue: $p \leftarrow \mathcal{Q}(0)$.
  if $\mathbf{x}(p) \in \mathcal{G}$ **then**
    return p with success.
  end if
  for each maneuver m in $\mathcal{M}$ **do**
    Add m to p: $p' \leftarrow p + m$.
    if p' is collision-free **then**
      Compute the cost $g(p')$.
      Compute the predicted cost-to-go $h(p')$ by extending $\mathcal{D}$ to the current vehicle position and using Dijkstra's algorithm.
      The predicted total cost of p' is $f(p') = g(p') + h(p')$.
      Insert p' into $\mathcal{Q}$ sorted by $f(p')$.
    end if
  end for
  for each waypoint $w \in \mathcal{W}$ **do**
    Calculate the appropriate trim m_t to align the vehicle with w.
    Add m_t to p: $p' \leftarrow p + m_t$.
    if p' is collision-free **then**
      Calculate the maneuver m_w that ends at w.
      Add m_w to p': $p' \leftarrow p' + m_w$.
      if p' is collision-free **then**
        Compute the cost $g(p')$.
        Compute the predicted cost-to-go $h(p')$ by extending $\mathcal{D}$ to the current vehicle position and using Dijkstra's algorithm.
        The predicted total cost of p' is $f(p') = g(p') + h(p')$.
        Insert p' into $\mathcal{Q}$ sorted by $f(p')$.
      end if
    end if
  end for
end while
return with failure.

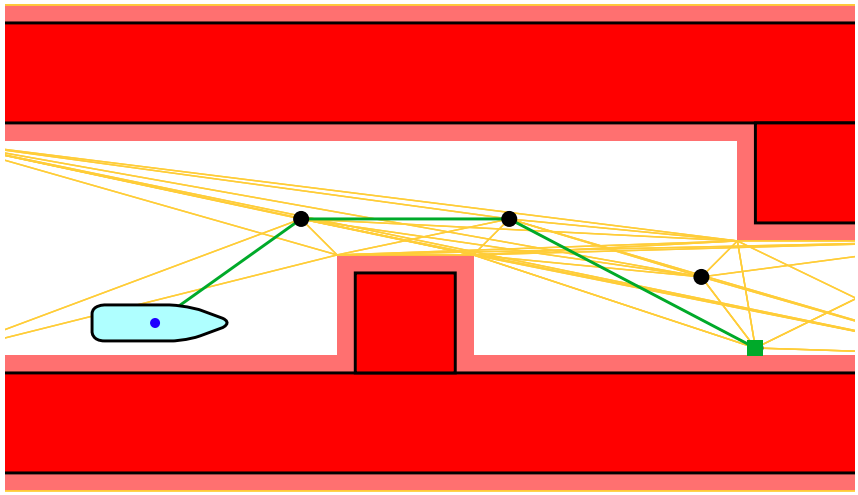

Figure 3-29: Obstacle graph for the example with $c_w = 0.5$. The minimum-cost path through the graph, shown in green, uses two of the waypoints.

waypoints to get around the obstacle before driving toward the goal. An extra node `e` is added to the plan to stop the vehicle. This plan was found after 33 iterations of the A* algorithm in 0.632 seconds on a 2.33 MHz Intel Core 2 Duo processor.

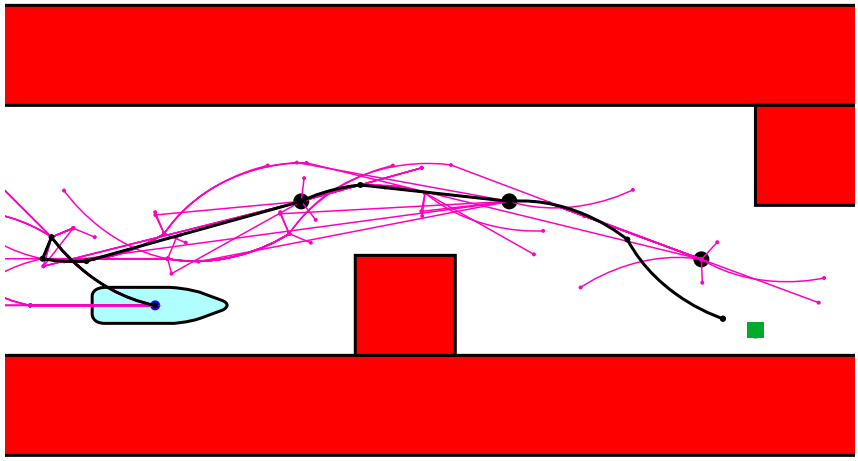

Figure 3-30: A* search tree (purple) and the optimal plan `eidawcwca` (black).

While the plan as drawn in Figure 3-30 is not smooth, the vehicle actually follows a smooth path. The nominal vehicle trajectory (evaluated from $\mathbf{r}$ and $\bar{\mathbf{e}}$) is shown in Figure 3-31 along with curves showing one standard deviation of cross-track position

error. Note how the error grows when driving in reverse or turning in place (no position feedback) and shrinks in the straight segments (position feedback enabled).

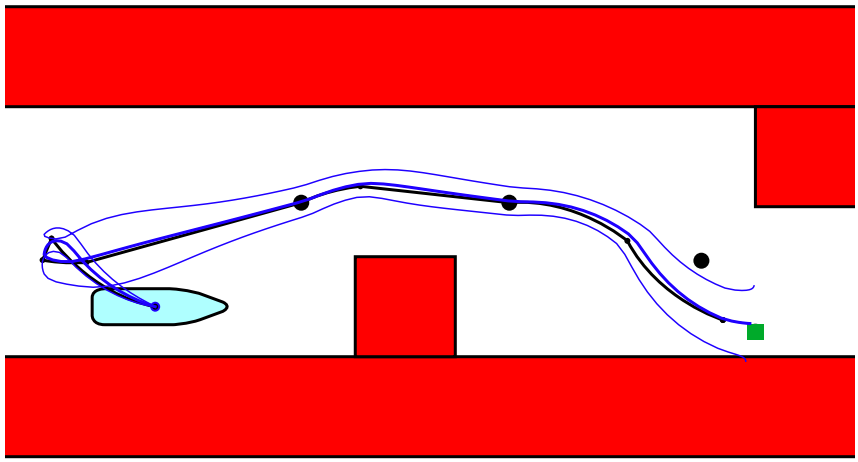

Figure 3-31: The optimal plan returned by the planner (thick black) and the nominal vehicle trajectory (thick blue). The thin blue lines show one standard deviation of cross-track position error.

The plan shown in Figure 3-31 was simulated with no disturbances. The resulting position at several times during the plan is shown in Figure 3-32.

A dramatic example of the effect of the collision probability on the planner's behavior is shown in Figure 3-33. In this example, the fastest route to the goal (taking 28 seconds) is a straight path between the obstacles. Because the passage is so narrow, the collision probability is 99.5%. The alternative plan `edbbcbcw`, shown in the figure with the corresponding search tree, drives around the obstacles with a collision probability of 1.5%. The duration of this plan is 40.3 seconds, but the planner has determined that the extra time is worth it to ensure the success of the mission.

3.9 Experimental Results

Several experiments were performed in the Towing Tank at the Massachusetts Institute of Technology using an underactuated autonomous surface vessel. The following

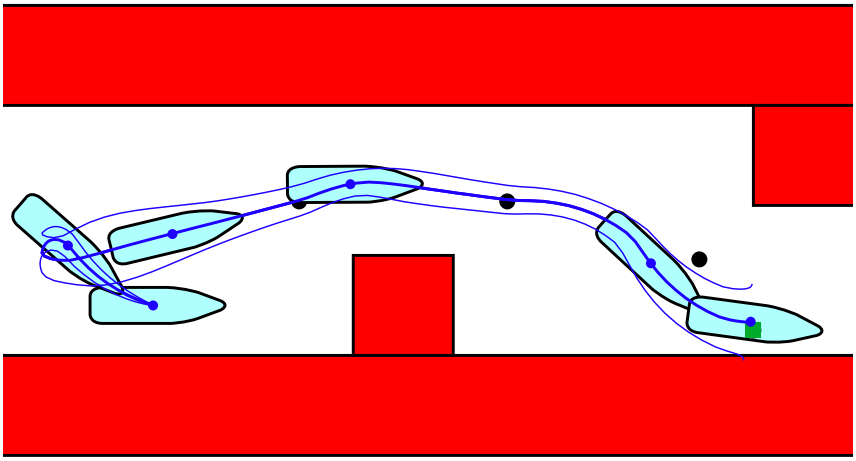

Figure 3-32: The vehicle position at several different times during the plan. These positions, determined from a simulation of the execution of the plan, match the predicted trajectory (the thick blue line) which incorporates the mean error $\bar{\mathbf{e}}$.

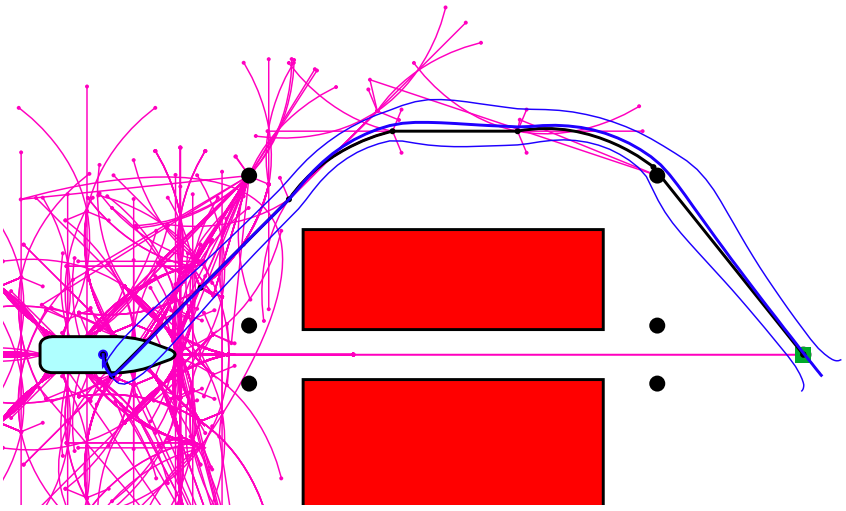

Figure 3-33: The planner chooses to drive around the obstacles instead of driving between them, increasing the plan duration by 44% but reducing the collision probability from 99.5% to 1.5%.

subsections describe the vehicle and the experiment setup.

3.9.1 Underactuated Surface Vessel

The vehicle used in the experiment is a 1.25-meter autonomous model of an icebreaking ship (Figure 3-34). The vessel is powered by a single azimuthing thruster under the stern which can generate a thrust vector in any direction in the horizontal plane. The details of the vessel are provided in Appendix A. A state space model for the vehicle dynamics, including the process noise due to the waves described in Section 3.9.3, was derived from a least-squares fit to input-output data from a series of simple system identification tests. The resulting model is included in the Appendix, Section A.6. The controller is a linear quadratic regulator designed around each control setpoint.

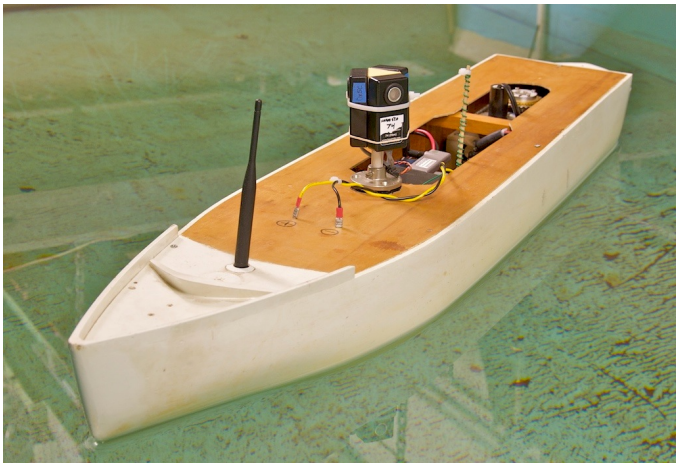

Figure 3-34: The 1.25-meter underactuated surface vessel used in the experiment.

3.9.2 Motion Plan

The planner (Algorithm 8) was used to find a path through the environment shown in Figure 3-35. The nominal forward speed is $U = 0.15$ m/sec and the nominal yaw rate is $R = 9°$/sec. There was no discounted cost for waypoints, so $c_w = 1$. The optimal plan `c(cw)c(aw)` uses two 5-second trim maneuvers `c` and two waypoint

maneuvers `cw` and `aw`. The duration of the plan is 32.36 seconds and the collision probability is 2.2%. This plan was found after 28 iterations of the search algorithm in 1.33 seconds on a 2.33 MHz Intel Core 2 Duo processor. If the planner does not consider the collision probability in the cost function, then the shortest motion plan drives straight between the two islands with a duration of 31.67 seconds and a collision probability of 11.4%; this plan is 2.1% shorter than the optimal plan but the vessel is over 5 times as likely to hit an obstacle. The optimal plan and the search tree are shown in Figure 3-35. The search tree is strongly directed toward the goal due to the A* heuristic function h.

3.9.3 Experiments

The optimal plan `c(cw)c(aw)` was executed by the autonomous surface vessel shown in Figure 3-34 while a wavemaker generated 2.4-Hz waves with a 2-cm wave height and a wavelength of 27 cm moving from left to right in Figure 3-35. The trajectories from five separate executions of the plan are plotted in the figure. For the most part, the trajectories remain within one standard deviation of the reference trajectory. Differences in the mean error can be attributed to modeling error and a slight unmodeled drift from left to right due to the net effects of the waves. Figure 3-36 shows the vehicle in the tank near the end of the motion plan.

3.10 Summary

In this chapter we have developed a robust motion planning algorithm for planar holonomic vehicles. The motion plans generated by the algorithm are robust to external stochastic disturbances because they include the predicted collision probability in the cost function. Despite the stochastic nature of the actual vehicle trajectories, we do not use Monte Carlo simulations to estimate the cost function. Rather, we have developed analytic predictions of the path-following error statistics and the resulting collision probability. These predictions can be evaluated very quickly for each motion plan so that the entire motion planning problem can be solved in just a few seconds.

The predicted error statistics and the predicted collision probability are based on an estimate of the vehicle parameters. The performance of the vehicle during the execution of the motion plan depends on the quality of the parameter estimates; the vehicle will behave as expected if the parameters are accurate, but there may be additional path-following errors if the parameter estimates are incorrect. The following two chapters describe how to learn the parameters online and how to predict the effects of that model learning on the path-following performance. Then the same planning algorithm will be used (Algorithm 8) but the cost function will include the effects of model uncertainty in addition to the stochastic disturbances. We will then see that this planner automatically chooses motion plans that actively reduce the model uncertainty to improve the overall mission performance.

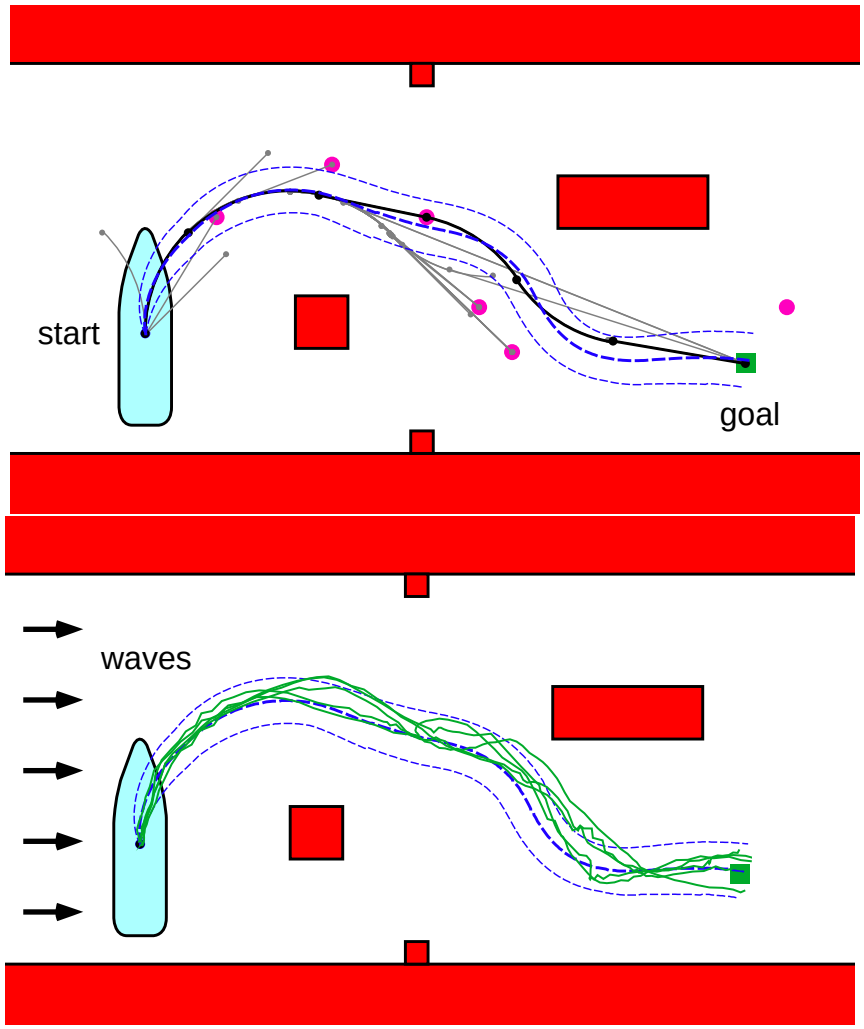

Figure 3-35: The vehicle drives from the initial configuration, as pictured, to the green square. The search tree is shown in gray (top) and the optimal plan is shown in black. The waypoints considered by the planner are shown as large pink circles. The predicted mean and standard deviation for the path-following error are shown as dashed blue lines. Five experimental trajectories are shown in green in the lower figure.

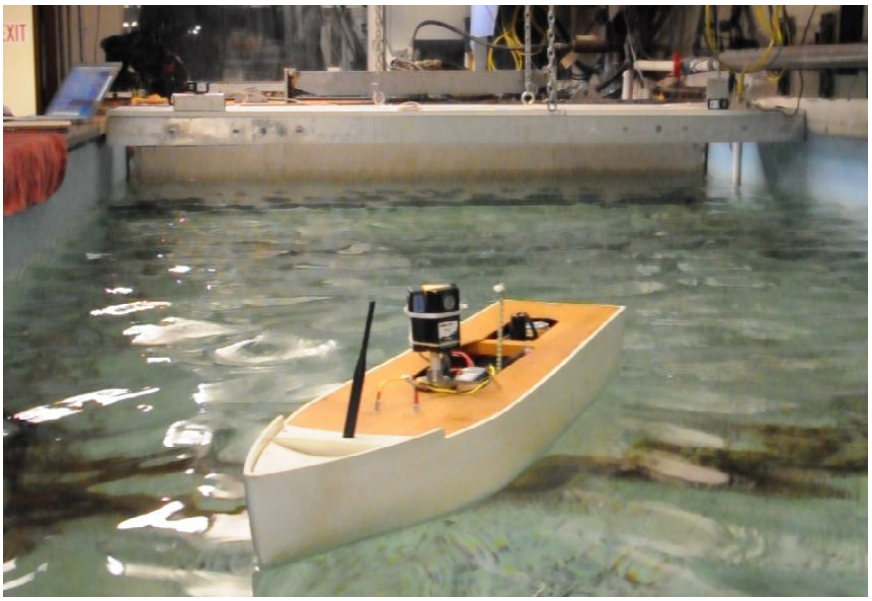

Figure 3-36: The vehicle near the end of the motion plan. The wavemaker can be seen behind the vehicle.

Chapter 4

Model Learning

In Chapter 3 we discussed how to generate motion plans that are robust to stochastic environmental disturbances. However, external disturbances are not the only potential source of path-following error. Because the state feedback controller is designed around an assumed model of the vehicle's dynamics and kinematics, the performance of the controller is related to the accuracy of the model. The system model may be derived mathematically, measured experimentally, learned online, or obtained through some combination of those methods. A mathematical model may be inaccurate if the assumptions upon which it is based are incorrect. An experimentally-derived model is subject to measurement error, and it may be inconvenient or expensive to perform the many system identification tests that are necessary to develop the model. Another problem is that the true vehicle model may change over time, either abruptly, due to a configuration change or damage, or gradually, due to use and wear. After the true model changes, the mathematical assumptions need to be reevaluated or more system identification experiments need to be performed.

The alternative to trying to establish an accurate model ahead of time is to learn the model online. Such a learning algorithm can adapt to changing parameter values to become more accurate over time as more learning data is collected. In fact, it is possible to predict how the model parameter uncertainty will change throughout a motion plan due to the learning algorithm. That information is useful when considering model uncertainty in the planner's cost function, which will be discussed in

Chapter 5.

The dynamic model of the vehicle presented in Chapter 3 has a rigid structure defined by a number of parameters. In this chapter we simplify the dynamic model by using the linearization of the dynamics around $\boldsymbol{\nu} = \mathbf{0}$, as shown in (3.6), to reduce the number of system parameters. Within this model structure, the simplest learning algorithm is least squares regression. While basic least squares regression is straightforward, some care needs to be taken to apply it to the multiple-input, multiple-output dynamic model for the planar holonomic vehicle. Other complications arise when the regression is performed in several batches (one batch per maneuver in the motion plan) and when the values of some of the parameters in the model are fixed (that is, they are assumed to be correct and they are not affected by the learning data). We fix some parameter values to focus the learning on other parameters. The least squares parameter estimation algorithm is derived in Section 4.1 with all of these factors taken into account.

As the model improves due to the learning algorithm, the path-following error improves as well. This model improvement must be taken into account if the model uncertainty is to be incorporated into the planner's cost function, as described in the next chapter. While it is impossible to predict how the parameter values themselves will change throughout the learning process, it *is* possible to predict how the covariance matrix of the parameter estimates changes as a function of the input data to the learning algorithm. Section 4.2 discusses how to use this prediction in active learning strategies to quickly and accurately identify the system parameters.

Due to the stochastic nature of the external disturbances and, consequently, the vehicle trajectory, the input data to the learning algorithm, $\boldsymbol{\Phi}$, is also stochastic. The parameter variance prediction is more complicated when the learning data is stochastic, as it involves the expected value of $(\boldsymbol{\Phi}\boldsymbol{\Phi}^T)^{-1}$, where $\boldsymbol{\Phi}$ has correlations through time. In Section 4.3 we develop an approximation to this expected value for a simplified system, and the solution is applied to the original vehicle model in Section 4.4.

The parameter covariance prediction is based on several assumptions: the process

and sensor noise levels are known, the initial parameter variance (initial confidence) is known, and the true parameter values do not change. If any of these assumptions are violated, then the predicted parameter variance may differ from the actual parameter variance as measured across all possible noise sequences. Therefore a posterior correction is needed to properly set the initial parameter confidence for the next planning task. For example, if the true parameter value changes during the mission, then the parameter confidence should be reduced at the end of the mission so that the parameter estimates will converge quickly to the new true values during the next mission. This posterior parameter variance estimate is derived in Section 4.5.2.

Predicting the expected information gain from different types of maneuvers is the first step in creating a planner that optimally balances model learning with goal-seeking behavior without any human input. The next chapter will translate the expected parameter variance evolution into an expected collision cost, thereby completing the integrated planning and learning algorithm.

4.1 Least Squares Parameter Estimation

In this section we describe the least squares regression algorithm for learning the parameters of the dynamic model of the vehicle. The full model is partitioned into three separate models, one for each output $\dot{u}$, $\dot{v}$ and $\dot{r}$. Within each partitioned model, some of the parameters may be fixed while others are free to be adjusted by the regression algorithm. Furthermore, the regression takes place in several batches, as a compromise between the full least squares regression and the recursive least squares algorithm. Using the entire data set at once is the simplest approach, but it means that the parameters are not updated until after the mission is over. The recursive least squares algorithm results in the fastest parameter updates, but predictions of the learning rate are more difficult with that approach. Updating the parameters after each maneuver allows the parameters to become more accurate throughout the mission while still allowing for simple parameter convergence predictions. These extensions to least squares regression mean that the update equations become very

complicated and quite a lot of accounting is necessary to keep track of the various partitioning matrices and pieces of learning data. However, the final result is a flexible tool for learning some or all of the parameters in one or more stages, with or without initial guesses for the parameter values.

4.1.1 Parameter Vector System Model Representation

A general linear model for a single output y has the form:

$$\begin{aligned} y &= \beta_1\phi_1 + \beta_2\phi_2 + \cdots + \beta_m\phi_m + w \\ &= \boldsymbol{\beta}^T\boldsymbol{\phi} + w \end{aligned} \tag{4.1}$$

where m is the number of parameters in the system and w is uncorrelated random noise, which we assume is sampled from a zero-mean Gaussian distribution. We can see that our model of the vehicle dynamics (3.4) has this structure if we incorporate sensor noise w_s.

$$\begin{aligned} y_u = \dot{u} + w_{su} &= a_{11}u + b_1\tau_x + f_u + w_u + w_{su} \\ y_v = \dot{v} + w_{sv} &= a_{22}v + a_{23}r + b_2\tau_y + f_v + w_v + w_{sv} \\ y_r = \dot{r} + w_{sr} &= a_{32}v + a_{33}r + b_3\tau_y + w_r + w_{sr} \end{aligned} \tag{4.2}$$

The terms f_u and f_v are the components of the wind force in body-fixed coordinates. However, we seek to learn f_U and f_V, the wind components in global-frame coordinates. Note that the model in (4.2) is a continuous-time model whose output is an acceleration. In practice the acceleration will be measured using a finite-difference approach. An equivalent discrete-time model could be constructed with the same number of parameters, but we choose the continuous-time model structure for compatibility with the models used in Chapter 3.

We build a learning data vector $\boldsymbol{\phi}$ which includes the input data used in the

regression, $\boldsymbol{\nu}$ and $\boldsymbol{\tau}$.

$$\boldsymbol{\phi} = \begin{bmatrix} \boldsymbol{\nu} \\ \boldsymbol{\tau} \\ 1 \end{bmatrix} \tag{4.3}$$

The model parameters in (4.2) are stored in a parameter vector $\boldsymbol{\beta}$:

$$\boldsymbol{\beta} = \begin{bmatrix} a_{11} & b_1 & a_{22} & a_{23} & b_2 & a_{32} & a_{33} & b_3 & f_U & f_V \end{bmatrix}^T \tag{4.4}$$

with

$$\begin{bmatrix} a_{11} \\ b_1 \\ f_u \end{bmatrix} = \mathbf{C}_{\boldsymbol{\beta}u}\,\boldsymbol{\beta} \qquad \begin{bmatrix} a_{22} \\ a_{23} \\ b_2 \\ f_v \end{bmatrix} = \mathbf{C}_{\boldsymbol{\beta}v}\,\boldsymbol{\beta} \qquad \begin{bmatrix} a_{32} \\ a_{33} \\ b_3 \end{bmatrix} = \mathbf{C}_{\boldsymbol{\beta}r}\,\boldsymbol{\beta} \tag{4.5}$$

The partitioning matrices $\mathbf{C}_{\boldsymbol{\beta}u}$, $\mathbf{C}_{\boldsymbol{\beta}v}$ and $\mathbf{C}_{\boldsymbol{\beta}r}$ extract the parameters associated with the surge, sway and yaw dynamics, respectively. These matrices also rotate the global wind parameter f_U and f_V to their body-reference equivalents f_u and f_v according to the vehicle heading ψ.

$$\begin{aligned}
\mathbf{C}_{\boldsymbol{\beta}u} &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cos\psi & \sin\psi \end{bmatrix} \\
\mathbf{C}_{\boldsymbol{\beta}v} &= \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\sin\psi & \cos\psi \end{bmatrix} \\
\mathbf{C}_{\boldsymbol{\beta}r} &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}
\end{aligned} \tag{4.6}$$

Next we can define a set of selection matrices for the $\boldsymbol{\phi}$ vector:

$$\begin{aligned}
\begin{bmatrix} u \\ \tau_x \\ 1 \end{bmatrix} &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{\phi} \equiv \mathbf{C}_{\boldsymbol{\phi} u} \boldsymbol{\phi} \\
\begin{bmatrix} v \\ r \\ \tau_y \\ 1 \end{bmatrix} &= \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{\phi} \equiv \mathbf{C}_{\boldsymbol{\phi} v} \boldsymbol{\phi} \\
\begin{bmatrix} v \\ r \\ \tau_y \end{bmatrix} &= \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \boldsymbol{\phi} \equiv \mathbf{C}_{\boldsymbol{\phi} r} \boldsymbol{\phi}
\end{aligned} \tag{4.7}$$

Using the partitions of $\boldsymbol{\beta}$ in (4.5) and $\boldsymbol{\phi}$ in (4.7), we can write (4.2) in a more compact form:

$$\begin{aligned}
y_u &= \begin{bmatrix} a_{11} & b_1 & f_u \end{bmatrix} \begin{bmatrix} u \\ \tau_x \\ 1 \end{bmatrix} + w_u + w_{su} = \boldsymbol{\beta}^T \mathbf{C}_{\boldsymbol{\beta} u}{}^T \mathbf{C}_{\boldsymbol{\phi} u} \boldsymbol{\phi} + w_u + w_{su} \\
y_v &= \begin{bmatrix} a_{22} & a_{23} & b_2 & f_v \end{bmatrix} \begin{bmatrix} v \\ r \\ \tau_y \\ 1 \end{bmatrix} + w_v + w_{sv} = \boldsymbol{\beta}^T \mathbf{C}_{\boldsymbol{\beta} v}{}^T \mathbf{C}_{\boldsymbol{\phi} v} \boldsymbol{\phi} + w_v + w_{sv} \\
y_r &= \begin{bmatrix} a_{32} & a_{33} & b_3 \end{bmatrix} \begin{bmatrix} v \\ r \\ \tau_y \end{bmatrix} + w_r + w_{sr} = \boldsymbol{\beta}^T \mathbf{C}_{\boldsymbol{\beta} r}{}^T \mathbf{C}_{\boldsymbol{\phi} r} \boldsymbol{\phi} + w_r + w_{sr}
\end{aligned} \tag{4.8}$$

Note the parallel between (4.8) and the general model (4.1).

4.1.2 Basic Linear Regression

For the general regression task of finding the best $\hat{\boldsymbol{\beta}}$ that fits (4.1), the output data is stored in a $1 \times N$ vector $\mathbf{Y}$ and the input data is stored in a $m \times N$ matrix $\boldsymbol{\Phi}$ where N is the number of data samples and m is the length of $\boldsymbol{\phi}$.

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\phi}_1 & \dots & \boldsymbol{\phi}_N \end{bmatrix} \tag{4.9}$$

$$\mathbf{Y} = \begin{bmatrix} y_1 & \dots & y_N \end{bmatrix} \tag{4.10}$$

$$\mathbf{w} = \begin{bmatrix} w_1 & \dots & w_N \end{bmatrix} \tag{4.11}$$

The full output vector can be written in terms of the parameter vector and the input data as follows:

$$\mathbf{Y} = \boldsymbol{\beta}^T \boldsymbol{\Phi} + \mathbf{w} \tag{4.12}$$

If we have an estimate for the parameter vector $\hat{\boldsymbol{\beta}}$, then the predicted output data given that estimate is:

$$\hat{\mathbf{Y}} = \hat{\boldsymbol{\beta}}^T \boldsymbol{\Phi} \tag{4.13}$$

In least squares regression, the optimal parameter estimate $\hat{\boldsymbol{\beta}}$ is the one that minimizes the sum of the squared estimation error (2.3). The sum of the squared error between the predicted output and the actual output is $(\hat{\mathbf{Y}} - \mathbf{Y})(\hat{\mathbf{Y}} - \mathbf{Y})^T$. The expected value of this quantity is minimized when its derivative with respect to $\hat{\boldsymbol{\beta}}$ is zero.

$$\begin{aligned} \mathbf{0} &= \frac{\partial}{\partial \hat{\boldsymbol{\beta}}} E\left[(\hat{\mathbf{Y}} - \mathbf{Y})(\hat{\mathbf{Y}} - \mathbf{Y})^T\right] \\ &= \frac{\partial}{\partial \hat{\boldsymbol{\beta}}} \left(\hat{\boldsymbol{\beta}}^T \boldsymbol{\Phi}\boldsymbol{\Phi}^T \hat{\boldsymbol{\beta}} - \hat{\boldsymbol{\beta}}^T \boldsymbol{\Phi}\mathbf{Y}^T - \mathbf{Y}\boldsymbol{\Phi}^T \hat{\boldsymbol{\beta}} + E[\mathbf{Y}\mathbf{Y}^T]\right) \\ &= -2\mathbf{Y}\boldsymbol{\Phi}^T + 2\hat{\boldsymbol{\beta}}^T \boldsymbol{\Phi}\boldsymbol{\Phi}^T \\ \hat{\boldsymbol{\beta}} &= \left(\boldsymbol{\Phi}\boldsymbol{\Phi}^T\right)^{-1} \boldsymbol{\Phi}\mathbf{Y}^T \end{aligned} \tag{4.14}$$

Equation (4.14) is the least-squares approximation of $\boldsymbol{\beta}$ given the input data $\boldsymbol{\Phi}$ and

the output data $\mathbf{Y}$. For convenience we define an *information matrix*, $\mathbf{R} = \mathbf{\Phi}\mathbf{\Phi}^T$. The trace of this positive-definite matrix increases whenever the algorithm is presented with new data $\boldsymbol{\phi}$.

Using the partitioning matrices (4.6) and (4.7), we can write the following:

$$\begin{aligned} \hat{\boldsymbol{\beta}}_u &= \mathbf{C}_{\boldsymbol{\beta} u}\hat{\boldsymbol{\beta}} &&= \left(\mathbf{C}_{\boldsymbol{\phi} u}\mathbf{\Phi}\mathbf{\Phi}^T{\mathbf{C}_{\boldsymbol{\phi} u}}^T\right)^{-1}\mathbf{C}_{\boldsymbol{\phi} u}\mathbf{\Phi}{\mathbf{Y}_u}^T \\ \hat{\boldsymbol{\beta}}_v &= \mathbf{C}_{\boldsymbol{\beta} v}\hat{\boldsymbol{\beta}} &&= \left(\mathbf{C}_{\boldsymbol{\phi} v}\mathbf{\Phi}\mathbf{\Phi}^T{\mathbf{C}_{\boldsymbol{\phi} v}}^T\right)^{-1}\mathbf{C}_{\boldsymbol{\phi} v}\mathbf{\Phi}{\mathbf{Y}_v}^T \\ \hat{\boldsymbol{\beta}}_r &= \mathbf{C}_{\boldsymbol{\beta} r}\hat{\boldsymbol{\beta}} &&= \left(\mathbf{C}_{\boldsymbol{\phi} r}\mathbf{\Phi}\mathbf{\Phi}^T{\mathbf{C}_{\boldsymbol{\phi} r}}^T\right)^{-1}\mathbf{C}_{\boldsymbol{\phi} r}\mathbf{\Phi}{\mathbf{Y}_r}^T \end{aligned} \tag{4.15}$$

Introducing the information matrices $\mathbf{R}_u = \mathbf{C}_{\boldsymbol{\phi} u}\mathbf{\Phi}\mathbf{\Phi}^T{\mathbf{C}_{\boldsymbol{\phi} u}}^T$, $\mathbf{R}_v = \mathbf{C}_{\boldsymbol{\phi} v}\mathbf{\Phi}\mathbf{\Phi}^T{\mathbf{C}_{\boldsymbol{\phi} v}}^T$ and $\mathbf{R}_r = \mathbf{C}_{\boldsymbol{\phi} r}\mathbf{\Phi}\mathbf{\Phi}^T{\mathbf{C}_{\boldsymbol{\phi} r}}^T$, the parameter estimation problem is divided into three groups:

$$\begin{aligned} \hat{\boldsymbol{\beta}}_u &= {\mathbf{R}_u}^{-1}\mathbf{C}_{\boldsymbol{\phi} u}\mathbf{\Phi}{\mathbf{Y}_u}^T \\ \hat{\boldsymbol{\beta}}_v &= {\mathbf{R}_v}^{-1}\mathbf{C}_{\boldsymbol{\phi} v}\mathbf{\Phi}{\mathbf{Y}_v}^T \\ \hat{\boldsymbol{\beta}}_r &= {\mathbf{R}_r}^{-1}\mathbf{C}_{\boldsymbol{\phi} r}\mathbf{\Phi}{\mathbf{Y}_r}^T \end{aligned} \tag{4.16}$$

The estimate vectors $\hat{\boldsymbol{\beta}}_u$ and $\hat{\boldsymbol{\beta}}_v$ contain the body-centered wind estimates f_u and f_v, respectively. Finally, the full parameter estimate (containing estimates for f_U and f_V) is assembled as follows:

$$\hat{\boldsymbol{\beta}} = {\mathbf{C}_{\boldsymbol{\beta} u}}^T\hat{\boldsymbol{\beta}}_u + {\mathbf{C}_{\boldsymbol{\beta} v}}^T\hat{\boldsymbol{\beta}}_v + {\mathbf{C}_{\boldsymbol{\beta} r}}^T\hat{\boldsymbol{\beta}}_r \tag{4.17}$$

4.1.3 Learning a Subset of Parameters

Let us temporarily return to the general regression problem (4.1) and (4.14). If some of the parameters in $\boldsymbol{\beta}$ are already known with an acceptable degree of confidence and we wish to learn the other parameters, then we must partition (4.1) into the known part and the unknown part. We define two selection matrices $\mathbf{C}_{fixed}$ and $\mathbf{C}_{free}$ to extract the fixed parameters and the free parameters (those which we wish to learn)

from $\boldsymbol{\beta}$.

$$\begin{aligned} \boldsymbol{\beta}_{fixed} &= \mathbf{C}_{fixed}\,\bar{\boldsymbol{\beta}} && (4.18) \\ \boldsymbol{\beta}_{free} &= \mathbf{C}_{free}\,\boldsymbol{\beta} && (4.19) \end{aligned}$$

The number of rows in $\mathbf{C}_{fixed}$ is m_{fixed}, the number of fixed parameters. The number of rows in $\mathbf{C}_{free}$ is m_{free}, which is the number of free parameters. Naturally $m_{fixed} + m_{free} = m$. Each row of $\mathbf{C}_{fixed}$ and $\mathbf{C}_{free}$ contains all zeros except for a single value of one corresponding to an element of $\boldsymbol{\beta}$. The fixed parameter vector $\bar{\boldsymbol{\beta}}$ has m values, but only the m_{fixed} elements extracted by $\mathbf{C}_{fixed}$ are used; the other elements can be set to any value, such as zero.

Using these selection matrices (4.18-4.19), the original model (4.1) becomes:

$$\begin{aligned} y &= \bar{\boldsymbol{\beta}}^T {\mathbf{C}_{fixed}}^T \mathbf{C}_{fixed}\boldsymbol{\phi} + \boldsymbol{\beta}^T {\mathbf{C}_{free}}^T \mathbf{C}_{free}\boldsymbol{\phi} + w \\ &\equiv \bar{\boldsymbol{\beta}}^T \mathbf{I}_{fixed}\boldsymbol{\phi} + \boldsymbol{\beta}^T \mathbf{I}_{free}\boldsymbol{\phi} + w \\ y - \bar{\boldsymbol{\beta}}^T \mathbf{I}_{fixed}\boldsymbol{\phi} &= \boldsymbol{\beta}^T \mathbf{I}_{free}\boldsymbol{\phi} + w && (4.20) \end{aligned}$$

In the equations above, $\mathbf{I}_{fixed}$ and $\mathbf{I}_{free}$ are $m \times m$ matrices with ones on the diagonal corresponding to the fixed and free parameters, respectively. Note that $\mathbf{I}_{fixed} + \mathbf{I}_{free} = \mathbf{I}_{m\times m}$. We can define $\mathbf{Y}'$ as a row vector containing the left side of (4.20) evaluated at each time point: $\mathbf{Y}' = \mathbf{Y} - \bar{\boldsymbol{\beta}}^T \mathbf{I}_{fixed}\boldsymbol{\Phi}$. Now the full problem (4.12) is written as follows:

$$\mathbf{Y}' = \boldsymbol{\beta}^T \mathbf{I}_{free}\boldsymbol{\Phi} + \mathbf{w} \qquad (4.21)$$

Repeating the derivation of the least squares parameter estimate (4.14) and applying it to the partitioned problem, we have:

$$\begin{aligned} \hat{\boldsymbol{\beta}}_{free,u} &= {\mathbf{R}_u}^{-1} \mathbf{C}_{free,u} \mathbf{C}_{\boldsymbol{\phi} u} \boldsymbol{\Phi} ({\mathbf{Y}_u}')^T \\ \hat{\boldsymbol{\beta}}_{free,v} &= {\mathbf{R}_v}^{-1} \mathbf{C}_{free,v} \mathbf{C}_{\boldsymbol{\phi} v} \boldsymbol{\Phi} ({\mathbf{Y}_v}')^T \qquad (4.22) \\ \hat{\boldsymbol{\beta}}_{free,r} &= {\mathbf{R}_r}^{-1} \mathbf{C}_{free,r} \mathbf{C}_{\boldsymbol{\phi} r} \boldsymbol{\Phi} ({\mathbf{Y}_r}')^T \end{aligned}$$

where

$$\begin{aligned}
\mathbf{R}_u &= \mathbf{C}_{free,u}\mathbf{C}_{\boldsymbol{\phi}u}\boldsymbol{\Phi}\boldsymbol{\Phi}^T\mathbf{C}_{\boldsymbol{\phi}u}{}^T\mathbf{C}_{free,u}{}^T & & \mathbf{Y}_u{}' &= \mathbf{Y}_u - \bar{\boldsymbol{\beta}}^T\mathbf{C}_{\boldsymbol{\beta}u}{}^T\mathbf{I}_{fixed,u}\mathbf{C}_{\boldsymbol{\phi}u}\boldsymbol{\Phi} \\
\mathbf{R}_v &= \mathbf{C}_{free,v}\mathbf{C}_{\boldsymbol{\phi}v}\boldsymbol{\Phi}\boldsymbol{\Phi}^T\mathbf{C}_{\boldsymbol{\phi}v}{}^T\mathbf{C}_{free,v}{}^T & \text{and} & \mathbf{Y}_v{}' &= \mathbf{Y}_v - \bar{\boldsymbol{\beta}}^T\mathbf{C}_{\boldsymbol{\beta}v}{}^T\mathbf{I}_{fixed,v}\mathbf{C}_{\boldsymbol{\phi}v}\boldsymbol{\Phi} \\
\mathbf{R}_r &= \mathbf{C}_{free,r}\mathbf{C}_{\boldsymbol{\phi}r}\boldsymbol{\Phi}\boldsymbol{\Phi}^T\mathbf{C}_{\boldsymbol{\phi}r}{}^T\mathbf{C}_{free,r}{}^T & & \mathbf{Y}_r{}' &= \mathbf{Y}_r - \bar{\boldsymbol{\beta}}^T\mathbf{C}_{\boldsymbol{\beta}r}{}^T\mathbf{I}_{fixed,r}\mathbf{C}_{\boldsymbol{\phi}r}\boldsymbol{\Phi}
\end{aligned} \tag{4.23}$$

Note that some of these matrices may be of zero size, depending on the size of $\mathbf{C}_{free}$ for each channel. The full parameter estimate vector, including the fixed terms, is then assembled as shown below. In this case $\mathbf{I}_{fixed}$ is a square matrix with ones on the diagonal corresponding to the fixed terms (those that are not being learned).

$$\hat{\boldsymbol{\beta}} = \mathbf{C}_{\boldsymbol{\beta}u}{}^T\mathbf{C}_{free,u}{}^T\hat{\boldsymbol{\beta}}_{free,u} + \mathbf{C}_{\boldsymbol{\beta}v}{}^T\mathbf{C}_{free,v}{}^T\hat{\boldsymbol{\beta}}_{free,v} + \mathbf{C}_{\boldsymbol{\beta}r}{}^T\mathbf{C}_{free,r}{}^T\hat{\boldsymbol{\beta}}_{free,r} + \mathbf{I}_{fixed}\bar{\boldsymbol{\beta}} \tag{4.24}$$

4.1.4 Batch Learning and Regularization

We now consider the case in which learning takes place in multiple stages. There are two reasons that this analysis is important. First, it is possible to get an intermediate parameter estimate before all of the data is collected. The extreme version of this is the recursive least squares algorithm in which the parameter estimate is updated after each data sample arrives. In our application we update the parameter estimate at the end of each maneuver in the motion plan, which likely contains multiple data samples. Second, the same algorithm for batch learning can be used to incorporate *regularization* into the learning process. Regularization refers to an initial confidence in some *a priori* parameter estimate, which may be obtained from a mathematical model, previous experiments, or any other source. If the confidence in the *a priori* estimate is large then the new data is given less weight than if there is low confidence in the *a priori* estimate or if no *a priori* data is available at all.

First, consider the case in which learning takes place in just two stages with the general model (4.1). In the first stage, N_0 data points are accumulated in $\boldsymbol{\Phi}_0$ and $\mathbf{Y}_0$. The parameter estimate resulting from the first stage is $\boldsymbol{\beta}_0$ using (4.14). In the second stage, N_1 data points are accumulated in $\boldsymbol{\Phi}_1$ and $\mathbf{Y}_1$. The entire data

set is therefore $\mathbf{\Phi} = [\mathbf{\Phi}_0 \ \mathbf{\Phi}_1]$ and $\mathbf{Y} = [\mathbf{Y}_0 \ \mathbf{Y}_1]$. It can be easily shown that the information matrix for the entire data set is $\mathbf{R} = \mathbf{R}_0 + \mathbf{R}_1$. It is useful to be able to evaluate the parameter estimate due to the entire data set without having to store the input/output data from the first stage. To do so, we use the Woodbury matrix identity:

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1} \tag{4.25}$$

We apply (4.25) twice to the general least squares parameter estimate equation (4.14). In the first application (the third line below), $A = \mathbf{R}_0$, $U = \mathbf{\Phi}_1$, $V = \mathbf{\Phi}_1{}^T$, and $C = \mathbf{I}_{N_1 \times N_1}$.

$$\begin{aligned}
\hat{\boldsymbol{\beta}} &= \mathbf{R}^{-1}\mathbf{\Phi}\mathbf{Y}^T \\
&= (\mathbf{R}_0 + \mathbf{\Phi}_1\mathbf{\Phi}_1{}^T)^{-1}\left(\mathbf{\Phi}_0\mathbf{Y}_0{}^T + \mathbf{\Phi}_1\mathbf{Y}_1{}^T\right) \\
&= \left(\mathbf{R}_0{}^{-1} - \mathbf{R}_0{}^{-1}\mathbf{\Phi}_1(\mathbf{I} + \mathbf{\Phi}_1{}^T\mathbf{R}_0{}^{-1}\mathbf{\Phi}_1)^{-1}\mathbf{\Phi}_1{}^T\mathbf{R}_0{}^{-1}\right)\left(\mathbf{\Phi}_0\mathbf{Y}_0{}^T + \mathbf{\Phi}_1\mathbf{Y}_1{}^T\right) \\
&= \hat{\boldsymbol{\beta}}_0 + \mathbf{R}_0{}^{-1}\mathbf{\Phi}_1\mathbf{Y}_1{}^T - \mathbf{R}_0{}^{-1}\mathbf{\Phi}_1(\mathbf{I} + \mathbf{\Phi}_1{}^T\mathbf{R}_0{}^{-1}\mathbf{\Phi}_1)^{-1}\left(\mathbf{\Phi}_1{}^T\hat{\boldsymbol{\beta}}_0 + \mathbf{\Phi}_1{}^T\mathbf{R}_0{}^{-1}\mathbf{\Phi}_1\mathbf{Y}_1{}^T\right)
\end{aligned} \tag{4.26}$$

Here we note that $\mathbf{\Phi}_1{}^T\hat{\boldsymbol{\beta}}_0 = \hat{\mathbf{Y}}_1^T$, and we can continue to simplify (4.26).

$$\begin{aligned}
\hat{\boldsymbol{\beta}} &= \hat{\boldsymbol{\beta}}_0 + \mathbf{R}_0{}^{-1}\mathbf{\Phi}_1\mathbf{Y}_1{}^T - \mathbf{R}_0{}^{-1}\mathbf{\Phi}_1(\mathbf{I} + \mathbf{\Phi}_1{}^T\mathbf{R}_0{}^{-1}\mathbf{\Phi}_1)^{-1}\left(\hat{\mathbf{Y}}_1^T + \mathbf{\Phi}_1{}^T\mathbf{R}_0{}^{-1}\mathbf{\Phi}_1\mathbf{Y}_1{}^T\right) \\
&= \hat{\boldsymbol{\beta}}_0 + \mathbf{R}_0{}^{-1}\mathbf{\Phi}_1\mathbf{Y}_1{}^T \\
&\qquad - \mathbf{R}_0{}^{-1}\mathbf{\Phi}_1(\mathbf{I} + \mathbf{\Phi}_1{}^T\mathbf{R}_0{}^{-1}\mathbf{\Phi}_1)^{-1}\left((\mathbf{I} + \mathbf{\Phi}_1{}^T\mathbf{R}_0{}^{-1}\mathbf{\Phi}_1)\mathbf{Y}_1{}^T + (\hat{\mathbf{Y}}_1 - \mathbf{Y}_1)^T\right) \\
&= \hat{\boldsymbol{\beta}}_0 + \cancel{\mathbf{R}_0{}^{-1}\mathbf{\Phi}_1\mathbf{Y}_1{}^T} - \cancel{\mathbf{R}_0{}^{-1}\mathbf{\Phi}_1\mathbf{Y}_1{}^T} - \mathbf{R}_0{}^{-1}\mathbf{\Phi}_1(\mathbf{I} + \mathbf{\Phi}_1{}^T\mathbf{R}_0{}^{-1}\mathbf{\Phi}_1)^{-1}(\hat{\mathbf{Y}}_1 - \mathbf{Y}_1)^T \\
&= \hat{\boldsymbol{\beta}}_0 + \mathbf{R}_0{}^{-1}\mathbf{\Phi}_1(\mathbf{I} + \mathbf{\Phi}_1{}^T\mathbf{R}_0{}^{-1}\mathbf{\Phi}_1)^{-1}(\mathbf{Y}_1 - \hat{\mathbf{Y}}_1)^T
\end{aligned} \tag{4.27}$$

Equation (4.27) is the recursive least squares algorithm when $N_1 = 1$. However, because (4.27) requires the inversion of an $N_1 \times N_1$ matrix, this form is not desirable when N_1 is large. Therefore we apply the Woodbury matrix identity a second time

(the second line below), with $A = \mathbf{I}$, $U = {\mathbf{\Phi}_1}^T$, $V = \mathbf{\Phi}_1$, and $C = {\mathbf{R}_0}^{-1}$.

$$\begin{aligned}
\hat{\boldsymbol{\beta}} &= \hat{\boldsymbol{\beta}}_0 + {\mathbf{R}_0}^{-1}\mathbf{\Phi}_1(\mathbf{I} + {\mathbf{\Phi}_1}^T{\mathbf{R}_0}^{-1}\mathbf{\Phi}_1)^{-1}(\mathbf{Y}_1 - \hat{\mathbf{Y}}_1)^T \\
&= \hat{\boldsymbol{\beta}}_0 + {\mathbf{R}_0}^{-1}\mathbf{\Phi}_1\left(\mathbf{I} - {\mathbf{\Phi}_1}^T(\mathbf{R}_0 + \mathbf{\Phi}_1{\mathbf{\Phi}_1}^T)^{-1}\mathbf{\Phi}_1\right)(\mathbf{Y}_1 - \hat{\mathbf{Y}}_1)^T \\
&= \hat{\boldsymbol{\beta}}_0 + {\mathbf{R}_0}^{-1}\left(\mathbf{I} - \mathbf{R}_1(\mathbf{R}_0 + \mathbf{R}_1)^{-1}\right)\mathbf{\Phi}_1(\mathbf{Y}_1 - \hat{\mathbf{Y}}_1)^T
\end{aligned} \tag{4.28}$$

Equation (4.28) calculates the same parameter estimate as if the entire dataset was used, but it only requires the $m \times m$ information matrix $\mathbf{R}_0$ and the parameter estimate $\hat{\boldsymbol{\beta}}_0$ from the first stage rather than the entire input-output data $\mathbf{\Phi}_0$ and $\mathbf{Y}_0$. Furthermore, (4.28) only requires the inversion of an $m \times m$ matrix. Equation (4.28) can be applied recursively each time a new batch of data arrives. When applied to the first batch of data, $\hat{\boldsymbol{\beta}}_0$ represents the *a priori* parameter estimate and $\mathbf{R}_0$ is proportional to the initial confidence of the *a priori* estimate, as discussed in Section 4.4.

For the partitioned system for the planar holonomic vehicle model, the initial parameter estimates are partitioned as follows:

$$\begin{aligned}
\hat{\boldsymbol{\beta}}_{free,u,0} &= \mathbf{C}_{free,u}\mathbf{C}_{\boldsymbol{\beta} u}\hat{\boldsymbol{\beta}}_0 \\
\hat{\boldsymbol{\beta}}_{free,v,0} &= \mathbf{C}_{free,v}\mathbf{C}_{\boldsymbol{\beta} v}\hat{\boldsymbol{\beta}}_0 \\
\hat{\boldsymbol{\beta}}_{free,r,0} &= \mathbf{C}_{free,r}\mathbf{C}_{\boldsymbol{\beta} r}\hat{\boldsymbol{\beta}}_0
\end{aligned} \tag{4.29}$$

The update equations are:

$$\begin{aligned}
\hat{\boldsymbol{\beta}}_{free,u} &= \hat{\boldsymbol{\beta}}_{free,u,0} + {\mathbf{R}_{u,0}}^{-1}\left(\mathbf{I} - \mathbf{R}_{u,1}(\mathbf{R}_{u,0} + \mathbf{R}_{u,1})^{-1}\right)\mathbf{C}_{free,u}\mathbf{C}_{\boldsymbol{\phi} u}\mathbf{\Phi}_1\left(\mathbf{Y}_{u,1}{}' - \hat{\mathbf{Y}}_{u,1}\right)^T \\
\hat{\boldsymbol{\beta}}_{free,v} &= \hat{\boldsymbol{\beta}}_{free,v,0} + {\mathbf{R}_{v,0}}^{-1}\left(\mathbf{I} - \mathbf{R}_{v,1}(\mathbf{R}_{v,0} + \mathbf{R}_{v,1})^{-1}\right)\mathbf{C}_{free,v}\mathbf{C}_{\boldsymbol{\phi} v}\mathbf{\Phi}_1\left(\mathbf{Y}_{v,1}{}' - \hat{\mathbf{Y}}_{v,1}\right)^T \\
\hat{\boldsymbol{\beta}}_{free,r} &= \hat{\boldsymbol{\beta}}_{free,r,0} + {\mathbf{R}_{r,0}}^{-1}\left(\mathbf{I} - \mathbf{R}_{r,1}(\mathbf{R}_{r,0} + \mathbf{R}_{r,1})^{-1}\right)\mathbf{C}_{free,r}\mathbf{C}_{\boldsymbol{\phi} r}\mathbf{\Phi}_1\left(\mathbf{Y}_{r,1}{}' - \hat{\mathbf{Y}}_{r,1}\right)^T
\end{aligned} \tag{4.30}$$

with each $\mathbf{R}$ and $\mathbf{Y}'$ defined in (4.23) and

$$\begin{aligned}
\hat{\mathbf{Y}}_{u,1} &= \hat{\boldsymbol{\beta}}_{free,u,0}{}^{T}\mathbf{C}_{free,u}\mathbf{C}_{\boldsymbol{\phi} u}\boldsymbol{\Phi}_1 \\
\hat{\mathbf{Y}}_{v,1} &= \hat{\boldsymbol{\beta}}_{free,v,0}{}^{T}\mathbf{C}_{free,v}\mathbf{C}_{\boldsymbol{\phi} v}\boldsymbol{\Phi}_1 \\
\hat{\mathbf{Y}}_{r,1} &= \hat{\boldsymbol{\beta}}_{free,r,0}{}^{T}\mathbf{C}_{free,r}\mathbf{C}_{\boldsymbol{\phi} r}\boldsymbol{\Phi}_1
\end{aligned} \tag{4.31}$$

The full parameter estimate vector is assembled as shown in (4.24). Note that because $\mathbf{C}_{\boldsymbol{\beta} u}$ and $\mathbf{C}_{\boldsymbol{\beta} v}$ are functions of the heading angle ψ, the learning batches must be small enough that ψ does not change appreciably throughout the batch. This is only an issue when the wind parameters are free (that is, they are being learned).

The update equations shown in (4.30) are used in the implementation of the least squares algorithm while the vehicle is executing a motion plan. For the planner to take model uncertainty into account while choosing the optimal motion plan, it needs to be able to predict how the parameters are likely to improve for each candidate motion plan considered by the planner. The next section describes how to predict the parameter convergence for each motion plan.

4.2 Parameter Convergence

In this section we explore the mean and covariance of the parameter estimates that are generated using the least squares learning algorithm (4.30). These statistics are taken across all possible sequences of noise $\mathbf{w}$ (4.11).

4.2.1 Mean and Covariance of the Parameter Estimate Vector

The expected value of the parameter vector is equal to the true value if the noise is uncorrelated with the input data, which we assume is true. This can be proven from the general regression solution by substituting (4.12) into (4.14) and taking the

expectation across all $\mathbf{w}$.

$$\begin{aligned} E[\hat{\boldsymbol{\beta}}] &= E\left[\mathbf{R}^{-1}\boldsymbol{\Phi}(\boldsymbol{\Phi}^T\boldsymbol{\beta} + \mathbf{w}^T)\right] \\ &= \mathbf{R}^{-1}\mathbf{R}\boldsymbol{\beta} + \mathbf{R}^{-1}E[\boldsymbol{\Phi}\mathbf{w}^T] \\ &= \boldsymbol{\beta} \\ \text{when } E[\boldsymbol{\Phi}\mathbf{w}^T] &= \mathbf{0} \end{aligned} \tag{4.32}$$

The covariance matrix $\mathbf{P}$ of the parameter estimate is derived below. Note that when $\mathbf{w}$ is a stationary process, then $E[\mathbf{w}^T\mathbf{w}] = W\,\mathbf{I}_{N\times N}$ where W is the noise variance.

$$\begin{aligned} \mathbf{P} &= E\left[(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta})(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta})^T\right] \\ &= E\left[\mathbf{R}^{-1}\boldsymbol{\Phi}\mathbf{w}^T\mathbf{w}\boldsymbol{\Phi}^T\mathbf{R}^{-1}\right] \\ &= W E\left[\mathbf{R}^{-1}\mathbf{R}\mathbf{R}^{-1}\right] \\ &= W E\left[\mathbf{R}^{-1}\right] \end{aligned} \tag{4.33}$$

For the partitioned system for the planar holonomic vehicle, the partitioned parameter covariance for the free parameters is:

$$\begin{aligned} \mathbf{P}_u &= W_u\, E\left[{\mathbf{R}_u}^{-1}\right] \\ \mathbf{P}_v &= W_v\, E\left[{\mathbf{R}_v}^{-1}\right] \\ \mathbf{P}_r &= W_r\, E\left[{\mathbf{R}_r}^{-1}\right] \end{aligned} \tag{4.34}$$

with the expected value of each inverse information matrix computed in Section 4.3. The noise variance is the sum of the variance of the process noise and the variance of the sensor noise: $W_u = \mathrm{var}(w_u) + \mathrm{var}(w_{su})$, with a similar result for the v and r channels.

4.2.2 Active Learning

If it is possible to predict the parameter convergence rate, then we can design motion plans for the purpose of reducing parameter uncertainty. This is the general goal of active learning: in particular, for most active learning problems the goal is to learn the parameters (equivalently, reduce the parameter uncertainty) as quickly as possible. The convergence of the parameter uncertainty, which corresponds to the reduction of $\mathbf{P}$, is a function of the learning data $\mathbf{\Phi}$. For different data sets $\mathbf{\Phi}$, the magnitude of $\mathbf{R} = \mathbf{\Phi}\mathbf{\Phi}^T$ will be different, and consequently $\mathbf{P}$ will be different. For the fastest parameter convergence, we can try to design $\mathbf{\Phi}$ so that $\mathbf{P}$ will be reduced as much as possible. Note that the expected parameter convergence does not depend on the output data $\mathbf{Y}$. It can be easily shown that the mean squared error of the parameter vector is equal to the trace of $\mathbf{P}$:

$$E\left[\|\hat{\boldsymbol{\beta}}\|^2\right] = \mathrm{Tr}(\mathbf{P}) \propto \mathrm{Tr}(\mathbf{R}^{-1}) \tag{4.35}$$

Given a data set $\mathbf{\Phi}$ and the corresponding $\mathbf{R}$, a new sample $\boldsymbol{\phi}$ augments $\mathbf{R}$ by $\boldsymbol{\phi}\boldsymbol{\phi}^T$. If we have the freedom to choose $\boldsymbol{\phi}$, then we would like to choose the $\boldsymbol{\phi}$ that minimizes $\mathrm{Tr}((\mathbf{R} + \boldsymbol{\phi}\boldsymbol{\phi}^T)^{-1})$. Using the Woodbury matrix identity (4.25), this is equivalent to:

$$\begin{aligned} \mathrm{Tr}\left((\mathbf{R} + \boldsymbol{\phi}\boldsymbol{\phi}^T)^{-1}\right) &= \mathrm{Tr}\left(\mathbf{R}^{-1} - \frac{1}{1 + \boldsymbol{\phi}^T\mathbf{R}^{-1}\boldsymbol{\phi}}\mathbf{R}^{-1}\boldsymbol{\phi}\boldsymbol{\phi}^T\mathbf{R}^{-1}\right) \\ &= \mathrm{Tr}\left(\mathbf{R}^{-1}\right) - \frac{\boldsymbol{\phi}^T\mathbf{R}^{-1}\mathbf{R}^{-1}\boldsymbol{\phi}}{1 + \boldsymbol{\phi}^T\mathbf{R}^{-1}\boldsymbol{\phi}} \end{aligned} \tag{4.36}$$

The sample that minimizes the mean squared parameter error the most is the one that maximizes the second term in (4.36). The maximizing sample has an infinite length and is proportional to the eigenvector of $\mathbf{R}^{-1}$ corresponding to the largest eigenvalue. The largest eigenvalue of $\mathbf{R}^{-1}$ corresponds to the smallest eigenvalue of $\mathbf{R}$, so this policy means that one must choose the mode of the input space that has been excited the least, and excite it as much as possible within the constraints of the system.

In practice, when the data $\boldsymbol{\phi}(t)$ is generated by a dynamic system, it is generally not possible to manipulate $\boldsymbol{\phi}$ directly at every sample time. Instead, changes in $\boldsymbol{\phi}$ are brought about by manipulating the control input $\boldsymbol{\tau}$. In this situation it is more helpful to be able to test the effects of different $\boldsymbol{\phi}$ samples arising from different control sequences $\boldsymbol{\tau}$ and select the control sequence that reduces the mean squared parameter error the most, rather than solving for the particular $\boldsymbol{\phi}$ that minimizes (4.36).

In our application, this task is complicated by three factors. First, we are not necessarily interested in minimizing the mean squared parameter error $\mathrm{Tr}(\mathbf{P})$; depending on the constraints in the environment and the planner's cost function, we may only be interested in minimizing certain modes of $\mathbf{P}$. Second, we are specifying a policy for $\boldsymbol{\tau}$ in the form of a reference trajectory and a set of controller gains, rather than an explicit control sequence. Third, the vector $\boldsymbol{\phi}$ is not a deterministic function of $\boldsymbol{\tau}$ because it is affected by process noise. Based on these three factors, we need to be able to predict the expected value of the full covariance matrix, $E[\mathbf{P}] \propto E[\mathbf{R}^{-1}]$, given a particular reference trajectory and a particular set of controller gains. The expected parameter covariance matrix is incorporated into the planner's cost function through its effect on the collision probability; this is described in Chapter 5.

4.2.3 Parameter Convergence Predictions

The expected value of the parameter covariance matrix $\mathbf{P}$ can be predicted if we can predict the expected value of the inverse of the information matrix, $E[\mathbf{R}^{-1}]$. Before evaluating this quantity for the planar holonomic vehicle described in (4.2), we consider correlated learning data that has been generated by a simple discrete-time state space system. The following section describes the simple state space system and the resulting correlated prediction of $E[\mathbf{R}^{-1}]$. Subsequently we will use that result to find the correlated prediction of that matrix for the holonomic vehicle system.

4.3 Predicting $E[\mathbf{R}^{-1}]$ for a Simple Correlated System

Computing the predicted parameter covariance from (4.33) is straightforward when the input data to the learning algorithm $\mathbf{\Phi}$ is known deterministically. In practice, however, $\mathbf{\Phi}$ represents a stochastic dataset, because it contains $\boldsymbol{\nu}(t)$ and $\boldsymbol{\tau}(t)$, which are themselves the output of a system driven by stochastic noise $\mathbf{w}_{\boldsymbol{\nu}}$. Consequently, the data is both stochastic and correlated through time; both of these factors make the partitioned covariance matrices in (4.34) very difficult to estimate analytically for the planar holonomic vehicle. To solve this problem we create a similar simple discrete-time system. The simple system is multidimensional and correlated through time, just like the planar holonomic vehicle. With this system we can compute a correlated prediction of $E[\mathbf{R}^{-1}]$. A summary of the solution is provided in Section 4.3.6. In Section 4.4 we will relate this solution back to the planar holonomic vehicle.

4.3.1 Discrete-Time System

Consider the simple m-dimensional discrete-time system shown below.

$$\boldsymbol{\phi}_{i+1} = \mathbf{a}\boldsymbol{\phi}_i + \mathbf{b} + \mathbf{w}_i \tag{4.37}$$

The state $\boldsymbol{\phi}_i$, the constant exogenous input $\mathbf{b}$, and the noise vector $\mathbf{w}_i$ are all $m \times 1$ vectors. The noise is sampled from a zero-mean multinormal distribution whose covariance matrix is $\mathbf{W} = E[\mathbf{w}_i {\mathbf{w}_i}^T]$. The noise samples are uncorrelated in time. The $m \times m$ matrix $\mathbf{a}$ is a constant linear filter on $\boldsymbol{\phi}_i$. The filter is not required to be stable, so the eigenvalues may be outside of the unit circle.

4.3.2 Mean and Covariance

The evolution of the mean state $\bar{\boldsymbol{\phi}}(t_i)$ can be computed analytically under certain conditions, or it can be found by recursively evaluating the sum in (4.37) with $\mathbf{w}_i = 0$.

The covariance matrix $\mathbf{\Sigma}(t_i)$ can also be found with an analytic approach under certain conditions when $m = 1$, and through a recursive summation for any m.

When starting from the initial condition $\boldsymbol{\phi}_0 = 0$, the evolution equation (4.37) can be expressed as an arithmetic sum:

$$\boldsymbol{\phi}_i = \sum_{k=0}^{i-1} \mathbf{a}^k \mathbf{b} + \sum_{k=0}^{i-1} \mathbf{a}^k \mathbf{w}_{i-k-1} \tag{4.38}$$

The sums in (4.38) have the following properties. The particular case for $m = 1$ is listed as well.

$$\begin{aligned}
\sum_{k=0}^{N-1} \mathbf{a}^k &= (\mathbf{I} - \mathbf{a})^{-1}(\mathbf{I} - \mathbf{a}^N) = \frac{1 - a^N}{1 - a} \quad \text{when } m = 1 && (4.39)\\
\sum_{k=1}^{N} \mathbf{a}^k &= (\mathbf{I} - \mathbf{a})^{-1}(\mathbf{I} - \mathbf{a}^N) - (\mathbf{I} - \mathbf{a}^N) = \frac{1 - a^N}{1 - a} - (1 - a^N) \quad \text{when } m = 1 \\
\sum_{k=0}^{\infty} \mathbf{a}^k &= (\mathbf{I} - \mathbf{a})^{-1} = \frac{1}{1 - a} \quad \text{when } m = 1
\end{aligned}$$

Recalling that $E[\mathbf{w}_i] = \mathbf{0}$, we can easily evaluate the expected value of $\boldsymbol{\phi}_i$:

$$\begin{aligned}
\bar{\boldsymbol{\phi}}_i = E[\boldsymbol{\phi}_i] &= \sum_{k=0}^{i-1} \mathbf{a}^k \mathbf{b} && (4.40)\\
&= (\mathbf{I} - \mathbf{a})^{-1}(\mathbf{I} - \mathbf{a}^i)\mathbf{b} && (4.41)
\end{aligned}$$

The covariance matrix for $\boldsymbol{\phi}_i$ is:

$$\begin{aligned}
\mathbf{\Sigma}_i = E[(\boldsymbol{\phi}_i - \bar{\boldsymbol{\phi}}_i)(\boldsymbol{\phi}_i - \bar{\boldsymbol{\phi}}_i)^T] &= E\left[\sum_{k=0}^{i-1} \mathbf{a}^k \mathbf{w}_{i-k-1} \sum_{k=0}^{i-1} \mathbf{w}_{i-k-1}^T (\mathbf{a}^k)^T\right] \\
&= \sum_{k=0}^{i-1} \mathbf{a}^k \mathbf{W} (\mathbf{a}^k)^T && (4.42)\\
&= \frac{W}{1 - a^2}\left(1 - a^{2i}\right) \quad \text{when } m = 1 && (4.43)
\end{aligned}$$

An important quantity used later is the covariance matrix between different sam-

ples $\boldsymbol{\phi}_i$ and $\boldsymbol{\phi}_j$. The covariance is defined below:

$$\begin{aligned}\mathrm{cov}(\boldsymbol{\phi}_i, \boldsymbol{\phi}_j) &= E[\boldsymbol{\phi}_i \boldsymbol{\phi}_j{}^T] - E[\boldsymbol{\phi}_i]E[\boldsymbol{\phi}_j]^T \\ &= E\left[\left(\bar{\boldsymbol{\phi}}_i + \sum_{k_1=0}^{i-1} \mathbf{a}^{k_1}\mathbf{w}_{i-k_1-1}\right)\left(\bar{\boldsymbol{\phi}}_j^T + \sum_{k_2=0}^{j-1} \mathbf{w}_{j-k_2-1}^T(\mathbf{a}^{k_2})^T\right)\right] - \bar{\boldsymbol{\phi}}_i\bar{\boldsymbol{\phi}}_j^T\end{aligned} \tag{4.44}$$

The first terms in the multiplication combine to $\bar{\boldsymbol{\phi}}_i\bar{\boldsymbol{\phi}}_j^T$, and the cross terms reduce to zero in the expectation because $E[\mathbf{w}_i] = 0$. For the remaining term we assume, without a loss of generality, that $i > j$ (which is allowed because the covariance is a symmetric function). The multiplied sums are only nonzero in the expectation when $k_1 = k_2 + i - j$. Because $i > j$, the resulting combined sum is only taken over j:

$$\begin{aligned}\mathrm{cov}(\boldsymbol{\phi}_i, \boldsymbol{\phi}_j) &= \sum_{k_2=0}^{j-1} \mathbf{a}^{k_2+i-j}\mathbf{W}(\mathbf{a}^{k_2})^T \\ &= \mathbf{a}^{i-j}\sum_{k_2=0}^{j-1} \mathbf{a}^{k_2}\mathbf{W}(\mathbf{a}^{k_2})^T \\ &= \mathbf{a}^{i-j}\boldsymbol{\Sigma}_j\end{aligned} \tag{4.45}$$

Relaxing the $i > j$ assumption, the covariance is:

$$\mathbf{V}_{ij} \equiv \mathrm{cov}(\boldsymbol{\phi}_i, \boldsymbol{\phi}_j) = \begin{cases} \mathbf{a}^{i-j}\boldsymbol{\Sigma}_j & \text{when } i \geq j \\ \boldsymbol{\Sigma}_i(\mathbf{a}^{j-i})^T & \text{when } i < j \end{cases} \tag{4.46}$$

$$= \frac{Wa^{|i-j|}}{1-a^2}\left(1 - a^{2\min(i,j)}\right) \quad \text{when } m = 1 \tag{4.47}$$

As we expect, $\mathbf{V}_{ii} = \boldsymbol{\Sigma}_i$. In the scalar case ($m = 1$), we can define the correlation coefficient between the samples ϕ_i and ϕ_j:

$$\mathrm{corr}(\phi_i, \phi_j) = \frac{\mathrm{cov}(\phi_i, \phi_j)}{\sqrt{\Sigma_i\Sigma_j}} \tag{4.48}$$

4.3.3 χ^2 Distribution and the Wishart Distribution

The information matrix used in the least squares learning algorithm is a summation of the square of the data vector over all of the time points.

$$\mathbf{R} = \mathbf{\Phi}\mathbf{\Phi}^T = \sum_{k=1}^{N} \boldsymbol{\phi}_i \boldsymbol{\phi}_i{}^T \tag{4.49}$$

When $m = 1$, $\bar{\phi}_i = 0$, Σ_i is constant, and there is no correlation between samples ($\text{cov}(\phi_i, \phi_j) = 0$ for $i \neq j$, which is equivalent to $a = 0$), then R represents a χ^2 distribution. The multi-parameter version, when $m > 1$, is known as the Wishart distribution [60]. When $\bar{\boldsymbol{\phi}}_i \neq \mathbf{0}$, $\mathbf{R}$ represents a non-central χ^2 distribution or (for $m > 1$) a non-central Wishart distribution. When $\bar{\boldsymbol{\phi}}_i$ and/or $\mathbf{\Sigma}_i$ are not constant, then $\mathbf{R}$ represents a time-varying non-central χ^2 distribution or a time-varying non-central Wishart distribution.

It is easy to derive the expected value of these distributions due to the linearity of the expectation operator:

$$\begin{aligned}
E[\mathbf{R}] &= E\left[\sum_{i=1}^{N} \boldsymbol{\phi}_i \boldsymbol{\phi}_i{}^T\right] \\
&= \sum_{i=1}^{N} E[\boldsymbol{\phi}_i \boldsymbol{\phi}_i{}^T] \\
&= \sum_{i=1}^{N} \left(\mathbf{\Sigma}_i + \bar{\boldsymbol{\phi}}_i \bar{\boldsymbol{\phi}}_i^T\right) && (4.50) \\
&= N(\mathbf{\Sigma} + \bar{\boldsymbol{\phi}}\bar{\boldsymbol{\phi}}^T) \quad \text{when } \bar{\boldsymbol{\phi}}_i \text{ and } \mathbf{\Sigma}_i \text{ are constant} && (4.51)
\end{aligned}$$

Figure 4-1 shows uncorrelated white noise ($m = 1$, $\bar{\phi} = 0$, Σ is constant) and the corresponding information R from a Monte Carlo simulation. The variance of R is plotted in the figure as well; the analytic solution is derived in the following pages, with the final result shown in (4.71).

The expected value of $\mathbf{R}$ as derived in (4.50) and (4.51) is valid even when the samples are correlated in time ($\text{cov}(\boldsymbol{\phi}_i, \boldsymbol{\phi}_j) \neq \mathbf{0}$ for $i \neq j$, which is equivalent to $\mathbf{a} \neq \mathbf{0}$), although in that case $\mathbf{R}$ does not meet the definition of the χ^2 or Wishart

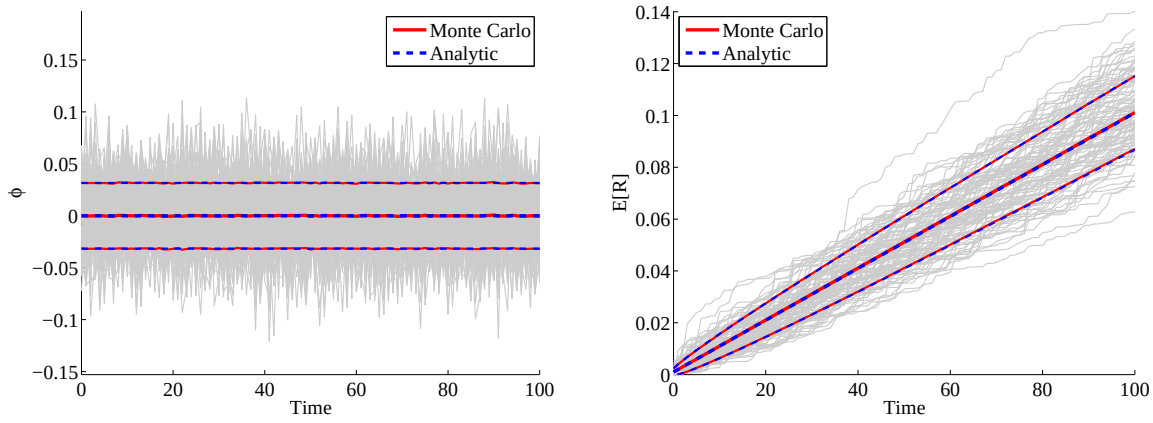

Figure 4-1: Uncorrelated white noise (left) and the information R (right) from a batch of 10,000 Monte Carlo simulations. 100 representative trajectories are shown, along with the mean and standard deviation as calculated from the data (red) and predicted from the analytic model (blue dashed).

distribution.

4.3.4 Inverse χ^2 and Inverse Wishart Distributions

For the parameter convergence predictions we are interested in $E[\mathbf{R}^{-1}]$ rather than $E[\mathbf{R}]$. If $\mathbf{R}$ meets all of the requirements for a χ^2 distribution (scalar, zero-mean, constant variance, and uncorrelated), then R^{-1} represents an inverse χ^2 distribution. Similarly if $\mathbf{R}$ represents a Wishart distribution, then $\mathbf{R}^{-1}$ represents an inverse Wishart distribution. One can derive (at length) the expected value of the inverse Wishart distribution [60]:

$$\begin{aligned} E[\mathbf{R}^{-1}] &= \frac{1}{N-m-1}\mathbf{\Sigma}^{-1} \\ &= \frac{N}{N-m-1}E[\mathbf{R}]^{-1} \end{aligned} \tag{4.52}$$

The inverse χ^2 distribution is a special case of the inverse Wishart distribution in which $m = 1$, so the expected value of the inverse χ^2 distribution is $E[R^{-1}] = \Sigma^{-1}/(N-2)$.

4.3.5 Inverse Non-central χ^2 and Wishart Distributions

In the scalar, non-central, uncorrelated case (which is steady by definition), we can use a Taylor series approximation to derive $E[R^{-1}]$. R is decomposed into two parts, $\bar{R} = N\bar{\phi}\bar{\phi}^T$ and $\tilde{R} = \sum_{i=1}^{N} \tilde{\phi}_i\tilde{\phi}_i^T$. If we assume that $\tilde{R}/\bar{R} < 1$, then we can make the following approximation:

$$\begin{aligned} E[R^{-1}] &= E\left[\frac{1}{\bar{R}+\tilde{R}}\right] \\ &\approx E\left[\frac{1}{\bar{R}}\left(1-\frac{\tilde{R}}{\bar{R}}\right)\right] \\ &= \frac{1}{\bar{R}} - \frac{E[\tilde{R}]}{\bar{R}^2} \end{aligned} \tag{4.53}$$

The multidimensional version of (4.53) is $E[\mathbf{R}^{-1}] \approx \bar{\mathbf{R}}^{-1} - \bar{\mathbf{R}}^{-1}E[\tilde{\mathbf{R}}]\bar{\mathbf{R}}^{-1}$, but it is not numerically stable because $\bar{\mathbf{R}}$ does not have full rank for the Wishart distribution. Empirically, the inverse non-central Wishart distribution has the same expected value as the inverse central Wishart distribution using the formulation in the second line of (4.52).

4.3.6 Summary of the Solution to $E[\mathbf{R}^{-1}]$

Here we present the final result for $E[\mathbf{R}^{-1}]$ for a multidimensional system with correlations. The remainder of this section is devoted to the derivation of this result.

Because the $m \times m$ matrix $\mathbf{R}^{-1}$ is symmetric, it can be represented by a vector $\mathbf{f}(\mathbf{R})$ whose length is $m(m+1)/2$. For example, for a system with $m = 2$, $E[\mathbf{R}^{-1}]$ is formed as follows:

$$E[\mathbf{R}^{-1}] = \begin{bmatrix} E[\mathbf{f}(\mathbf{R})_1] & E[\mathbf{f}(\mathbf{R})_2] \\ E[\mathbf{f}(\mathbf{R})_2] & E[\mathbf{f}(\mathbf{R})_3] \end{bmatrix} \tag{4.54}$$

The expected value of each element in (4.54) is a function of the covariance between

each of the elements of $E[\mathbf{R}]$. For example, for a system with $m = 2$, $E[\mathbf{f}(\mathbf{R})_k]$ is:

$$\begin{aligned} E[\mathbf{f}(\mathbf{R})_k] &= \mathbf{f}(E[\mathbf{R}])_k + \frac{1}{2}\left(\mathrm{var}(R_{11})\mathbf{H}_{k,11} + \mathrm{var}(R_{12})\mathbf{H}_{k,22} + \mathrm{var}(R_{22})\mathbf{H}_{k,33}\right) \\ &\quad + \mathrm{cov}(R_{11}, R_{12})\mathbf{H}_{k,12} + \mathrm{cov}(R_{11}, R_{22})\mathbf{H}_{k,13} + \mathrm{cov}(R_{12}, R_{22})\mathbf{H}_{k,23} \end{aligned} \quad (4.55)$$

where, for example, R_{12} is the element in the first row and second column of $E[\mathbf{R}]$, and $\mathbf{H}_k$ is a Hessian matrix defined in Section 4.3.8 and Equations (4.80-4.82). For $m \neq 2$, $E[\mathbf{f}(\mathbf{R})_k]$ has the same form as (4.55) with summations over the variances and covariances of the elements of $E[\mathbf{R}]$.

The variances and covariances can be generalized to a covariance $\mathrm{cov}(R_a, R_b)$ where R_a and R_b are the various elements of $\mathbf{R}$ extracted by unit vectors $\mathbf{I}_{1\ldots4}$, each specific to the locations of R_a and R_b within $\mathbf{R}$: $R_a = {\mathbf{I}_1}^T\mathbf{R}\mathbf{I}_2$ and $R_b = {\mathbf{I}_3}^T\mathbf{R}\mathbf{I}_4$. The covariance for each combination of R_a and R_b is evaluated with the following sum:

$$\begin{aligned} \mathrm{cov}(R_a, R_b) &= \sum_{i=1}^{N} \big({\mathbf{I}_1}^T\bar{\boldsymbol{\phi}}_i\bar{\boldsymbol{\phi}}_i^T\mathbf{I}_3{\mathbf{I}_2}^T\boldsymbol{\Gamma}_{0,i}\mathbf{I}_4 + {\mathbf{I}_1}^T\bar{\boldsymbol{\phi}}_i\bar{\boldsymbol{\phi}}_i^T\mathbf{I}_4{\mathbf{I}_2}^T\boldsymbol{\Gamma}_{0,i}\mathbf{I}_3 \\ &\quad + {\mathbf{I}_2}^T\bar{\boldsymbol{\phi}}_i\bar{\boldsymbol{\phi}}_i^T\mathbf{I}_3{\mathbf{I}_1}^T\boldsymbol{\Gamma}_{0,i}\mathbf{I}_4 + {\mathbf{I}_2}^T\bar{\boldsymbol{\phi}}_i\bar{\boldsymbol{\phi}}_i^T\mathbf{I}_4{\mathbf{I}_1}^T\boldsymbol{\Gamma}_{0,i}\mathbf{I}_3 \\ &\quad + {\mathbf{I}_1}^T\boldsymbol{\Gamma}_{1,i}\boldsymbol{\Sigma}_i\mathbf{I}_4 + {\mathbf{I}_1}^T\boldsymbol{\Gamma}_{2,i}\boldsymbol{\Sigma}_i\mathbf{I}_3 + {\mathbf{I}_3}^T\boldsymbol{\Gamma}_{3,i}\boldsymbol{\Sigma}_i\mathbf{I}_2 + {\mathbf{I}_4}^T\boldsymbol{\Gamma}_{4,i}\boldsymbol{\Sigma}_i\mathbf{I}_2 \\ &\quad - {\mathbf{I}_1}^T\boldsymbol{\Sigma}_i\mathbf{I}_3{\mathbf{I}_2}^T\boldsymbol{\Sigma}_i\mathbf{I}_4 - {\mathbf{I}_1}^T\boldsymbol{\Sigma}_i\mathbf{I}_4{\mathbf{I}_2}^T\boldsymbol{\Sigma}_i\mathbf{I}_3\big) \end{aligned} \quad (4.56)$$

where $\boldsymbol{\Gamma}_{0\ldots4,i}$ are listed below.

$$\begin{aligned} \boldsymbol{\Gamma}_{0,i} &= (\mathbf{I} - \mathbf{a})^{-1}(\mathbf{I} - \mathbf{a}^i)\boldsymbol{\Sigma}_i + \boldsymbol{\Sigma}_i(\mathbf{I} - (\mathbf{a}^i)^T)(\mathbf{I} - \mathbf{a})^{-T} - \boldsymbol{\Sigma}_i \\ \boldsymbol{\Gamma}_{1,i} &= \mathtt{dlyap}(\mathbf{a},\ \mathbf{a},\ \boldsymbol{\Sigma}_i\mathbf{I}_3{\mathbf{I}_2}^T - \mathbf{a}^i\boldsymbol{\Sigma}_i\mathbf{I}_3{\mathbf{I}_2}^T\mathbf{a}^i) \\ \boldsymbol{\Gamma}_{2,i} &= \mathtt{dlyap}(\mathbf{a},\ \mathbf{a},\ \boldsymbol{\Sigma}_i\mathbf{I}_4{\mathbf{I}_2}^T - \mathbf{a}^i\boldsymbol{\Sigma}_i\mathbf{I}_4{\mathbf{I}_2}^T\mathbf{a}^i) \\ \boldsymbol{\Gamma}_{3,i} &= \mathtt{dlyap}(\mathbf{a},\ \mathbf{a},\ \boldsymbol{\Sigma}_i\mathbf{I}_1{\mathbf{I}_4}^T - \mathbf{a}^i\boldsymbol{\Sigma}_i\mathbf{I}_1{\mathbf{I}_4}^T\mathbf{a}^i) \\ \boldsymbol{\Gamma}_{4,i} &= \mathtt{dlyap}(\mathbf{a},\ \mathbf{a},\ \boldsymbol{\Sigma}_i\mathbf{I}_1{\mathbf{I}_3}^T - \mathbf{a}^i\boldsymbol{\Sigma}_i\mathbf{I}_1{\mathbf{I}_3}^T\mathbf{a}^i) \end{aligned} \quad (4.57)$$

The overall procedure is listed in Algorithm 9. Section 4.3.7 provides the details of the derivation for a scalar ($m = 1$) system, and Section 4.3.8 provides the full

derivation for a multiparameter system.

Algorithm 9 Procedure for computing $E[\mathbf{R}^{-1}]$.

1: **for** $i = 1 : N$ **do**
2: Compute the three-dimensional Hessian matrix $\mathbf{H}$ (Section 4.3.8).
3: **for** each combination of a and b **do**
4: Add the new contribution to $\text{cov}(R_a, R_b)$ using the state statistics at t_i (4.56-4.57).
5: **end for**
6: **end for**
7: Evaluate $E[\mathbf{f}(\mathbf{R})_k]$ for each element k (4.55).
8: Form $E[\mathbf{R}^{-1}]$ using each element $E[\mathbf{f}(\mathbf{R})_k]$ (4.54).

4.3.7 Central Limit Theorem for a Scalar System with Correlation

The learning data for the planar holonomic vehicle, and for the simplified system model (4.37), is correlated through time. This correlation does not affect $E[\mathbf{R}]$, but it does corrupt the prediction of $E[\mathbf{R}^{-1}]$ using the inverse Wishart distribution (4.52). However, if we can predict the mean and variance of $\mathbf{R}$, then we can use the central limit theorem to approximate $\mathbf{R}$ as a normally distributed random variable. Next it is a simple process to find the mean of the inverse of the distribution using the Taylor series expansion of $\mathbf{R}^{-1}$ when $m = 1$; the process is more difficult when $m > 1$. In this section we focus on the scalar ($m = 1$) case. We define the inverse function $f(R)$:

$$f(R) = R^{-1} \qquad f' = -R^{-2} \qquad f''(R) = 2R^{-3} \tag{4.58}$$

The Taylor series for $f(R)$ about the mean $\bar{R} \equiv E[R]$ is:

$$\begin{aligned} R^{-1} &\approx f(\bar{R}) + f'(\bar{R})(R - \bar{R}) + \frac{1}{2} f''(\bar{R})(R - \bar{R})^2 \\ &= \bar{R}^{-1} - \bar{R}^{-2}(R - \bar{R}) + \bar{R}^{-3}(R - \bar{R})^2 \\ E[R^{-1}] &= \bar{R}^{-1} - \bar{R}^{-2}(E[R] - \bar{R}) + \bar{R}^{-3} E\left[(R - \bar{R})^2\right] \\ &= E[R]^{-1}\left(1 + E[R]^{-2}\,\text{var}(R)\right) \end{aligned} \tag{4.59}$$

We already have an expression for $E[R]$ (4.50), so we need to find an expression for $\text{var}(R)$ in order to approximate $E[R^{-1}]$. From the definition, $\text{var}(R)$ is:

$$\begin{aligned}
\text{var}(R) &= E[R^2] - E[R]^2 \\
&= E\left[\left(\sum_{i=1}^{N} {\phi_i}^2\right)^2\right] - E[R]^2 \\
&= \sum_{i=1}^{N}\sum_{j=1}^{N} E\left[{\phi_i}^2{\phi_j}^2\right] - E[R]^2 \qquad (4.60) \\
\text{where} \quad E[R]^2 &= \sum_{i=1}^{N}\sum_{j=1}^{N}(\Sigma_i + \bar{\phi}_i^2)(\Sigma_j + \bar{\phi}_j^2) \qquad (4.61)
\end{aligned}$$

Each term of the expectation in (4.60) can be expanded to reveal its mean and finite sum:

$$\begin{aligned}
E[{\phi_i}^2{\phi_j}^2] &= E\left[\left(\bar{\phi}_i + \sum_{k_1=0}^{i-1} a^{k_1} w_{i-k_1-1}\right)\left(\bar{\phi}_i + \sum_{k_2=0}^{i-1} a^{k_2} w_{i-k_2-1}\right)\right. \\
&\qquad \left.\left(\bar{\phi}_j + \sum_{k_3=0}^{j-1} a^{k_3} w_{j-k_3-1}\right)\left(\bar{\phi}_j + \sum_{k_4=0}^{j-1} a^{k_4} w_{j-k_4-1}\right)\right] \\
&= E\left[\bar{\phi}_i^2\bar{\phi}_j^2 + \bar{\phi}_i^2 S_3 S_4 + S_1 S_2 \bar{\phi}_j^2 + 4\bar{\phi}_i\bar{\phi}_j S_1 S_3 + S_1 S_2 S_3 S_4\right]
\end{aligned} \qquad (4.62)$$

In (4.62), S_n refers to the summation corresponding to the index k_n. Cross-terms that evaluate to zero in the expectation have been omitted for clarity. The remaining terms are computed below. Recall that $V_{ij} = \text{cov}(\phi_i, \phi_j)$.

$$\begin{aligned}
E[S_1 S_2] &= \Sigma_i \\
E[S_3 S_4] &= \Sigma_j \qquad (4.63) \\
E[S_1 S_3] &= V_{ij}
\end{aligned}$$

For the term $E[S_1 S_2 S_3 S_4]$ in (4.62), we must consider four different cases depending on how the indices combine. First note the following properties of a normally

distributed random variable w:

$$E[w_m w_n w_p w_q] = \begin{cases} E[{w_m}^4] = 3W^2 & \text{if } m = n = p = q \\ E[{w_m}^2]E[{w_p}^2] = W^2 & \text{if } m = n \text{ and } p = q \text{ and } m \neq p \\ 0 & \text{otherwise} \end{cases} \quad (4.64)$$

Using (4.64), the four different cases for $E[S_1 S_2 S_3 S_4]$ are listed below.

- 12,34: $k_1 = k_2$, $k_3 = k_4$, $k_1 \neq k_3 + i - j$. Both complete sums are computed, then the particular sum for which $k_1 = k_3 + i - j$ is subtracted out. In the first line we assume $i > j$, but this is relaxed in the second line.

$$\begin{aligned} E[S_1 S_2 S_3 S_4] &= \sum_{k_1=0}^{i-1} a^{2k_1} W \times \sum_{k_3=0}^{j-1} a^{2k_3} W - a^{i-j} \sum_{k_3=0}^{j-1} a^{4k_3} W^2 \\ &= \Sigma_i \Sigma_j - \frac{a^{2|i-j|} W^2}{1 - a^4} (1 - a^{4\min(i,j)}) \end{aligned}$$

- 13,24: $k_1 = k_3 + i - j$, $k_2 = k_4 + i - j$, $k_1 \neq k_2$. Both complete sums are computed, then the particular sum for which $k_1 = k_2$ (equivalently $k_3 = k_4$) is subtracted out.

$$\begin{aligned} E[S_1 S_2 S_3 S_4] &= a^{i-j} \sum_{k_3=0}^{j-1} a^{2k_3} W \times a^{i-j} \sum_{k_4=0}^{j-1} a^{2k_4} W - a^{i-j} \sum_{k_3=0}^{j-1} a^{4k_3} W^2 \\ &= {V_{ij}}^2 - \frac{a^{2|i-j|} W^2}{1 - a^4} (1 - a^{4\min(i,j)}) \end{aligned}$$

- 14,23: $k_1 = k_4 + i - j$, $k_2 = k_3 + i - j$, $k_1 \neq k_2$. Both complete sums are computed, then the particular sum for which $k_1 = k_2$ (equivalently $k_3 = k_4$) is subtracted out.

$$\begin{aligned} E[S_1 S_2 S_3 S_4] &= a^{i-j} \sum_{k_4=0}^{j-1} a^{2k_4} W \times a^{i-j} \sum_{k_3=0}^{j-1} a^{2k_3} W - a^{i-j} \sum_{k_4=0}^{j-1} a^{4k_4} W^2 \\ &= {V_{ij}}^2 - \frac{a^{2|i-j|} W^2}{1 - a^4} (1 - a^{4\min(i,j)}) \end{aligned}$$

- 1234: $k_1 = k_2$, $k_3 = k_4$, $k_1 = k_3 + i - j$. A single sum is computed for which all four indices refer to the same noise sample.

$$\begin{aligned} E[S_1 S_2 S_3 S_4] &= 3W^2 \sum_{k_3=0}^{j-1} a^{4k_3+2(i-j)} \\ &= \frac{3W^2 a^{2|i-j|}}{1-a^4} \left(1 - a^{4\min(i,j)}\right) \end{aligned}$$

Combining all of the terms, we arrive at:

$$\begin{aligned} E[{\phi_i}^2 {\phi_j}^2] &= \bar{\phi}_i^2 \bar{\phi}_j^2 + \bar{\phi}_i^2 \Sigma_j + \bar{\phi}_j^2 \Sigma_i + 4\bar{\phi}_i \bar{\phi}_j V_{ij} + \Sigma_i \Sigma_j + 2{V_{ij}}^2 \alpha \\ &= (\Sigma_i + \bar{\phi}_i^2)(\Sigma_j + \bar{\phi}_j^2) + 4\bar{\phi}_i \bar{\phi}_j V_{ij} + 2{V_{ij}}^2 \end{aligned} \tag{4.65}$$

Next we plug (4.65) into (4.60) to solve for $\mathrm{var}(R)$.

$$\begin{aligned} \mathrm{var}(R) &= \sum_{i=1}^{N} \sum_{j=1}^{N} E[{\phi_i}^2 {\phi_j}^2] - E[R]^2 \\ &= \sum_{i=1}^{N} \sum_{j=1}^{N} \left(4\bar{\phi}_i \bar{\phi}_j V_{ij} + 2{V_{ij}}^2\right) + \cancel{E[R]^2 - E[R]^2} \\ &= \sum_{i=1}^{N} \sum_{j=1}^{N} \left(4\bar{\phi}_i \bar{\phi}_j \, a^{|i-j|} \Sigma_{\min(i,j)} + 2\, a^{2|i-j|} {\Sigma_{\min(i,j)}}^2\right) \end{aligned} \tag{4.66}$$

We can evaluate the double sum in (4.66) for $i \geq j$ (with a new index $k_1 = i - j$), then for $j \geq i$ (with a new index $k_2 = j - i$), then to avoid double-counting the

diagonal we subtract one sum for $i = j$.

$$\begin{aligned}
\mathrm{var}(R) &= \sum_{i=1}^{N}\sum_{k_1=0}^{i-1}\left(4\bar{\phi}_i\bar{\phi}_{i-k_1}a^{k_1}\Sigma_{i-k_1} + 2a^{2k_1}{\Sigma_{i-k_1}}^2\right) \\
&\quad + \sum_{j=1}^{N}\sum_{k_2=0}^{j-1}\left(4\bar{\phi}_{j-k_2}\bar{\phi}_j a^{k_2}\Sigma_{j-k_2} + 2a^{2k_2}{\Sigma_{j-k_2}}^2\right) \\
&\quad - \sum_{i=1}^{N}\left(4\bar{\phi}_i^2\Sigma_i + 2{\Sigma_i}^2\right) \\
&\approx 2\sum_{i=1}^{N}\sum_{k_1=0}^{i-1}\left(4\bar{\phi}_i^2\Sigma_i a^{k_1} + 2{\Sigma_i}^2 a^{2k_1}\right) - \sum_{i=1}^{N}\left(4\bar{\phi}_i^2\Sigma_i + 2{\Sigma_i}^2\right) \qquad (4.67)
\end{aligned}$$

The approximation on the last line of (4.67) is valid if Σ_i does not change quickly with respect to the time constant of the filter a. Next we use the summation rule shown in (4.39) to evaluate the inner sum.

$$\begin{aligned}
\mathrm{var}(R) &= 2\sum_{i=1}^{N}\left(4\bar{\phi}_i^2\Sigma_i\frac{1-a^i}{1-a} + 2{\Sigma_i}^2\frac{1-a^{2i}}{1-a^2}\right) - \sum_{i=1}^{N}\left(4\bar{\phi}_i^2\Sigma_i + 2{\Sigma_i}^2\right) \\
&= \sum_{i=1}^{N}\left(8\bar{\phi}_i^2\Sigma_i\left(\frac{1-a^i}{1-a} - \frac{1}{2}\right) + 4{\Sigma_i}^2\left(\frac{1-a^{2i}}{1-a^2} - \frac{1}{2}\right)\right) \qquad (4.68)
\end{aligned}$$

When the mean and variance are steady, then $\bar{\phi}_i = \bar{\phi}$ and $\Sigma_i = \Sigma$. In that case we use the summation rules again to evaluate the variance of R:

$$\begin{aligned}
\mathrm{var}(R) &= \frac{8\bar{\phi}^2\Sigma}{1-a}\left(N - \frac{1-a^N}{1-a} + (1-a^N) - N\frac{1-a}{2}\right) \\
&\quad + \frac{4\Sigma^2}{1-a^2}\left(N - \frac{1-a^{2N}}{1-a^2} + (1-a^{2N}) - N\frac{1-a^2}{2}\right) \qquad (4.69)
\end{aligned}$$

When N is large, (4.69) reduces to:

$$\mathrm{var}(R)_{N\gg 1} = 8\bar{\phi}^2\Sigma N\left(\frac{1}{1-a} - \frac{1}{2}\right) + 4\Sigma^2 N\left(\frac{1}{1-a^2} - \frac{1}{2}\right) \qquad (4.70)$$

Note that when there is no correlation between states ($a = 0$), the variance (4.69)

reduces to:

$$\text{var}(R)_{a=0} = 4\bar{\phi}^2\Sigma N + 2\Sigma^2 N \tag{4.71}$$

The zero-mean (central) version of (4.71), $2\Sigma^2 N$, corresponds to the variance of the χ^2 distribution. The Taylor series approximation for $E[R^{-1}]$ (4.59) using this value is shown below.

$$\begin{aligned} E[R^{-1}] &= E[R]^{-1}\left(1 + E[R]^{-2}\,\text{var}(R)\right) \\ &= E[R]^{-1}\left(1 + \frac{2N\Sigma^2}{N^2\Sigma^2}\right) \\ &= E[R]^{-1}\left(1 + \frac{2}{N}\right) \end{aligned} \tag{4.72}$$

From (4.52), the exact expected value of the inverse χ^2 distribution is $E[R^{-1}] = E[R]^{-1}\frac{N}{N-2}$. The Taylor series of this expression is:

$$\begin{aligned} E[R^{-1}] &= E[R]^{-1}\frac{N}{N-2} \\ &= E[R]^{-1}\frac{1}{1-2/N} \\ &= E[R]^{-1}\left(1 + \frac{2}{N} + \frac{4}{N^2} + \frac{8}{N^3} + \cdots\right) \end{aligned} \tag{4.73}$$

Recall that the central limit theorem approach (4.72) uses the first two terms of the Taylor series expansion of the inverse function. As we might expect, that result matches the first two terms of the Taylor series expansion of the exact solution (4.73). A comparison of these results is shown in Figure 4-2.

The power of the central limit theorem approach is that it applies to the non-central case and correlated systems. The first important result is an approximation of the inverse non-central χ^2 distribution. If the square of the mean is much larger than the variance, $\bar{\phi}^2 \gg \Sigma$, then $E[R] \approx N\bar{\phi}^2$ and the expected value of the inverse

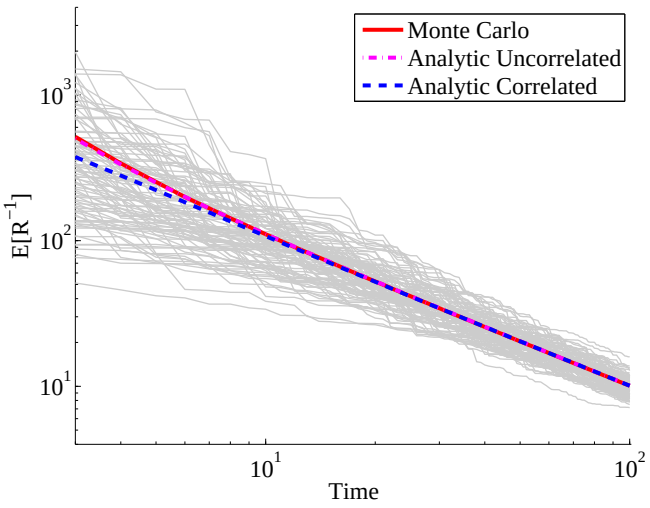

Figure 4-2: $E[R^{-1}]$ for uncorrelated white noise (Figure 4-1) using the exact expected value of the inverse χ^2 distribution (equation 4.52, magenta dash-dot) and the central limit theorem approach (Equation 4.72, blue dashed). 100 representative Monte Carlo results for R^{-1} are shown, and the expected value of all 10,000 Monte Carlo results is shown as a red curve.

of R is:

$$\begin{aligned} E[R^{-1}] &= E[R]^{-1}\left(1+\frac{4\bar{\phi}^2\Sigma N+2\Sigma^2 N}{\bar{\phi}^4N^2}\right) \\ &= E[R]^{-1}\left(1+\frac{4\Sigma}{\bar{\phi}^2N}+\frac{2\Sigma^2}{\bar{\phi}^4N}\right) \quad \ll 1 \\ &\approx E[R]^{-1} \end{aligned} \tag{4.74}$$

Equation (4.74) indicates that when the mean is significantly larger than the standard deviation of the input data, then the zeroth-order approximation of the inverse is valid.

Figure 4-3 shows the advantage of using the central limit theorem approach for a system with correlation. In this example the data is zero-mean and steady with $a = 0.95$. The central limit approach with the variance of R calculated from (4.69) converges to the true result after a dozen sample points, while the uncorrelated prediction (4.52) still has not converged after 100 sample points.

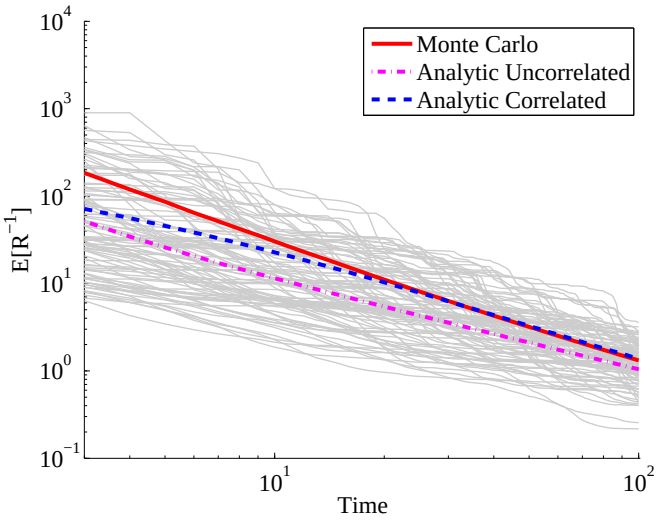

Figure 4-3: $E[R^{-1}]$ for a steady zero-mean correlated system with $a = 0.95$ using the uncorrelated prediction from the inverse χ^2 distribution (equation 4.52, magenta dash-dot) and the central limit theorem approach (Equation 4.69, blue dashed). The correlated prediction converges after a dozen sample points. 100 representative Monte Carlo results for R^{-1} are shown, and the expected value of all 10,000 Monte Carlo results is shown as a red curve.

Figure 4-4 shows the data and information evolution for a correlated system with $a = 0.95$ and a time-varying mean and variance. The correlated and uncorrelated predictions are shown in Figure 4-5.

4.3.8 Central Limit Theorem for a Multiparameter System

To extend the result from Section 4.3.7 to multiparameter systems ($m > 1$), it is necessary to find an equivalent Taylor series approximation for $\mathbf{R}^{-1}$ around $E[\mathbf{R}] \equiv \bar{\mathbf{R}}$. However, the Taylor series is not defined for matrices. To get around this problem, the matrix $\mathbf{R}$ can be reformed as a vector (simplified by the fact that $\mathbf{R}$ is symmetric) and then we can compute the Taylor series of this vector before reforming the matrix. This process is demonstrated for the case when $\mathbf{R}$ is a 2×2 matrix, but the concept can be extended to $m = 3$ or higher as well.

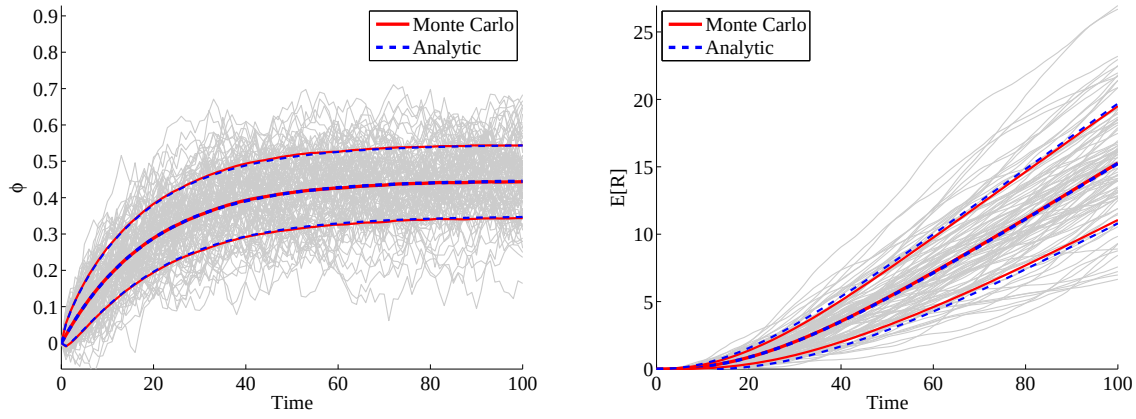

Figure 4-4: 100 trajectories from a correlated system with $a = 0.95$ (left) and the information R (right). The mean and standard deviation from a 10,000-point Monte Carlo simulation are shown (red), along with the mean and standard deviation as predicted from the analytic model (blue dashed).

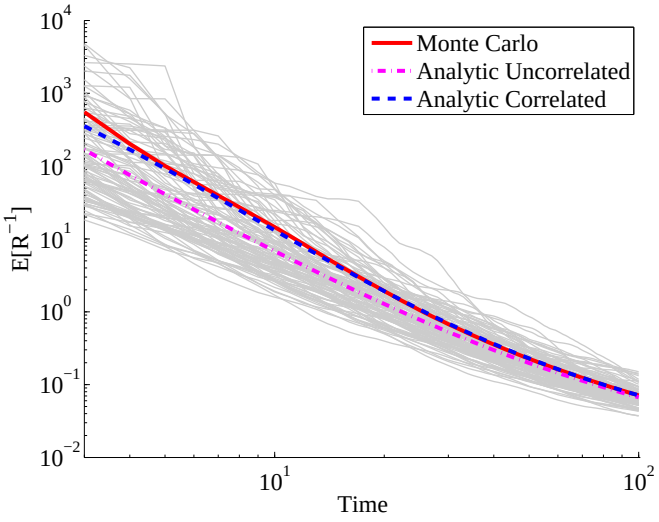

Figure 4-5: $E[R^{-1}]$ for the data shown in Figure 4-4 using the uncorrelated prediction from the inverse χ^2 distribution (Equation 4.52, magenta dash-dot) and the central limit theorem approach (equation 4.69, blue dashed). 100 representative Monte Carlo results for R^{-1} are shown, and the expected value of all 10,000 Monte Carlo results is shown as a red curve.

First we reform the matrix $\mathbf{R}$ as a vector $\mathbf{r}$ whose length is $m(m+1)/2$.

$$\mathbf{R} = \begin{bmatrix} R_{11} & R_{12} \\ R_{12} & R_{22} \end{bmatrix} \qquad \mathbf{r} = \begin{bmatrix} R_{11} \\ R_{12} \\ R_{22} \end{bmatrix} \tag{4.75}$$

Next we note the inverse of a 2×2 matrix:

$$\mathbf{R}^{-1} = \begin{bmatrix} R_{11} & R_{12} \\ R_{12} & R_{22} \end{bmatrix}^{-1} = \frac{1}{|R|} \begin{bmatrix} R_{22} & -R_{12} \\ -R_{12} & R_{11} \end{bmatrix} \quad \text{where} \quad |\mathbf{R}| = R_{11}R_{22} - {R_{12}}^2 \tag{4.76}$$

If $\mathbf{r}$ is the vector form of $\mathbf{R}$, then we define a new function $\mathbf{f}(\mathbf{r})$ to be the vector form of $\mathbf{R}^{-1}$. Using (4.76), we can write $\mathbf{f}(\mathbf{r})$ as follows:

$$\mathbf{f}(\mathbf{r}) = \frac{1}{|\mathbf{R}|} \begin{bmatrix} R_{22} \\ -R_{12} \\ R_{11} \end{bmatrix} = \frac{1}{R_{11}R_{22} - {R_{12}}^2} \begin{bmatrix} R_{22} \\ -R_{12} \\ R_{11} \end{bmatrix} \tag{4.77}$$

For each element k of $\mathbf{f}(\mathbf{r})$ we can write the first three terms of the Taylor series approximation around $\bar{\mathbf{r}}$. Note that each term in this equation is a scalar.

$$\mathbf{f}(\mathbf{r})_k \approx \mathbf{f}(\bar{\mathbf{r}})_k + \left.\frac{\partial \mathbf{f}(\mathbf{r})_k}{\partial \mathbf{r}}\right|_{\bar{\mathbf{r}}} (\mathbf{r} - \bar{\mathbf{r}}) + \frac{1}{2}(\mathbf{r} - \bar{\mathbf{r}})^T \left.\frac{\partial^2 \mathbf{f}(\mathbf{r})_k}{\partial \mathbf{r}^2}\right|_{\bar{\mathbf{r}}} (\mathbf{r} - \bar{\mathbf{r}}) \tag{4.78}$$

In (4.78), $\left.\frac{\partial \mathbf{f}(\mathbf{r})_k}{\partial \mathbf{r}}\right|_{\bar{\mathbf{r}}}$ is the gradient of $\mathbf{f}(\mathbf{r})_k$ evaluated at $\mathbf{r} = \bar{\mathbf{r}}$ and $\left.\frac{\partial^2 \mathbf{f}(\mathbf{r})_k}{\partial \mathbf{r}^2}\right|_{\bar{\mathbf{r}}}$ is the Hessian of $\mathbf{f}(\mathbf{r})_k$ evaluated at $\mathbf{r} = \bar{\mathbf{r}}$. The gradient vector for each element k can be summarized in the Jacobian matrix, shown below, in which $\left.\frac{\partial \mathbf{f}(\mathbf{r})_k}{\partial \mathbf{r}}\right|_{\bar{\mathbf{r}}}$ is the kth row of the matrix and all of the matrix elements are taken from $\bar{\mathbf{R}}$.

$$\left.\frac{\partial \mathbf{f}(\mathbf{r})}{\partial \mathbf{r}}\right|_{\bar{\mathbf{r}}} = \frac{1}{|\mathbf{R}|^2} \begin{bmatrix} -{R_{22}}^2 & 2R_{12}R_{22} & -{R_{12}}^2 \\ R_{12}R_{22} & -R_{11}R_{22} - {R_{12}}^2 & R_{11}R_{12} \\ -{R_{12}}^2 & 2R_{11}R_{12} & -{R_{11}}^2 \end{bmatrix} \tag{4.79}$$

The Hessian of $\mathbf{f}(\mathbf{r})$ must be computed separately for each element k of $\mathbf{f}(\mathbf{r})$. The

Hessians for the three elements are:

$$\mathbf{H}_1 = \left.\frac{\partial^2 \mathbf{f}(\mathbf{r})_1}{\partial \mathbf{r}^2}\right|_{\bar{\mathbf{r}}} = \frac{1}{|\mathbf{R}|^3}\begin{bmatrix} 2{R_{22}}^3 & -4R_{12}{R_{22}}^2 & 2{R_{12}}^2R_{22} \\ -4R_{12}{R_{22}}^2 & 2R_{22}C_2 & -2R_{12}C_1 \\ 2{R_{12}}^2R_{22} & -2R_{12}C_1 & 2{R_{12}}^2R_{11} \end{bmatrix} \tag{4.80}$$

$$\mathbf{H}_2 = \left.\frac{\partial^2 \mathbf{f}(\mathbf{r})_2}{\partial \mathbf{r}^2}\right|_{\bar{\mathbf{r}}} = \frac{1}{|\mathbf{R}|^3}\begin{bmatrix} -2R_{12}{R_{22}}^2 & R_{22}C_2 & -R_{12}C_1 \\ R_{22}C_2 & -2R_{12}(3R_{11}R_{22} + {R_{12}}^2) & R_{11}C_2 \\ -R_{12}C_1 & R_{11}C_2 & -2R_{12}{R_{11}}^2 \end{bmatrix} \tag{4.81}$$

$$\mathbf{H}_3 = \left.\frac{\partial^2 \mathbf{f}(\mathbf{r})_3}{\partial \mathbf{r}^2}\right|_{\bar{\mathbf{r}}} = \frac{1}{|\mathbf{R}|^3}\begin{bmatrix} 2{R_{12}}^2R_{22} & -2R_{12}C_1 & 2R_{11}{R_{12}}^2 \\ -2R_{12}C_1 & 2R_{11}C_2 & -4{R_{11}}^2R_{12} \\ 2R_{11}{R_{12}}^2 & -4{R_{11}}^2R_{12} & 2{R_{11}}^3 \end{bmatrix} \tag{4.82}$$

where $C_1 = {R_{12}}^2 + R_{11}R_{22}$ and $C_2 = 3{R_{12}}^2 + R_{11}R_{22}$. The full Hessian is a three-dimensional matrix with dimensions $3 \times 3 \times 3$ for $m = 2$. Note the symmetry of the Hessian across all three dimensions. For $m > 2$ it is easier to compute the Hessian matrices numerically using a finite difference approach with very small perturbations of each element from the mean value.

As in the scalar case, the next step is to take the expected value of the Taylor series (4.78) to evaluate $E[\mathbf{R}^{-1}]$. In the expectation, the second term of (4.78) is zero because $E[\mathbf{r}] = E[\bar{\mathbf{r}}] = \bar{\mathbf{r}}$, leaving the first and third terms. For each element k:

$$E[\mathbf{f}(\mathbf{r})_k] = \mathbf{f}(\bar{\mathbf{r}})_k + E\left[\frac{1}{2}(\mathbf{r} - \bar{\mathbf{r}})^T \mathbf{H}_k (\mathbf{r} - \bar{\mathbf{r}})\right] \tag{4.83}$$

where we recall that the elements in each $\mathbf{H}_k$ are taken from $\bar{\mathbf{R}}$. In the following equation, the notation $\mathbf{H}_{k,jl}$ refers to the j'th row and l'th column of the k'th Hessian matrix. For example, $\mathbf{H}_{3,13} = 2R_{11}{R_{12}}^2$. Plugging the matrix elements of $\mathbf{H}_k$ into

(4.83) results in the following equation for each element k:

$$\begin{aligned} E[\mathbf{f}(\mathbf{r})_k] &= \mathbf{f}(\bar{\mathbf{r}})_k + \frac{1}{2}E\left[\tilde{R}_{11}^2\mathbf{H}_{k,11} + \tilde{R}_{12}^2\mathbf{H}_{k,22} + \tilde{R}_{22}^2\mathbf{H}_{k,33}\right. \\ &\quad \left. + \ 2\tilde{R}_{11}\tilde{R}_{12}\mathbf{H}_{k,12} + 2\tilde{R}_{11}\tilde{R}_{22}\mathbf{H}_{k,13} + 2\tilde{R}_{12}\tilde{R}_{22}\mathbf{H}_{k,23}\right] \\ &= \mathbf{f}(\bar{\mathbf{r}})_k + \frac{1}{2}\left(\mathrm{var}(R_{11})\mathbf{H}_{k,11} + \mathrm{var}(R_{12})\mathbf{H}_{k,22} + \mathrm{var}(R_{22})\mathbf{H}_{k,33}\right) \\ &\quad + \mathrm{cov}(R_{11},R_{12})\mathbf{H}_{k,12} + \mathrm{cov}(R_{11},R_{22})\mathbf{H}_{k,13} + \mathrm{cov}(R_{12},R_{22})\mathbf{H}_{k,23} \end{aligned} \tag{4.84}$$

The variances and covariances in (4.84) can be generalized to the covariance $\mathrm{cov}(R_a, R_b)$ in which R_a and R_b are the various combinations of R_{11}, R_{12}, and R_{22}. The matrix elements R_a and R_b can be extracted from the full R matrix using two 2×1 column vectors each: $R_a = \mathbf{I}_1{}^T\mathbf{R}\mathbf{I}_2$ and $R_b = \mathbf{I}_3{}^T\mathbf{R}\mathbf{I}_4$. The column vectors have a value of 1 at the appropriate location corresponding to the row or column of R_a or R_b and zeros everywhere else. For example, if $R_a = R_{12}$ and $R_b = R_{22}$ then $\mathbf{I}_1 = [1\ 0]^T$, $\mathbf{I}_2 = [0\ 1]^T$, $\mathbf{I}_3 = [0\ 1]^T$ and $\mathbf{I}_4 = [0\ 1]^T$. From the definition of the covariance, $\mathrm{cov}(R_a, R_b)$ is:

$$\begin{aligned} \mathrm{cov}(R_a,R_b) &= E\left[R_aR_b\right] - E[R_a]E[R_b] \\ &= E\left[\mathbf{I}_1{}^T\mathbf{R}\mathbf{I}_2\mathbf{I}_3{}^T\mathbf{R}\mathbf{I}_4\right] - \mathbf{I}_1{}^TE[\mathbf{R}]\mathbf{I}_2\mathbf{I}_3{}^TE[\mathbf{R}]\mathbf{I}_4 \\ &= E\left[\mathbf{I}_1{}^T\sum_{i=1}^{N}\boldsymbol{\phi}_i\boldsymbol{\phi}_i{}^T\mathbf{I}_2\mathbf{I}_3{}^T\sum_{j=1}^{N}\boldsymbol{\phi}_j\boldsymbol{\phi}_j{}^T\mathbf{I}_4\right] - \mathbf{I}_1{}^TE[\mathbf{R}]\mathbf{I}_2\mathbf{I}_3{}^TE[\mathbf{R}]\mathbf{I}_4 \\ &= \sum_{i=1}^{N}\sum_{j=1}^{N}E\left[\mathbf{I}_1{}^T\boldsymbol{\phi}_i\boldsymbol{\phi}_i{}^T\mathbf{I}_2\mathbf{I}_3{}^T\boldsymbol{\phi}_j\boldsymbol{\phi}_j{}^T\mathbf{I}_4\right] - \mathbf{I}_1{}^TE[\mathbf{R}]\mathbf{I}_2\mathbf{I}_3{}^TE[\mathbf{R}]\mathbf{I}_4 \end{aligned} \tag{4.85}$$

The argument of the summation in (4.85) is:

$$
\begin{aligned}
& E\left[\mathbf{I}_1{}^T\left(\bar{\boldsymbol{\phi}}_i+\sum_{k_1=0}^{i-1}\mathbf{a}^{k_1}\mathbf{w}_{i-k_1-1}\right)\left(\bar{\boldsymbol{\phi}}_i^T+\sum_{k_2=0}^{i-1}\mathbf{w}_{i-k_2-1}{}^T(\mathbf{a}^{k_2})^T\right)\mathbf{I}_2\right. \\
& \qquad \left.\times\ \mathbf{I}_3{}^T\left(\bar{\boldsymbol{\phi}}_j+\sum_{k_3=0}^{j-1}\mathbf{a}^{k_3}\mathbf{w}_{j-k_3-1}\right)\left(\bar{\boldsymbol{\phi}}_j^T+\sum_{k_4=0}^{j-1}\mathbf{w}_{j-k_4-1}{}^T(\mathbf{a}^{k_4})^T\right)\mathbf{I}_4\right] \\
& = E\left[\mathbf{I}_1{}^T(\bar{\boldsymbol{\phi}}_i+\mathbf{S}_1)(\bar{\boldsymbol{\phi}}_i+\mathbf{S}_2)^T\mathbf{I}_2\mathbf{I}_3{}^T(\bar{\boldsymbol{\phi}}_j+\mathbf{S}_3)(\bar{\boldsymbol{\phi}}_j+\mathbf{S}_4)^T\mathbf{I}_4\right] \\
& = \mathbf{I}_1{}^T\bar{\boldsymbol{\phi}}_i\bar{\boldsymbol{\phi}}_i^T\mathbf{I}_2\mathbf{I}_3{}^T\bar{\boldsymbol{\phi}}_j\bar{\boldsymbol{\phi}}_j^T\mathbf{I}_4+\mathbf{I}_1{}^T\bar{\boldsymbol{\phi}}_i\bar{\boldsymbol{\phi}}_i^T\mathbf{I}_2\mathbf{I}_3{}^TE[\mathbf{S}_3\mathbf{S}_4{}^T]\mathbf{I}_4+\mathbf{I}_1{}^TE[\mathbf{S}_1\mathbf{S}_2{}^T]\mathbf{I}_2\mathbf{I}_3{}^T\bar{\boldsymbol{\phi}}_j\bar{\boldsymbol{\phi}}_j^T\mathbf{I}_4 \\
& \quad + E[\mathbf{I}_1{}^T\bar{\boldsymbol{\phi}}_i\mathbf{S}_2{}^T\mathbf{I}_2\mathbf{I}_3{}^T\bar{\boldsymbol{\phi}}_j\mathbf{S}_4{}^T\mathbf{I}_4]+E[\mathbf{I}_1{}^T\bar{\boldsymbol{\phi}}_i\mathbf{S}_2{}^T\mathbf{I}_2\mathbf{I}_3{}^T\mathbf{S}_3\bar{\boldsymbol{\phi}}_j^T\mathbf{I}_4] \\
& \quad + E[\mathbf{I}_1{}^T\mathbf{S}_1\bar{\boldsymbol{\phi}}_i^T\mathbf{I}_2\mathbf{I}_3{}^T\bar{\boldsymbol{\phi}}_j\mathbf{S}_4{}^T\mathbf{I}_4]+E[\mathbf{I}_1{}^T\mathbf{S}_1\bar{\boldsymbol{\phi}}_i^T\mathbf{I}_2\mathbf{I}_3{}^T\mathbf{S}_3\bar{\boldsymbol{\phi}}_j^T\mathbf{I}_4] \\
& \quad + E\left[\mathbf{I}_1{}^T\mathbf{S}_1\mathbf{S}_2{}^T\mathbf{I}_2\mathbf{I}_3{}^T\mathbf{S}_3\mathbf{S}_4{}^T\mathbf{I}_4\right] \qquad (4.86)
\end{aligned}
$$

Each term above can be though of as the product of four scalars $\mathbf{I}_n{}^T\mathbf{S}_n$, so those four scalars can be rearranged within each term. The fourth, fifth, sixth, and seventh terms in (4.86) can be simplified as follows:

$$
\begin{aligned}
E[\mathbf{I}_1{}^T\bar{\boldsymbol{\phi}}_i\mathbf{S}_2{}^T\mathbf{I}_2\mathbf{I}_3{}^T\bar{\boldsymbol{\phi}}_j\mathbf{S}_4{}^T\mathbf{I}_4] &= E[\mathbf{I}_1{}^T\bar{\boldsymbol{\phi}}_i\bar{\boldsymbol{\phi}}_j^T\mathbf{I}_3\mathbf{I}_2{}^T\mathbf{S}_2\mathbf{S}_4{}^T\mathbf{I}_4] = \mathbf{I}_1{}^T\bar{\boldsymbol{\phi}}_i\bar{\boldsymbol{\phi}}_j^T\mathbf{I}_3\mathbf{I}_2{}^T\mathbf{V}_{ij}\mathbf{I}_4 \\
E[\mathbf{I}_1{}^T\bar{\boldsymbol{\phi}}_i\mathbf{S}_2{}^T\mathbf{I}_2\mathbf{I}_3{}^T\mathbf{S}_3\bar{\boldsymbol{\phi}}_j^T\mathbf{I}_4] &= E[\mathbf{I}_1{}^T\bar{\boldsymbol{\phi}}_i\bar{\boldsymbol{\phi}}_j^T\mathbf{I}_4\mathbf{I}_2{}^T\mathbf{S}_2\mathbf{S}_3{}^T\mathbf{I}_3] = \mathbf{I}_1{}^T\bar{\boldsymbol{\phi}}_i\bar{\boldsymbol{\phi}}_j^T\mathbf{I}_4\mathbf{I}_2{}^T\mathbf{V}_{ij}\mathbf{I}_3 \\
E[\mathbf{I}_1{}^T\mathbf{S}_1\bar{\boldsymbol{\phi}}_i^T\mathbf{I}_2\mathbf{I}_3{}^T\bar{\boldsymbol{\phi}}_j\mathbf{S}_4{}^T\mathbf{I}_4] &= E[\mathbf{I}_2{}^T\bar{\boldsymbol{\phi}}_i\bar{\boldsymbol{\phi}}_j^T\mathbf{I}_3\mathbf{I}_1{}^T\mathbf{S}_1\mathbf{S}_4{}^T\mathbf{I}_4] = \mathbf{I}_2{}^T\bar{\boldsymbol{\phi}}_i\bar{\boldsymbol{\phi}}_j^T\mathbf{I}_3\mathbf{I}_1{}^T\mathbf{V}_{ij}\mathbf{I}_4 \\
E[\mathbf{I}_1{}^T\mathbf{S}_1\bar{\boldsymbol{\phi}}_i^T\mathbf{I}_2\mathbf{I}_3{}^T\mathbf{S}_3\bar{\boldsymbol{\phi}}_j^T\mathbf{I}_4] &= E[\mathbf{I}_2{}^T\bar{\boldsymbol{\phi}}_i\bar{\boldsymbol{\phi}}_j^T\mathbf{I}_4\mathbf{I}_1{}^T\mathbf{S}_1\mathbf{S}_3{}^T\mathbf{I}_3] = \mathbf{I}_2{}^T\bar{\boldsymbol{\phi}}_i\bar{\boldsymbol{\phi}}_j^T\mathbf{I}_4\mathbf{I}_1{}^T\mathbf{V}_{ij}\mathbf{I}_3
\end{aligned}
\qquad (4.87)
$$

Following the scalar derivation, we consider the eighth term in (4.86) by looking at the various combinations of sums. We ignore the fourth moment contributions because they cancel out in the end.

- 12,34: $k_1 = k_2$, $k_3 = k_4$:

$$E[\mathbf{I}_1{}^T\mathbf{S}_1\mathbf{S}_2{}^T\mathbf{I}_2][\mathbf{I}_3{}^T\mathbf{S}_3\mathbf{S}_4{}^T\mathbf{I}_4] = \mathbf{I}_1{}^T\boldsymbol{\Sigma}_i\mathbf{I}_2\mathbf{I}_3{}^T\boldsymbol{\Sigma}_j\mathbf{I}_4$$

- 13,24: $k_1 = k_3 + i - j$, $k_2 = k_4 + i - j$:

$$E[\mathbf{I}_1{}^T\mathbf{S}_1\mathbf{S}_3{}^T\mathbf{I}_3][\mathbf{I}_2{}^T\mathbf{S}_2\mathbf{S}_4{}^T\mathbf{I}_4] = \mathbf{I}_1{}^T\mathbf{V}_{ij}\mathbf{I}_3\mathbf{I}_2{}^T\mathbf{V}_{ij}\mathbf{I}_4$$

- 14,23: $k_1 = k_4 + i - j$, $k_2 = k_3 + i - j$:

$$E[\mathbf{I}_1{}^T\mathbf{S}_1\mathbf{S}_4{}^T\mathbf{I}_4][\mathbf{I}_2{}^T\mathbf{S}_2\mathbf{S}_3{}^T\mathbf{I}_3] = \mathbf{I}_1{}^T\mathbf{V}_{ij}\mathbf{I}_4\mathbf{I}_2{}^T\mathbf{V}_{ij}\mathbf{I}_3$$

By combining the first three terms from (4.86) with the first equation above, we can rewrite the expected value as follows:

$$\begin{aligned} E[\cdots] &= \mathbf{I}_1{}^T(\boldsymbol{\Sigma}_i + \bar{\boldsymbol{\phi}}_i\bar{\boldsymbol{\phi}}_i^T)\mathbf{I}_2\mathbf{I}_3{}^T(\boldsymbol{\Sigma}_j + \bar{\boldsymbol{\phi}}_j\bar{\boldsymbol{\phi}}_j^T)\mathbf{I}_4 + \mathbf{I}_1{}^T\bar{\boldsymbol{\phi}}_i\bar{\boldsymbol{\phi}}_j^T\mathbf{I}_3\mathbf{I}_2{}^T\mathbf{V}_{ij}\mathbf{I}_4 \\ &\quad + \mathbf{I}_1{}^T\bar{\boldsymbol{\phi}}_i\bar{\boldsymbol{\phi}}_j^T\mathbf{I}_4\mathbf{I}_2{}^T\mathbf{V}_{ij}\mathbf{I}_3 + \mathbf{I}_2{}^T\bar{\boldsymbol{\phi}}_i\bar{\boldsymbol{\phi}}_j^T\mathbf{I}_3\mathbf{I}_1{}^T\mathbf{V}_{ij}\mathbf{I}_4 + \mathbf{I}_2{}^T\bar{\boldsymbol{\phi}}_i\bar{\boldsymbol{\phi}}_j^T\mathbf{I}_4\mathbf{I}_1{}^T\mathbf{V}_{ij}\mathbf{I}_3 \\ &\quad + \mathbf{I}_1{}^T\mathbf{V}_{ij}\mathbf{I}_3\mathbf{I}_2{}^T\mathbf{V}_{ij}\mathbf{I}_4 + \mathbf{I}_1{}^T\mathbf{V}_{ij}\mathbf{I}_4\mathbf{I}_2{}^T\mathbf{V}_{ij}\mathbf{I}_3 \end{aligned} \tag{4.88}$$

The first term in (4.88) cancels out with $E[R_a]E[R_b]$ in the summation for $\text{cov}(R_a, R_b)$ (4.85), so the covariance is:

$$\begin{aligned} \text{cov}(R_a, R_b) &= \sum_{i=1}^{N}\sum_{j=1}^{N}\big(\mathbf{I}_1{}^T\bar{\boldsymbol{\phi}}_i\bar{\boldsymbol{\phi}}_j^T\mathbf{I}_3\mathbf{I}_2{}^T\mathbf{V}_{ij}\mathbf{I}_4 + \mathbf{I}_1{}^T\bar{\boldsymbol{\phi}}_i\bar{\boldsymbol{\phi}}_j^T\mathbf{I}_4\mathbf{I}_2{}^T\mathbf{V}_{ij}\mathbf{I}_3 \\ &\qquad + \mathbf{I}_2{}^T\bar{\boldsymbol{\phi}}_i\bar{\boldsymbol{\phi}}_j^T\mathbf{I}_3\mathbf{I}_1{}^T\mathbf{V}_{ij}\mathbf{I}_4 + \mathbf{I}_2{}^T\bar{\boldsymbol{\phi}}_i\bar{\boldsymbol{\phi}}_j^T\mathbf{I}_4\mathbf{I}_1{}^T\mathbf{V}_{ij}\mathbf{I}_3 \\ &\qquad + \mathbf{I}_1{}^T\mathbf{V}_{ij}\mathbf{I}_3\mathbf{I}_2{}^T\mathbf{V}_{ij}\mathbf{I}_4 + \mathbf{I}_1{}^T\mathbf{V}_{ij}\mathbf{I}_4\mathbf{I}_2{}^T\mathbf{V}_{ij}\mathbf{I}_3\big) \end{aligned} \tag{4.89}$$

As in the scalar case, we evaluate the double sum for $i \geq j$ (with a new index $k_1 = i - j$) and for $j \geq i$ (with a new index $k_2 = j - i$) then we subtract one sum for

$i = j$. We assume that $\bar{\boldsymbol{\phi}}$ and $\boldsymbol{\Sigma}$ change slowly in time, as we did in the scalar case.

$$\begin{aligned}
\mathrm{cov}(R_a, R_b) &= \sum_{i=1}^{N}\sum_{k_1=0}^{i-1} \big(\mathbf{I}_1{}^T \bar{\boldsymbol{\phi}}_i \bar{\boldsymbol{\phi}}_i^T \mathbf{I}_3 \mathbf{I}_2{}^T \mathbf{a}^{k_1} \boldsymbol{\Sigma}_i \mathbf{I}_4 + \mathbf{I}_1{}^T \bar{\boldsymbol{\phi}}_i \bar{\boldsymbol{\phi}}_i^T \mathbf{I}_4 \mathbf{I}_2{}^T \mathbf{a}^{k_1} \boldsymbol{\Sigma}_i \mathbf{I}_3 \\
&\qquad + \mathbf{I}_2{}^T \bar{\boldsymbol{\phi}}_i \bar{\boldsymbol{\phi}}_i^T \mathbf{I}_3 \mathbf{I}_1{}^T \mathbf{a}^{k_1} \boldsymbol{\Sigma}_i \mathbf{I}_4 + \mathbf{I}_2{}^T \bar{\boldsymbol{\phi}}_i \bar{\boldsymbol{\phi}}_i^T \mathbf{I}_4 \mathbf{I}_1{}^T \mathbf{a}^{k_1} \boldsymbol{\Sigma}_i \mathbf{I}_3 \\
&\qquad + \mathbf{I}_1{}^T \mathbf{a}^{k_1} \boldsymbol{\Sigma}_i \mathbf{I}_3 \mathbf{I}_2{}^T \mathbf{a}^{k_1} \boldsymbol{\Sigma}_i \mathbf{I}_4 + \mathbf{I}_1{}^T \mathbf{a}^{k_1} \boldsymbol{\Sigma}_i \mathbf{I}_4 \mathbf{I}_2{}^T \mathbf{a}^{k_1} \boldsymbol{\Sigma}_i \mathbf{I}_3 \big) \\
&+ \sum_{j=1}^{N}\sum_{k_2=0}^{j-1} \big(\mathbf{I}_1{}^T \bar{\boldsymbol{\phi}}_j \bar{\boldsymbol{\phi}}_j^T \mathbf{I}_3 \mathbf{I}_2{}^T \boldsymbol{\Sigma}_j (\mathbf{a}^{k_2})^T \mathbf{I}_4 + \mathbf{I}_1{}^T \bar{\boldsymbol{\phi}}_j \bar{\boldsymbol{\phi}}_j^T \mathbf{I}_4 \mathbf{I}_2{}^T \boldsymbol{\Sigma}_j (\mathbf{a}^{k_2})^T \mathbf{I}_3 \\
&\qquad + \mathbf{I}_2{}^T \bar{\boldsymbol{\phi}}_j \bar{\boldsymbol{\phi}}_j^T \mathbf{I}_3 \mathbf{I}_1{}^T \boldsymbol{\Sigma}_j (\mathbf{a}^{k_2})^T \mathbf{I}_4 + \mathbf{I}_2{}^T \bar{\boldsymbol{\phi}}_j \bar{\boldsymbol{\phi}}_j^T \mathbf{I}_4 \mathbf{I}_1{}^T \boldsymbol{\Sigma}_j (\mathbf{a}^{k_2})^T \mathbf{I}_3 \\
&\qquad + \mathbf{I}_1{}^T \boldsymbol{\Sigma}_j (\mathbf{a}^{k_2})^T \mathbf{I}_3 \mathbf{I}_2{}^T \boldsymbol{\Sigma}_j (\mathbf{a}^{k_2})^T \mathbf{I}_4 + \mathbf{I}_1{}^T \boldsymbol{\Sigma}_j (\mathbf{a}^{k_2})^T \mathbf{I}_4 \mathbf{I}_2{}^T \boldsymbol{\Sigma}_j (\mathbf{a}^{k_2})^T \mathbf{I}_3 \big) \\
&- \sum_{i=1}^{N} \big(\mathbf{I}_1{}^T \bar{\boldsymbol{\phi}}_i \bar{\boldsymbol{\phi}}_i^T \mathbf{I}_3 \mathbf{I}_2{}^T \boldsymbol{\Sigma}_i \mathbf{I}_4 + \mathbf{I}_1{}^T \bar{\boldsymbol{\phi}}_i \bar{\boldsymbol{\phi}}_i^T \mathbf{I}_4 \mathbf{I}_2{}^T \boldsymbol{\Sigma}_i \mathbf{I}_3 \\
&\qquad + \mathbf{I}_2{}^T \bar{\boldsymbol{\phi}}_i \bar{\boldsymbol{\phi}}_i^T \mathbf{I}_3 \mathbf{I}_1{}^T \boldsymbol{\Sigma}_i \mathbf{I}_4 + \mathbf{I}_2{}^T \bar{\boldsymbol{\phi}}_i \bar{\boldsymbol{\phi}}_i^T \mathbf{I}_4 \mathbf{I}_1{}^T \boldsymbol{\Sigma}_i \mathbf{I}_3 \\
&\qquad + \mathbf{I}_1{}^T \boldsymbol{\Sigma}_i \mathbf{I}_3 \mathbf{I}_2{}^T \boldsymbol{\Sigma}_i \mathbf{I}_4 + \mathbf{I}_1{}^T \boldsymbol{\Sigma}_i \mathbf{I}_4 \mathbf{I}_2{}^T \boldsymbol{\Sigma}_i \mathbf{I}_3 \big)
\end{aligned} \tag{4.90}$$

Each of the first four terms from each sum multiplies $(\mathbf{a}^k \boldsymbol{\Sigma}_i + \boldsymbol{\Sigma}_i (\mathbf{a}^k)^T - \boldsymbol{\Sigma}_i)$. Using the summation rule (4.39), this quantity is written as $\boldsymbol{\Gamma}_{0,i}$:

$$\boldsymbol{\Gamma}_{0,i} \equiv \sum_{k=0}^{i-1} (\mathbf{a}^k \boldsymbol{\Sigma}_i + \boldsymbol{\Sigma}_i (\mathbf{a}^k)^T - \boldsymbol{\Sigma}_i) = (\mathbf{I} - \mathbf{a})^{-1} (\mathbf{I} - \mathbf{a}^i) \boldsymbol{\Sigma}_i + \boldsymbol{\Sigma}_i (\mathbf{I} - (\mathbf{a}^i)^T)(\mathbf{I} - \mathbf{a})^{-T} - \boldsymbol{\Sigma}_i \tag{4.91}$$

The last two terms in the second double sum of (4.90) can be arranged to have the same format as the first sum:

$$\begin{aligned}
\mathbf{I}_1{}^T \boldsymbol{\Sigma}_i (\mathbf{a}^k)^T \mathbf{I}_3 \mathbf{I}_2{}^T \boldsymbol{\Sigma}_i (\mathbf{a}^k)^T \mathbf{I}_4 &= \mathbf{I}_3{}^T \mathbf{a}^k \boldsymbol{\Sigma}_i \mathbf{I}_1 \mathbf{I}_4{}^T \mathbf{a}^k \boldsymbol{\Sigma}_i \mathbf{I}_2 \\
\mathbf{I}_1{}^T \boldsymbol{\Sigma}_i (\mathbf{a}^k)^T \mathbf{I}_4 \mathbf{I}_2{}^T \boldsymbol{\Sigma}_i (\mathbf{a}^k)^T \mathbf{I}_3 &= \mathbf{I}_4{}^T \mathbf{a}^k \boldsymbol{\Sigma}_i \mathbf{I}_1 \mathbf{I}_3{}^T \mathbf{a}^k \boldsymbol{\Sigma}_i \mathbf{I}_2
\end{aligned}$$

We can then compute four more $\boldsymbol{\Gamma}$ matrices by solving the non-symmetric version of the discrete-time Lyapunov equation, shown below as a MATLAB function. The numerical procedure for evaluating the discrete-time Lyapunov equation is described

in Section B.2.

$$\mathbf{\Gamma}_{1,i} = \sum_{k=0}^{i-1} \mathbf{a}^k \mathbf{\Sigma}_i \mathbf{I}_3 \mathbf{I}_2{}^T \mathbf{a}^k = \texttt{dlyap}(\mathbf{a},\ \mathbf{a},\ \mathbf{\Sigma}_i \mathbf{I}_3 \mathbf{I}_2{}^T - \mathbf{a}^i \mathbf{\Sigma}_i \mathbf{I}_3 \mathbf{I}_2{}^T \mathbf{a}^i) \tag{4.92}$$

$$\mathbf{\Gamma}_{2,i} = \sum_{k=0}^{i-1} \mathbf{a}^k \mathbf{\Sigma}_i \mathbf{I}_4 \mathbf{I}_2{}^T \mathbf{a}^k = \texttt{dlyap}(\mathbf{a},\ \mathbf{a},\ \mathbf{\Sigma}_i \mathbf{I}_4 \mathbf{I}_2{}^T - \mathbf{a}^i \mathbf{\Sigma}_i \mathbf{I}_4 \mathbf{I}_2{}^T \mathbf{a}^i) \tag{4.93}$$

$$\mathbf{\Gamma}_{3,i} = \sum_{k=0}^{i-1} \mathbf{a}^k \mathbf{\Sigma}_i \mathbf{I}_1 \mathbf{I}_4{}^T \mathbf{a}^k = \texttt{dlyap}(\mathbf{a},\ \mathbf{a},\ \mathbf{\Sigma}_i \mathbf{I}_1 \mathbf{I}_4{}^T - \mathbf{a}^i \mathbf{\Sigma}_i \mathbf{I}_1 \mathbf{I}_4{}^T \mathbf{a}^i) \tag{4.94}$$

$$\mathbf{\Gamma}_{4,i} = \sum_{k=0}^{i-1} \mathbf{a}^k \mathbf{\Sigma}_i \mathbf{I}_1 \mathbf{I}_3{}^T \mathbf{a}^k = \texttt{dlyap}(\mathbf{a},\ \mathbf{a},\ \mathbf{\Sigma}_i \mathbf{I}_1 \mathbf{I}_3{}^T - \mathbf{a}^i \mathbf{\Sigma}_i \mathbf{I}_1 \mathbf{I}_3{}^T \mathbf{a}^i) \tag{4.95}$$

Using the $\mathbf{\Gamma}$ matrices, the final result for the covariance is:

$$\begin{aligned} \operatorname{cov}(R_a, R_b) \;=\; & \sum_{i=1}^{N} \big(\mathbf{I}_1{}^T \bar{\boldsymbol{\phi}}_i \bar{\boldsymbol{\phi}}_i^T \mathbf{I}_3 \mathbf{I}_2{}^T \mathbf{\Gamma}_{0,i} \mathbf{I}_4 + \mathbf{I}_1{}^T \bar{\boldsymbol{\phi}}_i \bar{\boldsymbol{\phi}}_i^T \mathbf{I}_4 \mathbf{I}_2{}^T \mathbf{\Gamma}_{0,i} \mathbf{I}_3 \\ & + \mathbf{I}_2{}^T \bar{\boldsymbol{\phi}}_i \bar{\boldsymbol{\phi}}_i^T \mathbf{I}_3 \mathbf{I}_1{}^T \mathbf{\Gamma}_{0,i} \mathbf{I}_4 + \mathbf{I}_2{}^T \bar{\boldsymbol{\phi}}_i \bar{\boldsymbol{\phi}}_i^T \mathbf{I}_4 \mathbf{I}_1{}^T \mathbf{\Gamma}_{0,i} \mathbf{I}_3 \\ & + \mathbf{I}_1{}^T \mathbf{\Gamma}_{1,i} \mathbf{\Sigma}_i \mathbf{I}_4 + \mathbf{I}_1{}^T \mathbf{\Gamma}_{2,i} \mathbf{\Sigma}_i \mathbf{I}_3 + \mathbf{I}_3{}^T \mathbf{\Gamma}_{3,i} \mathbf{\Sigma}_i \mathbf{I}_2 + \mathbf{I}_4{}^T \mathbf{\Gamma}_{4,i} \mathbf{\Sigma}_i \mathbf{I}_2 \\ & - \; \mathbf{I}_1{}^T \mathbf{\Sigma}_i \mathbf{I}_3 \mathbf{I}_2{}^T \mathbf{\Sigma}_i \mathbf{I}_4 - \mathbf{I}_1{}^T \mathbf{\Sigma}_i \mathbf{I}_4 \mathbf{I}_2{}^T \mathbf{\Sigma}_i \mathbf{I}_3 \big) \end{aligned} \tag{4.96}$$

Using (4.96) we can compute all of the variances and covariances in the equation for $E[\mathbf{f}(\mathbf{r})_k]$ (4.84). The last step is to re-form $E[\mathbf{R}^{-1}]$ from $E[\mathbf{f}(\mathbf{r})]$:

$$E[\mathbf{R}^{-1}] = \begin{bmatrix} E[\mathbf{f}(\mathbf{r})_1] & E[\mathbf{f}(\mathbf{r})_2] \\ E[\mathbf{f}(\mathbf{r})_2] & E[\mathbf{f}(\mathbf{r})_3] \end{bmatrix} \tag{4.97}$$

Figure 4-6 shows the convergence of the various elements of the $E[\mathbf{R}^{-1}]$ matrix for an $m = 2$ system with a zero mean and a steady variance. The prediction using the correlated approach described above converges to the true values faster than the prediction using the uncorrelated inverse Wishart distribution, although the uncorrelated prediction eventually converges as well as $N \to \infty$. Figure 4-7 shows a similar plot for a correlated, time-varying, non-central $m = 3$ system.

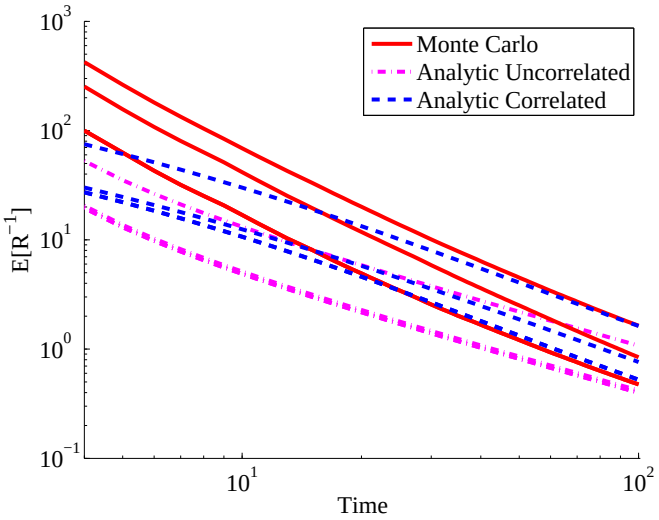

Figure 4-6: The various elements $E[\mathbf{R}^{-1}]$ for a correlated, steady, central $m = 2$ system. The red curves show the result of a 10,000-point Monte Carlo simulation. The magenta dash-dot curves show the uncorrelated prediction from the inverse Wishart distribution, and the blue dashed curves show the correlated prediction from (4.97). The correlated prediction converges to the true values faster than the uncorrelated prediction.

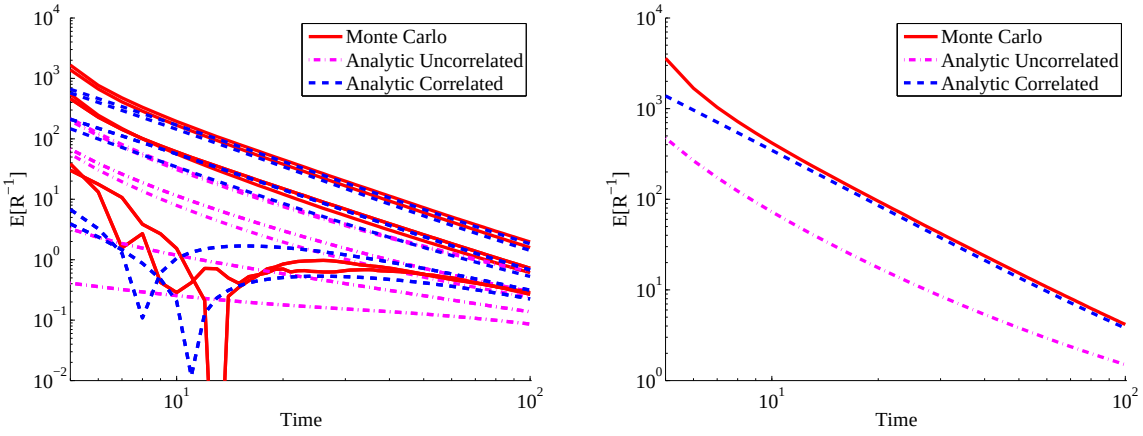

Figure 4-7: Left: The various elements $E[\mathbf{R}^{-1}]$ for a correlated, time-varying, non-central $m = 3$ system. Right: The trace of $E[\mathbf{R}^{-1}]$. The correlated prediction (blue dashed) tracks the Monte Carlo evolution (red) better than the uncorrelated prediction from the inverse Wishart distribution (magenta dash-dot), especially for the off-diagonal terms in the left plot.

4.4 Applying the Correlated Prediction of $E[\mathbf{R}^{-1}]$ to the Parameter Convergence Problem

In the previous section we derived the prediction of $E[\mathbf{R}^{-1}]$ when the learning data $\boldsymbol{\phi}$ is generated by a discrete-time state space model with process noise. In this section we transform the model for the planar holonomic vehicle to match that problem. To prevent singularities, we only consider the first five elements of $\boldsymbol{\phi}$, excluding the last element (with a value of 1) used by the wind parameters. First, we write the remaining vector of learning data in terms of the reference-local error vector $\mathbf{e_0}$.

$$\boldsymbol{\phi}_{\boldsymbol{\nu\tau}} = \begin{bmatrix} \boldsymbol{\nu} \\ \boldsymbol{\tau} \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{I} & \mathbf{0}_{3\times 3} \\ -\mathbf{K}_{\boldsymbol{\nu}} & -\mathbf{K}_{\boldsymbol{\eta 0}} \end{bmatrix}}_{\mathbf{C}_{\boldsymbol{\phi e}}} \mathbf{e_0} + \underbrace{\begin{bmatrix} \mathbf{I} \\ -\hat{\mathbf{b}}^{(inv)}\hat{\mathbf{a}} \end{bmatrix} \boldsymbol{\nu}_{\mathbf{r}}}_{\mathbf{B}_{\boldsymbol{\phi e}}} \tag{4.98}$$

We note that the mean and variance of $\boldsymbol{\phi}_{\boldsymbol{\nu\tau}}$ can be computed from the statistics of $\mathbf{e_0}$, as shown below.

$$\bar{\boldsymbol{\phi}}_{\boldsymbol{\nu\tau}} = \mathbf{C}_{\boldsymbol{\phi e}}\bar{\mathbf{e}}_{\mathbf{0}} + \mathbf{B}_{\boldsymbol{\phi e}} \tag{4.99}$$

$$\mathrm{cov}(\boldsymbol{\phi}_{\boldsymbol{\nu\tau}}) = \mathbf{C}_{\boldsymbol{\phi e}}\boldsymbol{\Sigma}_{\mathbf{0}}{\mathbf{C}_{\boldsymbol{\phi e}}}^{T} \tag{4.100}$$

By solving for $\mathbf{e_0}$ in (4.98) using the pseudoinverse, we have the following approximate result:

$$\mathbf{e_0} \approx {\mathbf{C}_{\boldsymbol{\phi e}}}^{*}(\boldsymbol{\phi}_{\boldsymbol{\nu\tau}} - \mathbf{B}_{\boldsymbol{\phi e}}). \tag{4.101}$$

Next we differentiate (4.98) with respect to time and plug in the approximate relation (4.101) to arrive at a differential equation for $\boldsymbol{\phi}_{\boldsymbol{\nu\tau}}$.

$$\begin{aligned} \dot{\boldsymbol{\phi}}_{\boldsymbol{\nu\tau}} &= \mathbf{C}_{\boldsymbol{\phi e}}\dot{\mathbf{e}}_{\mathbf{0}} \\ &= \mathbf{C}_{\boldsymbol{\phi e}}\mathbf{A_0}\mathbf{e_0} \\ &\approx \underbrace{\mathbf{C}_{\boldsymbol{\phi e}}\mathbf{A_0}{\mathbf{C}_{\boldsymbol{\phi e}}}^{*}}_{\mathbf{A}_{\boldsymbol{\phi}}}\boldsymbol{\phi}_{\boldsymbol{\nu\tau}} - \underbrace{\mathbf{C}_{\boldsymbol{\phi e}}\mathbf{A_0}{\mathbf{C}_{\boldsymbol{\phi e}}}^{*}\mathbf{B}_{\boldsymbol{\phi e}}}_{\mathbf{B}_{\boldsymbol{\phi}}} \end{aligned} \tag{4.102}$$

For the partitioned system, we need to extract the data for the free parameters in surge, sway and yaw. For the free surge parameters, we need to pre-multiply $\boldsymbol{\phi_{\nu\tau}}$ by $\mathbf{C}_{f\phi u}$ which is a version of $\mathbf{C}_{free,u}\mathbf{C}_{\phi u}$ truncated to exclude the last element of $\boldsymbol{\phi}$.

$$\begin{aligned}
\boldsymbol{\phi}_{\boldsymbol{\nu\tau} free,u} &= \mathbf{C}_{f\phi u}\boldsymbol{\phi}_{\boldsymbol{\nu\tau}} \\
\dot{\boldsymbol{\phi}}_{\boldsymbol{\nu\tau} free,u} &\approx \mathbf{C}_{f\phi u}\mathbf{A}_{\boldsymbol{\phi}}\boldsymbol{\phi}_{\boldsymbol{\nu\tau}} - \mathbf{C}_{f\phi u}\mathbf{B}_{\boldsymbol{\phi}} \\
&\approx \underbrace{\mathbf{C}_{f\phi u}\mathbf{A}_{\boldsymbol{\phi}}\mathbf{C}_{f\phi u}{}^{*}}_{\mathbf{A}_{\boldsymbol{\phi} u}}\boldsymbol{\phi}_{\boldsymbol{\nu\tau} free,u} - \underbrace{\mathbf{C}_{f\phi u}\mathbf{B}_{\boldsymbol{\phi}}}_{\mathbf{B}_{\boldsymbol{\phi} u}}
\end{aligned} \tag{4.103}$$

Equation (4.103) is the continuous-time version of (4.37), so the equivalent discrete-time filter matrix is $a = \exp(\mathbf{A}_{\boldsymbol{\phi} u}\Delta t)$. The equivalent mean data value is $\bar{\phi}_i = \mathbf{C}_{f\phi u}\,\bar{\boldsymbol{\phi}}_{\boldsymbol{\nu\tau}}(t_i)$, and the equivalent data variance is $\Sigma_i = \mathbf{C}_{f\phi u}\,\mathrm{cov}(\boldsymbol{\phi}_{\boldsymbol{\nu\tau}}(t_i))\mathbf{C}_{f\phi u}{}^{T}$. Recall that $\bar{\boldsymbol{\phi}}_{\boldsymbol{\nu\tau}}$ and $\mathrm{cov}(\boldsymbol{\phi}_{\boldsymbol{\nu\tau}})$ are defined in (4.99) and (4.100). Using these values for a, $\bar{\phi}_i$ and Σ_i, we can evaluate the correlated prediction for $E[\mathbf{R}_u{}^{-1}]$. The predictions for $E[\mathbf{R}_v{}^{-1}]$ and $E[\mathbf{R}_r{}^{-1}]$ are calculated in a similar manner using the corresponding matrices for v and r.

Computing $E[\mathbf{R}_u]$, which is required in the derivation of $E[\mathbf{R}_u{}^{-1}]$, is straightforward. First we compute $E[\mathbf{R}]$ for the full $\boldsymbol{\phi}$ vector, then we can partition it as needed for $E[\mathbf{R}_u]$, $E[\mathbf{R}_v]$ and $E[\mathbf{R}_r]$ as described below.

$$\begin{aligned}
E[\mathbf{R}] &= E\left[\boldsymbol{\Phi}\boldsymbol{\Phi}^{T}\right] \\
&= E\left[\sum_{j=1}^{N}\boldsymbol{\phi}_j\boldsymbol{\phi}_j{}^{T}\right] \\
&= \sum_{j=1}^{N}E\left[\boldsymbol{\phi}_j\boldsymbol{\phi}_j{}^{T}\right] \\
&= \sum_{j=1}^{N}\left(\mathrm{cov}(\boldsymbol{\phi}_j) + \bar{\boldsymbol{\phi}}_j\bar{\boldsymbol{\phi}}_j^{T}\right)
\end{aligned} \tag{4.104}$$

In practice the mean error $\bar{\mathbf{e}}_{\mathbf{0}}$ and error variance $\boldsymbol{\Sigma}_{\mathbf{0}}$ need not be evaluated at every time that the learning data arrives. A more appropriate sample time is t_{obs}, when those values are computed for the collision probability prediction. $E[\mathbf{R}]$ is evaluated

every t_{obs} seconds and also at the end of each maneuver using the trapezoidal rule.

If there is an *a priori* parameter estimate $\hat{\boldsymbol{\beta}}_0$ with an initial covariance matrix $\mathbf{P}_0$, then an equivalent initial information matrix can be formed for each channel. For the surge channel, $\mathbf{P}_{0,u} = \mathbf{C}_{free,u}\mathbf{C}_{\boldsymbol{\beta}u}\mathbf{P}_0{\mathbf{C}_{\boldsymbol{\beta}u}}^T{\mathbf{C}_{free,u}}^T$. The equivalent *a priori* information matrix is $W_u{\mathbf{P}_0}^{-1}$. The matrices $E[\mathbf{R}_u]$, $E[\mathbf{R}_v]$ and $E[\mathbf{R}_r]$ can be computed from the full expected information matrix and the *a priori* information.

$$\begin{aligned} E[\mathbf{R}_u] &= \mathbf{C}_{free,u}\mathbf{C}_{\boldsymbol{\phi}u}E[\mathbf{R}]{\mathbf{C}_{\boldsymbol{\phi}u}}^T{\mathbf{C}_{free,u}}^T + W_u{\mathbf{P}_{0,u}}^{-1} \\ E[\mathbf{R}_v] &= \mathbf{C}_{free,v}\mathbf{C}_{\boldsymbol{\phi}vr}E[\mathbf{R}]{\mathbf{C}_{\boldsymbol{\phi}vr}}^T{\mathbf{C}_{free,v}}^T + W_v{\mathbf{P}_{0,v}}^{-1} \\ E[\mathbf{R}_r] &= \mathbf{C}_{free,r}\mathbf{C}_{\boldsymbol{\phi}vr}E[\mathbf{R}]{\mathbf{C}_{\boldsymbol{\phi}vr}}^T{\mathbf{C}_{free,r}}^T + W_r{\mathbf{P}_{0,r}}^{-1} \end{aligned} \tag{4.105}$$

At the same time points when $E[\mathbf{R}]$ is updated, each value of $\mathrm{cov}(R_a, R_b)$ is updated based on (4.96) for use in the correlated prediction of $E[\mathbf{R}^{-1}]$ for each channel. Finally, the parameter convergence prediction is $\mathbf{P}_N = WE[\mathbf{R}^{-1}]$, applied to each channel.

An example of the parameter convergence prediction is shown in Figure 4-8. The plan `ebbbawaw` brings the vehicle around an obstacle using one intermediate waypoint. During this plan we seek to learn the values of the surge damping a_{11}, the yaw damping a_{33}, and the yaw input gain b_3. The *a priori* parameter values are equal to the true values, and the *a priori* parameter variances are 0.01, 1, and 0.0016, respectively. The learning data is collected at a rate of 1 Hz. The first part of the plan is straight, so we expect very little parameter convergence for the yaw parameters in that region. The process noise covariance is $\mathbf{W}_{\boldsymbol{\nu}} = \mathrm{diag}([2 \times 10^{-5},\ 2 \times 10^{-5},\ 3 \times 10^{-3}])$ and the sensor noise covariance is $\mathbf{W}_{\mathbf{s}} = \mathrm{diag}([0.001,\ 0.001,\ 0.1])$. The parameter convergence is plotted in Figure 4-8 using the correlated prediction, the uncorrelated inverse Wishart prediction, and the statistics from 10,000 Monte Carlo simulations. The correlated prediction matches the Monte Carlo variance better than the uncorrelated prediction, especially when the number of sample points is low. The yaw parameters do not start to converge until the vehicle begins the `a` trim.

While Algorithm 9 and the extensions in this section can be used to predict the

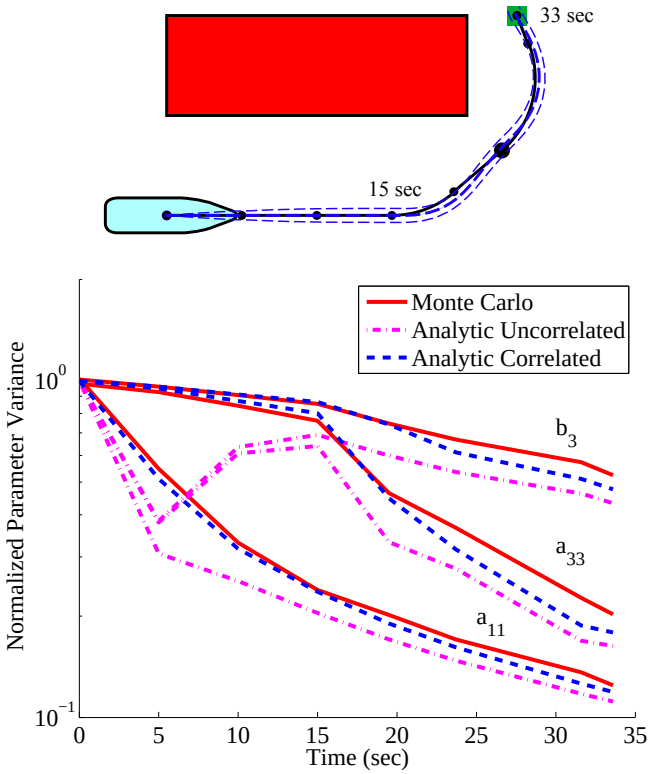

Figure 4-8: The parameter converge predictions for three model parameters while executing the motion plan shown above. The correlated prediction (blue dashed) matches the statistics from 10,000 Monte Carlo simulations (red) better than the uncorrelated inverse Wishart prediction (magenta dash-dot), especially early in the motion plan. Note that the yaw parameters do not begin to converge until the vehicle begins the turn around the obstacle.

parameter covariance evolution for any motion plan, the actual parameter uncertainty may follow a different evolution if the assumed noise models are incorrect or the true parameter values change unexpectedly. In other words, these predictions are open-loop. To add robustness to the overall approach, it is necessary to adjust the parameter covariance estimates after the learning data has been collected during the plan execution. This posterior adjustment is discussed in the next section.

4.5 Posterior Parameter Covariance Estimates

The parameter convergence predictions developed in Section 4.4 are used during the planning process to evaluate the collision probability associated with model uncertainty (discussed further in Chapter 5). The Monte Carlo results in Figure 4-8 show the variance of the various learned parameters that would be observed across many executions of the same motion plan if all of the assumptions about the sensor noise, process noise and *a priori* data were correct.

However, it is not advisable to use the final parameter convergence prediction $\mathbf{P}_N$ as the *a priori* parameter covariance for the next mission. The true posterior parameter covariance may be different from the prediction if the process noise variance or the sensor noise variance is different than expected. Model structure errors can also add bias to the posterior parameter covariance.

The more significant issue is when the true parameter value changes before or during a mission. In that case the expected parameter covariance may be small, causing significant regularization of the data, while the true parameter is far from the *a priori* value. When the true parameter changes we would like the *a priori* parameter variance $\mathbf{P}_0$ to be increased to reflect a corresponding lack of confidence in the *a priori* parameter vector $\hat{\boldsymbol{\beta}}_0$.

Fortunately, parameter error contributes to an easily measurable quantity, the *prediction error*. The true and expected prediction error are derived below.

4.5.1 Prediction Error

Consider the following plant.

$$\mathbf{Y} = \boldsymbol{\beta}^T \boldsymbol{\Phi} + \mathbf{w} \tag{4.106}$$

Without regularization, the parameter estimate after N samples is:

$$\begin{aligned}
\hat{\boldsymbol{\beta}}_N &= \mathbf{R}_1{}^{-1} \boldsymbol{\Phi} \mathbf{Y}^T \qquad \text{(for implementation)} && (4.107)\\
&= \mathbf{R}_1{}^{-1} \boldsymbol{\Phi} (\boldsymbol{\Phi}^T \boldsymbol{\beta} + \mathbf{w}^T) \\
&= \boldsymbol{\beta} + \mathbf{R}_1{}^{-1} \boldsymbol{\Phi} \mathbf{w}^T \qquad \text{(for analysis)} && (4.108)
\end{aligned}$$

where $\mathbf{R}_1 = \boldsymbol{\Phi}\boldsymbol{\Phi}^{\boldsymbol{T}}$. We may have an initial estimate of the noise variance $\hat{\sigma}_w^2$, parameter vector $\hat{\boldsymbol{\beta}}_0$, and parameter variance $\hat{\mathbf{P}}_{0,prior}$. Using this (possibly incorrect) *a priori* data, the equivalent initial information matrix is $\mathbf{R}_0 = \hat{\sigma}_w^2 \hat{\mathbf{P}}_{0,prior}{}^{-1}$ and the parameter estimate is:

$$\begin{aligned}
\hat{\boldsymbol{\beta}}_N &= \hat{\boldsymbol{\beta}}_0 + \mathbf{R}_0{}^{-1} \left(\mathbf{I} - \mathbf{R}_1 (\mathbf{R}_0 + \mathbf{R}_1)^{-1} \right) \boldsymbol{\Phi} \left(\mathbf{Y} - \hat{\boldsymbol{\beta}}_0^T \boldsymbol{\Phi} \right)^T \\
&= \hat{\boldsymbol{\beta}}_0 + \mathbf{R}_0{}^{-1} \left(\mathbf{I} - \mathbf{R}_1 (\mathbf{R}_0 + \mathbf{R}_1)^{-1} \right) \left(\boldsymbol{\Phi} \mathbf{Y}^T - \mathbf{R}_1 \hat{\boldsymbol{\beta}}_0 \right) \\
&= (\mathbf{I} - \mathbf{G}) \hat{\boldsymbol{\beta}}_0 + \mathbf{G} \mathbf{R}_1{}^{-1} \boldsymbol{\Phi} \mathbf{Y}^T \qquad \text{(for implementation)} && (4.109)\\
&= (\mathbf{I} - \mathbf{G}) \hat{\boldsymbol{\beta}}_0 + \mathbf{G} \mathbf{R}_1{}^{-1} \boldsymbol{\Phi} (\boldsymbol{\Phi}^T \boldsymbol{\beta} + \mathbf{w}^T) \\
&= (\mathbf{I} - \mathbf{G}) \hat{\boldsymbol{\beta}}_0 + \mathbf{G} \boldsymbol{\beta} + \mathbf{G} \mathbf{R}_1{}^{-1} \boldsymbol{\Phi} \mathbf{w}^T \qquad \text{(for analysis)} && (4.110)
\end{aligned}$$

where $\mathbf{G} = \mathbf{R}_0{}^{-1} \left(\mathbf{I} - \mathbf{R}_1 (\mathbf{R}_0 + \mathbf{R}_1)^{-1} \right) \mathbf{R}_1$. It can be seen that $\mathbf{G} \to \mathbf{0}$ as $\mathbf{R}_1 \to \mathbf{0}$ and $\mathbf{G} \to \mathbf{I}$ as $\mathbf{R}_1 \gg \mathbf{R}_0$, which is the case of no regularization. Next we can define the initial and final parameter error.

$$\begin{aligned}
\tilde{\boldsymbol{\beta}}_0 &= \hat{\boldsymbol{\beta}}_0 - \boldsymbol{\beta} && (4.111)\\
\tilde{\boldsymbol{\beta}}_N &= \hat{\boldsymbol{\beta}}_N - \boldsymbol{\beta} \\
&= (\mathbf{I} - \mathbf{G}) \tilde{\boldsymbol{\beta}}_0 + \mathbf{G} \mathbf{R}_1{}^{-1} \boldsymbol{\Phi} \mathbf{w}^T && (4.112)
\end{aligned}$$

The expected value of the parameter error is zero when no regularization is used

($\mathbf{G} = \mathbf{I}$) and it is equal to the initial parameter error when no data is collected ($\mathbf{G} = \mathbf{0}$).

The prediction error is the vector of errors between the measured outputs and the outputs predicted by the final parameter estimate $\hat{\boldsymbol{\beta}}_N$.

$$\begin{aligned} \tilde{\mathbf{Y}} &= \hat{\boldsymbol{\beta}}_N^T \boldsymbol{\Phi} - \mathbf{Y} \quad \text{(for implementation)} && (4.113) \\ &= \hat{\boldsymbol{\beta}}_N^T \boldsymbol{\Phi} - \boldsymbol{\beta}^T \boldsymbol{\Phi} - \mathbf{w} \\ &= \tilde{\boldsymbol{\beta}}_N^T \boldsymbol{\Phi} - \mathbf{w} \quad \text{(for analysis)} && (4.114) \end{aligned}$$

For a particular value of parameter error $\tilde{\boldsymbol{\beta}}_N$ and a particular sequence of input data $\boldsymbol{\Phi}$ (essentially, for a particular motion plan) the expected value of the prediction error is $E[\tilde{\mathbf{Y}}] = \tilde{\boldsymbol{\beta}}_N^T \boldsymbol{\Phi}$. The variance of the prediction error, $\hat{\sigma}_y^2$ is derived below.

$$\hat{\sigma}_y^2\big|_{\tilde{\boldsymbol{\beta}}_N} = \frac{1}{N} E\left[\left(\tilde{\mathbf{Y}} - E[\tilde{\mathbf{Y}}]\right)\left(\tilde{\mathbf{Y}} - E[\tilde{\mathbf{Y}}]\right)^T\right] = \frac{1}{N} E\left[\mathbf{w}\mathbf{w}^T\right] = \mathbf{W} = {\sigma_w}^2 \quad (4.115)$$

Accepting a slight abuse of terminology, the mean squared prediction error is $\frac{1}{N}\tilde{\mathbf{Y}}\tilde{\mathbf{Y}}^T$. The expected value of the mean squared prediction error is derived below. We use the properties that $\mathrm{Tr}(a) = a$ when a is a scalar and $\mathrm{Tr}(AB) = \mathrm{Tr}(BA)$.

$$\begin{aligned} \frac{1}{N} E\left[\tilde{\mathbf{Y}}\tilde{\mathbf{Y}}^T\right] &= \frac{1}{N} E\left[\tilde{\boldsymbol{\beta}}_N^T \boldsymbol{\Phi}\boldsymbol{\Phi}^T \tilde{\boldsymbol{\beta}}_N + \tilde{\boldsymbol{\beta}}_N^T \boldsymbol{\Phi}\mathbf{w}^T + \mathbf{w}\boldsymbol{\Phi}^T \tilde{\boldsymbol{\beta}}_N + \mathbf{w}\mathbf{w}^T\right] \\ &= \frac{1}{N} \mathrm{Tr}\left(\boldsymbol{\Phi}^T E\left[\tilde{\boldsymbol{\beta}}_N \tilde{\boldsymbol{\beta}}_N^T\right] \boldsymbol{\Phi}\right) + \mathbf{W} \quad (4.116) \end{aligned}$$

4.5.2 Posterior Parameter Covariance Adjustment

We define the posterior parameter covariance as $\mathbf{P}_{post} = E\left[\tilde{\boldsymbol{\beta}}_N \tilde{\boldsymbol{\beta}}_N^T\right]$. Because this quantity is inside a trace function in (4.116), we cannot solve for it exactly; however, we can solve for a scaling factor between this quantity and the final parameter convergence prediction $\mathbf{P}_N$, which was derived in Section 4.4. We define this scaling factor as α:

$$\mathbf{P}_{post} = \alpha \mathbf{P}_N \quad (4.117)$$

The next step is to plug (4.117) into (4.116) to solve for α using the measured mean squared prediction error $\frac{1}{N}\tilde{\mathbf{Y}}\tilde{\mathbf{Y}}^T$.

$$\begin{aligned} E\left[\tilde{\mathbf{Y}}\tilde{\mathbf{Y}}^T\right] &= \alpha \mathrm{Tr}\left(\mathbf{\Phi}^T\mathbf{P}_N\mathbf{\Phi}]\right) + N\mathbf{W} \\ \hat{\alpha} &= \frac{\tilde{\mathbf{Y}}\tilde{\mathbf{Y}}^T - N\mathbf{W}}{\mathrm{Tr}\left(\mathbf{\Phi}^T\mathbf{P}_N\mathbf{\Phi}]\right)} \end{aligned} \tag{4.118}$$

The scaling factor $\hat{\alpha}$ must be lower-bounded by zero to prevent the posterior parameter covariance from becoming negative. However, to prevent overconfidence due to a particularly noise-free dataset, we instead lower-bound $\hat{\alpha}$ by 1 so that $\mathbf{P}_{post}$ will never be smaller than $\mathbf{P}_N$.

Assuming the sensor and process noise levels are well known, there are two reasons for $\hat{\alpha}$ to be different from unity: initial overconfidence (the *a priori* parameter covariance is too small) or changing true parameter values (the value of $\boldsymbol{\beta}$ learned in the previous mission has since changed). To check the sensitivity of $\hat{\alpha}$ to these effects, a set of Monte Carlo simulations were run. With $m = 2$, $N = 10$, $\mathbf{\Phi}$ as a random (but fixed) matrix, and $\boldsymbol{\beta}$ as a random (but fixed) vector, the least squares learning algorithm was run 1,000 times and the mean value of $\hat{\alpha}$ was computed. The initial parameter guess $\hat{\boldsymbol{\beta}}_0$ was sampled from an initial distribution centered around $\hat{\boldsymbol{\beta}}_0$ with a covariance matrix $\mathbf{P}_0$. The *a priori* parameter covariance $E[\mathbf{P}_0]$ was scaled by a factor of 0.1, 1, and 10, and the *a priori* parameter vector $\hat{\boldsymbol{\beta}}_0$ was scaled by a factor of 0.1, 1, and 10 with respect to the true parameter value. The mean value of $\hat{\alpha}$ across the 1,000 trials is shown in Table 4.1.

Table 4.1: The effect of changing the *a priori* parameter values (rows) and *a priori* parameter covariance (columns) on the posterior parameter covariance factor $\hat{\alpha}$.

		$\mathbf{P}_0$ scaling		
		0.1	1	10
$\hat{\boldsymbol{\beta}}_0$ scaling	0.1	24.946	1.578	1.172
	1	5.362	1.440	1.183
	10	2178.6	57.556	1.533

It can be seen in the table that $\hat{\alpha}$ increases the posterior parameter covariance over

the prediction when the initial confidence is too high (left column) but it preserves the predicted covariance when the confidence is appropriate or too low (middle and right columns). Similarly it increases when the true parameter value changes (top and bottom rows). The smallest element in the table corresponds to a changed parameter value but a confidence that is too low, so in that case the predicted value of $\mathbf{P}_N$ is appropriate.

This posterior adjustment can handle incorrect estimates of the noise levels, as well. If there is more noise in the system than expected, then $\hat{\alpha}$ will increase the final parameter variance to account for the added parameter uncertainty.

4.6 Summary

In this chapter we have developed a least squares learning algorithm for the planar holonomic vehicle's parameter values. The learning algorithm is designed to be run in several batches, with each batch corresponding to a maneuver in the motion plan. In this way, the parameter estimates are updated at the end of each maneuver.

The parameter covariance predictions derived in this chapter (Algorithm 9 and Equation 4.34) are used in the next chapter to evaluate the collision risk associated with parameter uncertainty for each motion plan. The collision risk will be lower after the parameter values have converged to the true values, so being able to predict the parameter convergence ahead of time is very important. The posterior parameter covariance corrections add feedback to the learning process on a larger mission-to-mission scale.

The planning process from Chapter 3 and the learning algorithm from this chapter are brought together in Chapter 5 to create a complete integrated motion planning and model learning algorithm for autonomous mobile robots.

Chapter 5

Integrated Motion Planning and Model Learning

Throughout this thesis we have developed the ingredients necessary for an integrated motion planning and model learning algorithm. In Chapter 3 we developed a robust cost-based motion planner that uses a prediction of the collision probability to find plans that are safe and efficient (Algorithm 8). In Chapter 4 we described a model learning algorithm that is used to generate parameter estimates while the vehicle executes a motion plan. Additionally, we derived predictions of the parameter covariance throughout a motion plan, which can be used to compare the uncertainty evolution across different plans (Algorithm 9 with Equation 4.34). In this chapter we map the parameter covariance evolution back into the collision probability prediction so that the effect that different motion plans have on the learning rate can be seen in the planner's cost function. The result of this link is that the planning and learning problems are unified in a single framework, and the robust motion planner explores the state space, as necessary, to find optimal motion plans.

This chapter begins with a high-level view of the integrated motion planning and model learning algorithm. Then we present a simple example using a 2D grid-based mobile robot simulation to see the effects of the integrated strategy. For the planar holonomic vehicle, the mapping from parameter uncertainty to collision probability is performed using Hermite quadrature, which is described in Section 5.3. The full

algorithm is listed in Section 5.4 along with notes about implementation details. Finally, simulations and experimental results are presented in Section 5.5.

5.1 Overview

This section provides an overview of the integrated motion planning and model learning algorithm. First we recall the exploration vs. exploitation problem discussed in Chapter 2, then we describe how the motion planner and parameter convergence predictions are combined in the same framework.

5.1.1 Exploration vs. Exploitation

If a mobile robot performs online learning while executing a motion plan, then the input data used by the learning algorithm is a function of the motion plan. For example, a motion plan that drives the vehicle in a straight line generates much less turning-related data than a motion plan that drives the vehicle in a circle; this can be seen in Figure 4-8. Overall, a learning algorithm has a faster convergence rate when the input data provided to it has a high signal-to-noise ratio. When learning model parameters, the signals are state variables. In this context, large signals correspond to highly excited states. For example, the surge thrust gain is best learned when the surge control input is large. For fast overall parameter convergence the best policy excites all of the states used by the learning algorithm, essentially exploring the state space for novel data. However, this policy will likely delay the vehicle from its mission.

Meanwhile, executing the time-optimal plan can be considered exploitation of the existing model parameters. However, the vehicle's path-following performance depends on the quality of its controller, and large path-following errors may lead to collisions if there are obstacles near the reference path. The optimal controller is designed based on the model parameters of the vehicle; as the parameter estimates improve, so does the path-following performance. This is especially true for the feedforward part of the controller in which the estimated vehicle dynamics are canceled out in the open loop. Thus pure exploitation has a cost as well.

5.1.2 Integrated Strategy

To maximize the overall mission performance, we must find a balance between exploring the state space and exploiting the existing model information to reach the goal. It is possible to find this balance within the motion planning framework described in Chapter 3. Merging learning and planning requires two key steps:

1. *Predict the model uncertainty evolution for each candidate plan* – Using the predictions from Chapter 4, estimate the parameter covariance at each stage of the motion plan.

2. *Predict the tangible cost associated with each candidate plan, given the predicted model uncertainty evolution* – The tangible cost is the combined cost used by the motion planning algorithm, which includes both the plan duration and the collision probability (3.148).

The first step was studied in Chapter 4; for a given motion plan, noise model, and *a priori* data, we can predict how much the parameter values will have converged due to the least squares learning algorithm. Those predictions are used in the second step to estimate the collision probability due to the parameter variance. The cost function used by the motion planner includes this collision probability estimate, so the planner incorporates the learning estimates into the planning process. The key insight is that the optimal plan within this framework will *automatically incorporate active learning strategies* as necessary.

Figure 5-1 shows how the parameter convergence predictions are woven into the planning framework. Note that parameter convergence does not appear explicitly in the cost function. The robot is not intrinsically motivated, as in [46]; the planner does not seek to explicitly minimize the parameter variance, but rather minimize the detrimental effects that a high parameter variance would have on the collision probability.

The following section shows how this integrated planning and learning strategy works in a very simple, illustrative example.

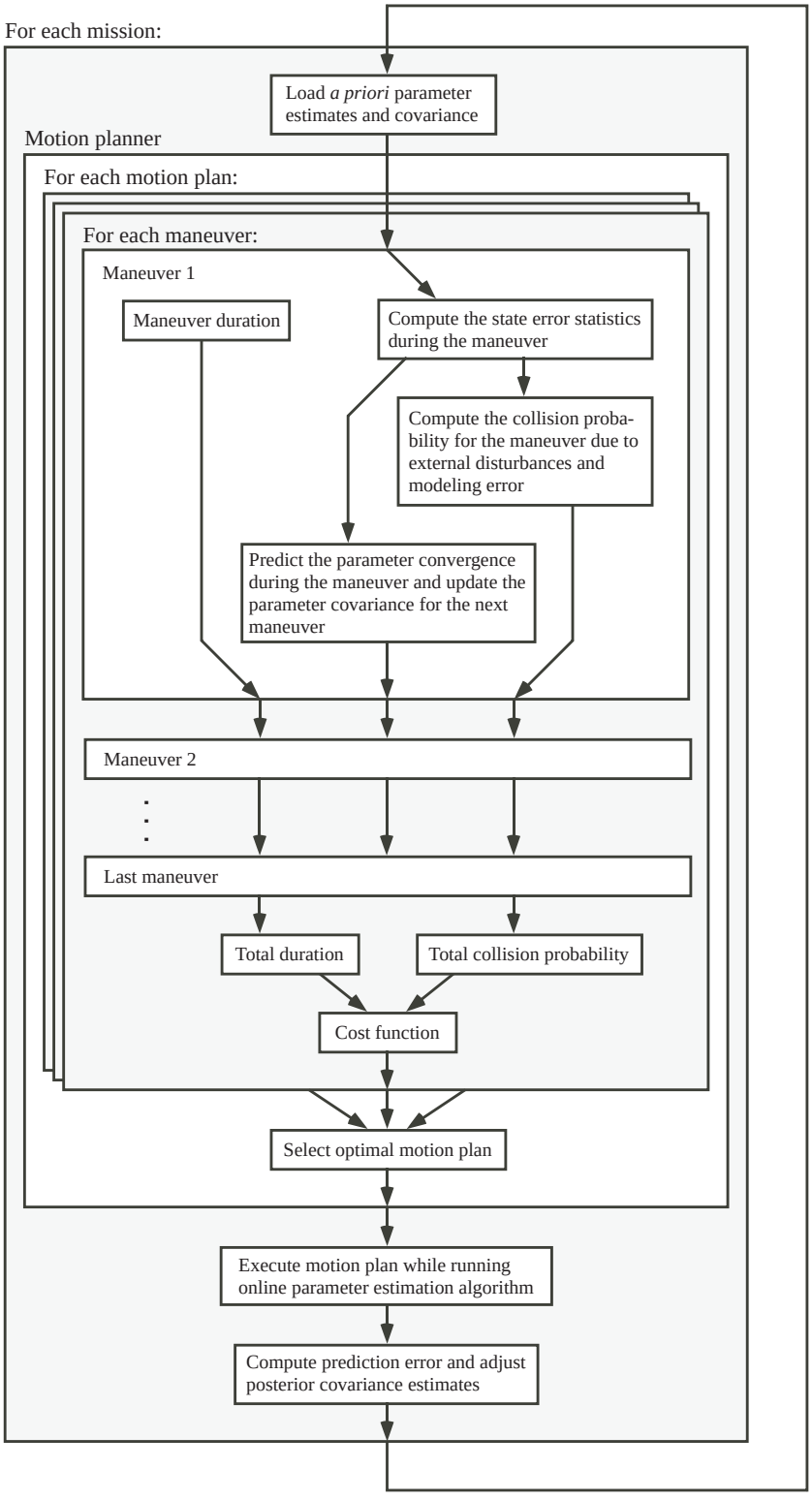

Figure 5-1: The integrated motion planning and model learning framework. Parameter convergence does appear explicitly in the planner's cost function, but rather implicitly through the predicted collision probability for each candidate plan.

5.2 Simple 2D Example

In this section we apply the integrated motion planning and model learning algorithm described above to a simple mobile robot model to show how the planner uses active learning to reduce the predicted cost of motion plans. Consider a mobile robot moving on a 2D Cartesian grid from a start point to a goal point. Some grid points are occupied by obstacles. The robot has a perfect map of the environment and perfect knowledge of its position on the map. The A* search algorithm is used to find a collision-free path connecting adjacent grid points (no diagonals) between the start point and the goal point. The robot moves between grid points using an impulse whose strength must be learned by the robot. One model parameter is the appropriate impulse strength in the X direction (East/West), and the other model parameter is the appropriate impulse strength in the Y direction (North/South). Least squares regression is used to learn the two model parameters during the execution of the motion plan. There are no external disturbances, so path-following performance is solely a function of the model uncertainty. Furthermore, we assume that the X and Y motions of the robot are uncoupled. The assumptions used in this example are very restrictive, but they reduce the mathematical complexity of the parameter convergence and collision probability predictions.

5.2.1 Plant Description and Control Law

The robot state is its position vector $\mathbf{x}_k = [X_k, Y_k]^T$ at the time index t_k. The control input vector is $\boldsymbol{\tau}_k = [\tau_{x,k}, \tau_{y,k}]^T$. The amount of motion resulting from the control input is a function of the scalar gains b_x and b_y, which form the diagonal of a 2×2 matrix $\mathbf{B}$. The system model is shown below.

$$\begin{aligned} \begin{bmatrix} X_{k+1} \\ Y_{k+1} \end{bmatrix} &= \begin{bmatrix} X_k \\ Y_k \end{bmatrix} + \begin{bmatrix} b_x & 0 \\ 0 & b_y \end{bmatrix} \begin{bmatrix} \tau_{x,k} \\ \tau_{y,k} \end{bmatrix} \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \mathbf{B}\boldsymbol{\tau}_k \end{aligned} \tag{5.1}$$

A motion plan defines the reference position at each time index, $\mathbf{r}_k = [r_{x,k}, r_{y,k}]^T$.

The reference positions are integers, but the actual robot positions are real numbers. The position error is $\mathbf{e}_k = \mathbf{x}_k - \mathbf{r}_k$. The motion plan can be specified by the cardinal directions North (`N`), South (`S`), East (`E`) and West (`W`). An example motion plan `EEENNNWSWN` is shown in Figure 5-2.

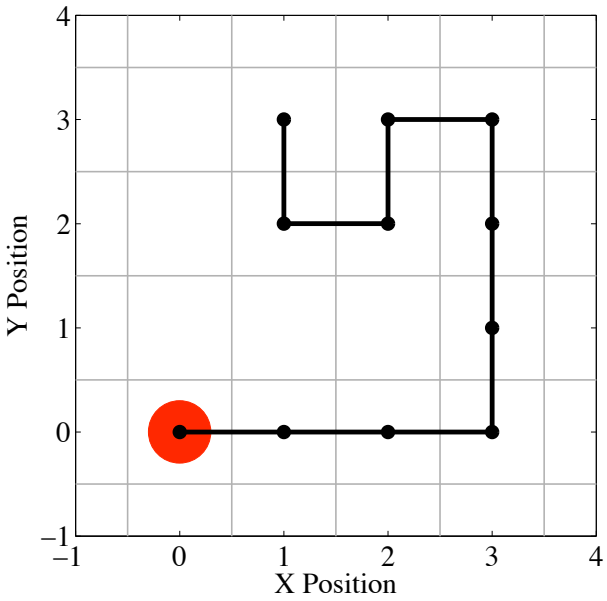

Figure 5-2: The mobile robot (indicated by a red circle) in the grid world and the motion plan `EEENNNWSWN`.

Equation (5.1) can be solved exactly to find the control action $\boldsymbol{\tau}_k$ that moves the robot from the current state $\mathbf{x}_k$ to the next reference state $\mathbf{r}_{k+1}$.

$$\boldsymbol{\tau}_k = \mathbf{B}^{-1}(\mathbf{r}_{k+1} - \mathbf{x}_k) \tag{5.2}$$

However, if the control gains are not known exactly, then the best estimate $\hat{\mathbf{B}} = \text{diag}([\hat{b}_x, \hat{b}_y])$ must be used instead of $\mathbf{B}$. The resulting control law can be written either in terms of the current state $\mathbf{x}_k$ or the position error $\mathbf{e}_k$.

$$\begin{aligned} \boldsymbol{\tau}_k &= \hat{\mathbf{B}}^{-1}(\mathbf{r}_{k+1} - \mathbf{x}_k) \\ &= \hat{\mathbf{B}}^{-1}(\mathbf{r}_{k+1} - (\mathbf{r}_k + \mathbf{e}_k)) \\ &= \hat{\mathbf{B}}^{-1}(\mathbf{r}_{k+1} - \mathbf{r}_k) - \hat{\mathbf{B}}^{-1}\mathbf{e}_k \end{aligned} \tag{5.3}$$

Equation (5.3) shows the separation of the feedforward term, which is a function of the change in the reference position, and the feedback term, which is a function of the current position error. The evolution of the error is derived below.

$$\begin{aligned}
\mathbf{e}_{k+1} &= \mathbf{x}_{k+1} - \mathbf{r}_{k+1} \\
&= \mathbf{x}_k + \mathbf{B}\hat{\mathbf{B}}^{-1}(\mathbf{r}_{k+1} - \mathbf{x}_k) - \mathbf{r}_{k+1} \\
&= (\mathbf{I} - \mathbf{B}\hat{\mathbf{B}}^{-1})(\mathbf{x}_k - \mathbf{r}_{k+1}) \\
&= (\mathbf{I} - \mathbf{B}\hat{\mathbf{B}}^{-1})(\mathbf{e}_k - (\mathbf{r}_{k+1} - \mathbf{r}_k))
\end{aligned} \tag{5.4}$$

5.2.2 Least Squares Learning Algorithm

Least squares regression is used to learn the parameter values b_x and b_y. The estimates used in the control law (5.3) at t_k are $\hat{b}_{x,k}$ and $\hat{b}_{y,k}$. At each timestep the input data to the learning algorithm is $\boldsymbol{\tau}_k$ and the output is $\mathbf{y}_k \equiv \mathbf{x}_k - \mathbf{x}_{k-1}$. This vector is measured with additive zero-mean uncorrelated Gaussian noise $y_{x,k}$ and $y_{y,k}$; the variance of each noise sample is W. Because the X and Y channels are uncoupled, we can perform the learning for each channel separately. For notational convenience the x and y subscripts will be removed in this analysis. The linear system for each channel is:

$$y_k = b\,\tau_{k-1} + w_k \quad \text{for} \quad k > 0 \tag{5.5}$$

N samples are collected during the learning process. The inputs τ_k are stored in a $1 \times N$ vector $\boldsymbol{\Phi}$, and the outputs y_k are stored in the $1 \times N$ vector $\mathbf{Y}$. The noise values are stored in the $1 \times N$ vector $\mathbf{w}$.

$$\begin{aligned}
\boldsymbol{\Phi} &= [\,\tau_0 \ \ldots \ \tau_{N-1}\,] \\
\mathbf{Y} &= [\,y_1 \ \ldots \ y_N\,] \\
\mathbf{w} &= [\,w_1 \ \ldots \ w_N\,]
\end{aligned} \tag{5.6}$$

Using the notation in (5.6), the system can be written as follows:

$$\mathbf{Y} = b\,\mathbf{\Phi} + \mathbf{w}. \tag{5.7}$$

The least squares estimate for b is:

$$\hat{b}_N = \mathbf{R}^{-1}\mathbf{\Phi}\mathbf{Y}^T \quad \text{where} \quad R = \mathbf{\Phi}\mathbf{\Phi}^T. \tag{5.8}$$

Following the analysis in Chapter 4, we know that $E[\hat{b}_N] = 0$ and $E[P] = WE[R^{-1}]$ where $E[P]$ is the expected parameter variance. If we have an *a priori* estimate of the parameter value $\hat{b}_0$ with a covariance P_0, then the equivalent *a priori* information is $R_0 = W/P_0$ and the regularized parameter estimate is:

$$\hat{b}_N = \hat{b}_0 + {R_0}^{-1}(1 - R_1(R_0 + R_1)^{-1})\mathbf{\Phi}(\mathbf{Z} - \hat{\mathbf{Z}})^T \tag{5.9}$$

where $\hat{\mathbf{Z}} = \hat{b}_0\mathbf{\Phi}$.

5.2.3 Predicting the Parameter Convergence

To simplify the analysis of the parameter convergence, we use the zeroth-order approximation $E[P] \approx WE[R]^{-1}$. The expected information $E[R]$ is computed as follows:

$$\begin{aligned} E[R] &= E\left[\sum_{k=0}^{N-1} {\tau_k}^2 + R_0\right] \\ &= \sum_{k=0}^{N-1} E[{\tau_k}^2] + R_0 \\ &= \sum_{k=1}^{N-1} \left(\bar{\tau}_k^2 + \mathrm{var}(\tau_k)\right) + R_0 \end{aligned} \tag{5.10}$$

where $\bar{\tau}_k = (r_{k+1} - r_k)/b$ and $\text{var}(\tau_k) = E[(\tau_k - \bar{\tau}_k)^2]$ is computed below.

$$\begin{aligned}
\tau_k &= \frac{1}{\hat{b}_k}(r_{k+1} - x_k) \\
&\approx \left(\frac{1}{b} - \frac{\hat{b}_k - b}{b^2}\right)(r_{k+1} - r_k) \\
\tau_k - \bar{\tau}_k &= -\frac{\hat{b}_k - b}{b^2}(r_{k+1} - r_k) \\
E[(\tau_k - \bar{\tau}_k)^2] &= \frac{1}{b^2}(r_{k+1} - r_k)^2 E[P_k] \qquad (5.11)
\end{aligned}$$

Combining the effect of the mean and the variance, we have:

$$E[{\tau_k}^2] = \bar{\tau}_k^2(1 + E[P_k]). \qquad (5.12)$$

The expected parameter variance at each step is computed using the following recursive relation, starting from the *a priori* parameter variance P_0.

$$E[P_N] = W\left(\sum_{k=0}^{N-1} \bar{\tau}_k^2(1 + E[P_k]) + \frac{W}{P_0}\right)^{-1} \qquad (5.13)$$

Because $\bar{\tau}_k$ is only a function of the reference path, the parameter variance evolution can be computed for each motion plan considered by the planner. The parameter variance prediction depends on the true parameter value b which is not known perfectly during the planning process. The best approximation we can use before collecting any data is the *a priori* parameter value $\hat{b}_0$.

The diagonal matrix $\mathbf{P}_k$ is formed by the two expected parameter variances $E[P_{x,k}]$ and $E[P_{y,k}]$. The parameter variance evolution was evaluated for the motion plan shown in Figure 5-2 with $\mathbf{P}_0 = 0.1 \times \mathbf{I}$ and $W = 0.01$. In this example the true parameter values are $b_x = 1$ and $b_y = -1$. The predicted parameter variance for each channel is plotted in Figure 5-3. A 10,000-point Monte Carlo simulation was run using *a priori* parameter estimates sampled from a normal distribution centered around the true parameter values with a variance $\mathbf{P}_0$: $\hat{\mathbf{B}}_0 \sim N(\mathbf{B}, \mathbf{P}_0)$. The variance of the resulting batch of parameter estimates at each time point is plotted in Figure

5-3. Note that the $\hat{b}_y$ variance does not reduce until the reference position changes in the Y direction. When the reference does not change, there is no added information, so no learning takes place. Similarly, when the X reference stops changing, the $\hat{b}_x$ variance stops diminishing.

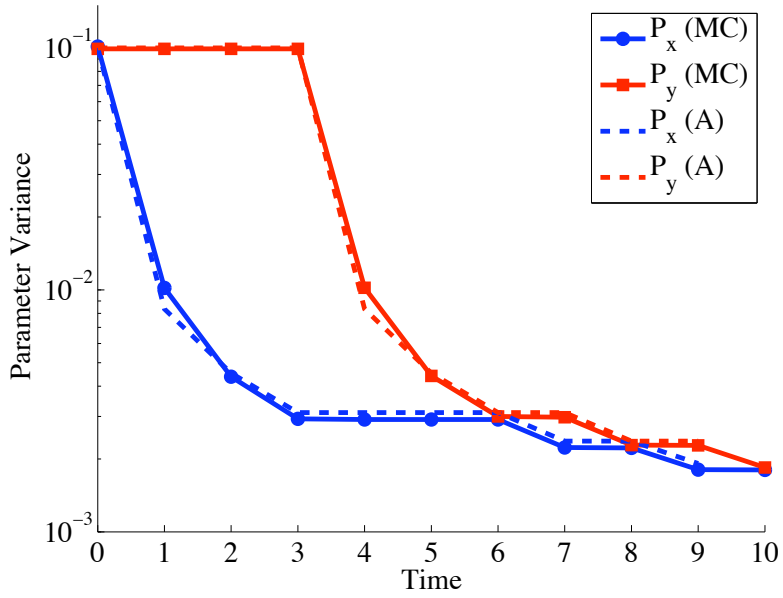

Figure 5-3: Predicted parameter variance evolution (dashed) and the result of a 10,000-point Monte Carlo simulation (solid) for the motion plan shown in Figure 5-2.

5.2.4 Predicting the Collision Probability

The probability of the robot hitting an obstacle is a function of the path-following error variance, which is in turn a function of the parameter variance. Using the same approximation as in (5.11), the error is:

$$\begin{aligned} e_k &= \left(1 - \frac{b}{\hat{b}_{k-1}}\right)(e_{k-1} - (r_k - r_{k-1})) \\ &\approx \frac{\hat{b}_{k-1} - b}{b}(e_{k-1} - (r_k - r_{k-1})) \end{aligned} \tag{5.14}$$

The mean error is 0, so the error variance is $V_k = E[{e_k}^2]$. The recursive definition

of V_k is derived below.

$$\begin{aligned} E[{e_k}^2] &= E\left[\frac{(\hat{b}_{k-1}-b)^2}{b^2}\left({e_{k-1}}^2 - 2e_{k-1}(r_k - r_{k-1}) + (r_k - r_{k-1})^2\right)\right] \\ V_k &= \frac{E[P_{k-1}]}{b^2}V_{k-1} + E[P_{k-1}]\bar{\tau}_{k-1}{}^2 \end{aligned} \tag{5.15}$$

Using the error variance prediction we compute the probability that the robot will collide with each obstacle at each point in time. Due to the zero-mean Gaussian noise assumption, the robot's position is normally distributed around the reference position at each time index t_k. The position variance for each channel $V_{x,k}$ and $V_{y,k}$ is computed using (5.15). The distance between the edge of the robot at t_k and the nearest edge of the j'th obstacle is d_{kj} and the angle of the shortest connecting line is θ_{kj}. The error variance in the θ_{kj} direction is:

$$V_{kj} = \cos^2(\theta_{kj})V_{x,k} + \sin^2(\theta_{kj})V_{y,k} \tag{5.16}$$

If we define the position error along the θ_{kj} axis as x_θ, then the probability density function for x_θ is:

$$p(x_\theta) = \frac{1}{\sqrt{2\pi V_{kj}}}\exp\left(-\frac{{x_\theta}^2}{2V_{kj}}\right) \tag{5.17}$$

The integral of (5.17) from the edge of the obstacle to infinity is the probability that the robot is intersecting with the obstacle:

$$P_{hit,kj} = \frac{1}{2} - \frac{1}{2}\,\mathrm{erf}\left(\frac{d_{kj}}{\sqrt{2V_{kj}}}\right) \tag{5.18}$$

The robot can only survive to the end of the motion plan if it successfully avoids each obstacle at each time point, so the overall collision probability is:

$$P_{hit} = 1 - \prod_{k=0}^{N}\prod_{j=1}^{\#\,obs}(1 - P_{hit,kj}). \tag{5.19}$$

5.2.5 Planning Algorithm

The A* planning algorithm is used to find the optimal collision-free path from the start position to the goal position. We use the same plan cost as in Chapter 3:

$$g = \frac{T}{1 - P_{hit}} \tag{5.20}$$

where the plan duration T is simply the plan length N. The plan cost does not explicitly include the expected parameter variance $E[\mathbf{P}_N]$; the cost is a more subtle function of the learning process. Note that typical planners for grid worlds do not apply to this problem because the plan cost at each location depends on the path used to get there; the search space is actually a tree structure rather than a grid.

The heuristic function used by the planner, h, is the same as in the original motion planning problem (3.149) using a visibility graph connecting the obstacle corners. In this case, however, the robot's reference path can only move between adjacent grid points so the Manhattan distance is used as the cost of each graph edge.

The search procedure is shown in Algorithm 10. The planner branches out from the search tree nodes by concatenation an action a from the set $\{\mathtt{N}, \mathtt{S}, \mathtt{E}, \mathtt{W}\}$, starting with a node at the initial position. Branches are discarded if they intersect with an obstacle. For each new plan the cost is computed using the predicted parameter variance and the predicted error variance, and the plan is inserted into the search queue sorted by the predicted total cost $f = g + h$. As long as a free path to the goal exists (which can be verified using the heuristic's network graph from the start location), then the algorithm returns the optimal plan.

5.2.6 Simulation Results

The planner shown in Algorithm 10 was run on two different maps to show how the integrated planner optimally balances exploration of the state space with goal-seeking behavior. The first example is shown in Figure 5-4. The robot, indicated by the red circle, must drive to the red square. The shortest path is to drive straight into the hallway and up to the goal: `EEEEEENNN`. However, with that motion plan

Algorithm 10 Expanding A* Motion Planning Algorithm for the Simple 2D Example.

```
1: Load the obstacle map and the goal position.
2: Construct the obstacle graph 𝒟.
3: Load the a priori parameter estimates B̂_0 and the initial variance P_0.
4: Initialize the search queue 𝒬 with a plan containing only the start position.
5: while 𝒬 is not empty do
6:    Remove the first plan from the queue, p ← 𝒬(0).
7:    if p ends at the goal then
8:       return p with success.
9:    end if
10:   for each action a ∈ {N, S, E, W} do
11:      Add the action to p:  p' ← p + a.
12:      if p' is nominally collision-free then
13:         Compute the cost g(p').
14:         Compute the predicted cost-to-go h(p').
15:         Insert p' into 𝒬 sorted by f(p') = g(p') + h(p').
16:      end if
17:   end for
18: end while
19: return with failure.
```

the first time the robot learns about the Y parameter is when the robot first drives North in the hallway, where path errors have a high probability of resulting in a collision. With the initial parameter estimate $\hat{\mathbf{B}}_0 = \mathbf{B}$ and an initial parameter variance $\mathbf{P}_0 = 2\mathbf{I}$ with $W = 0.3$, the planner chooses the plan `ESENEEEENNN` as shown in the figure. This plan was found after 2,372 iterations of the A* algorithm, which took 0.653 seconds on a 2.33 MHz Intel Core 2 Duo processor. The predicted collision probability for this motion plan is 16.6%. The collision probability measured from a set of 10,000 Monte Carlo simulations is 21.9%. The difference can be attributed to the various approximations used in the predictions of $E[P_k]$ and V_k. By contrast the predicted collision probability for the shortest plan `EEEEEENNN` is 33.1% and the measured collision probability is 34.5%.

The *a priori* parameter estimates are actually correct in the first example, but the low confidence (high initial variance) of those parameters causes the planner to choose actions that improve the confidence in the parameters before entering the constrained region of the space. It is more realistic to assume that the *a priori* parameter estimates

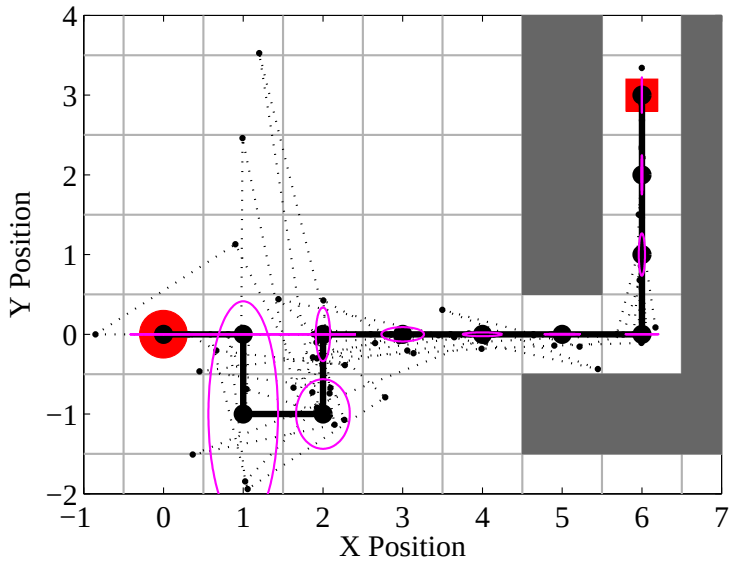

Figure 5-4: The vehicle chooses to practice the North/South motion before entering the hallway to improve the parameter convergence and reduce path errors later in the plan. The ellipses show one standard deviation of predicted position error at each step of the plan, and the dotted lines show 10 representative Monte Carlo simulations.

are wrong. While this affects the *values* of the resulting parameter convergence and error variance predictions, it does not affect the *trends*, that is, the shape of the curves in Figure 5-3 is the same even if the scale is not exactly correct. Continuing this logic, the planner will choose motion plans that explore the action space as long as the initial parameter variance is large.

To investigate the potential effects of incorrect *a priori* parameter values, the same example was run with $\hat{b}_{x,0} = 2$ and $\hat{b}_{y,0} = -0.5$, which are 200% and 50% of the true values, respectively. The planner returned the same motion plan `ESENEEEENNN` with a predicted collision probability of 14.8% and a measured collision probability of 23.5%. In this case the prediction is farther from the measured value, but the planner still chose actions to learn the correct parameter values.

A second example is shown in Figure 5-5. Using correct *a priori* parameters and $\mathbf{P}_0 = \mathbf{I}$ with $W = 0.1$, the optimal plan `WSNEEENNEESS` was found after 0.345 seconds in 1,044 iterations. The predicted collision probability is 20.5% and the measured collision probability from a 10,000-point Monte Carlo simulation is 33.6%. When the

incorrect *a priori* parameters are used (the same as in the previous example), the planner chooses the similar plan `SNWEEENNEESS`. In this case the predicted collision probability is 18.9% and the measured collision probability is 36.0%.

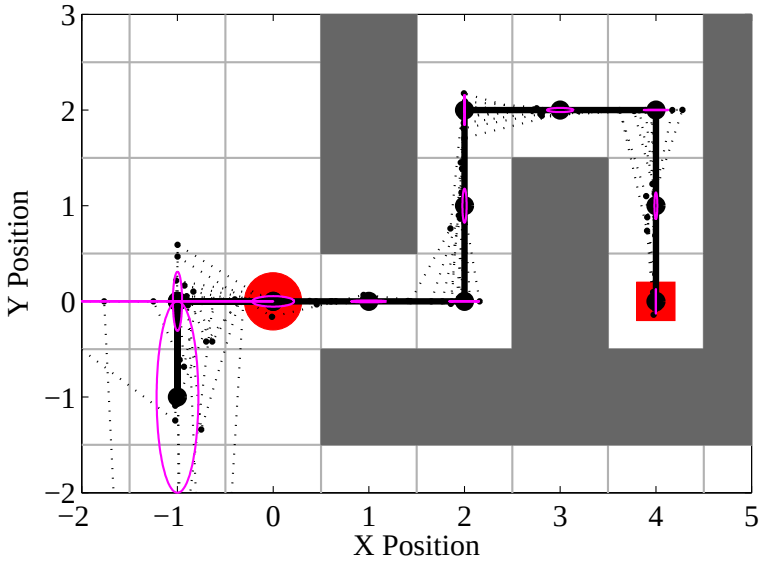

Figure 5-5: The vehicle backs away from the hallway before entering to improve the parameter convergence and reduce path errors later in the plan. The ellipses show one standard deviation of predicted position error at each step of the plan, and the dotted lines show 10 representative Monte Carlo simulations.

5.2.7 Analysis

In both examples, the planner chooses motion plans that expose the robot to actions that will be needed later in the constrained region of the space. These actions are practiced in the open regions of the space where large position errors can be tolerated. Practicing the actions has the most impact when the initial parameter variance is large; in that case it takes very few maneuvers to reduce the parameter variance and position error variance. As the information R increases, the effect of each subsequent action on the parameter error is reduced. Therefore the integrated planning and learning algorithm is ideally suited for situations in which the robot is starting from a clean slate (very little *a priori* knowledge) or the robot senses a configuration change

and chooses to reduce its confidence (increase the parameter variance) to quickly learn the new parameter values.

5.2.8 Conditions for Practicing

Consider a motion plan p_0 that is the shortest path from the start position to the goal position. The duration is of this plan is T_0 and the collision probability is $P_{hit,0}$, resulting in a plan cost $g_0 = T_0/(1 - P_{hit,0})$. An alternative plan p_1 cannot have a shorter duration: $T_1 \geq T_0$. We are interested in the conditions on p_1 that make it more optimal than p_0, that is, $g_1 < g_0$.

$$\begin{aligned} \frac{T_1}{1 - P_{hit,1}} &< \frac{T_0}{1 - P_{hit,0}} \\ P_{hit,1} &< 1 - \frac{T_1}{T_0}(1 - P_{hit,0}) \\ &< 1 - \frac{T_1}{T_0} + \frac{T_1}{T_0} P_{hit,0} \\ &< \left(1 + \frac{\Delta T}{T_0}\right) P_{hit,0} - \frac{\Delta T}{T_0} \end{aligned} \tag{5.21}$$

Next we can examine the two extreme cases of (5.21).

- When the shortest path has a very high collision probability ($P_{hit,0} \to 1$) then any additional duration Δt will be acceptable as long as $P_{hit,1} < P_{hit,0}$.

- When the shortest path has a low collision probability ($P_{hit,0} \to 0$) then the collision probability must be reduced by $\Delta T/T_0$ for the planner to choose p_1 over p_0. Because $P_{hit,1}$ has a lower-bound of 0, p_1 will never be chosen if $\Delta T/T_0 > P_{hit,0}$.

Consider the very simple example shown in Figure 5-6 with $\mathbf{P}_0 = \mathbf{I}$ and $\mathbf{W} = 0.1$. One of the shortest paths is $p_0 =$ `EEEEN` ($T_0 = 5$) which drives parallel to the wall before moving up to the goal. The collision probability for this plan is $P_{hit,0} = 42.1\%$. The main contribution to the collision probability is in the last step, when the robot approaches within $d = 0.2$ meters of the wall. The planner returns the optimal

path p_1 = `SENEEEN` ($\Delta T = 2$) shown in the figure with a collision probability of $P_{hit,1} = 14.3\%$.

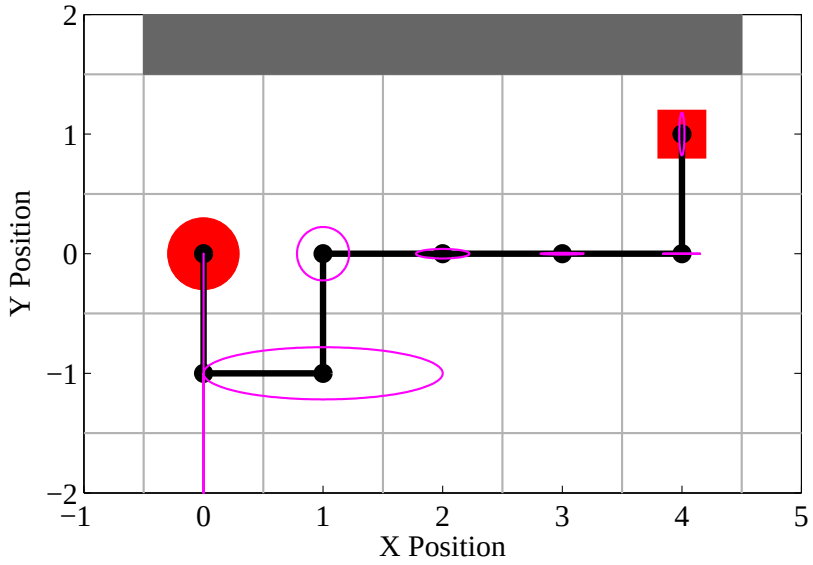

Figure 5-6: A very simple planning problem to study the conditions for using a practicing maneuver as shown.

Without practicing, the expected Y parameter variance is $E[P_{y,0}] = P_0$. From (5.15), the Y variance after the first North action in the hallway is $V_y = P_0/b^2$. Ignoring all other sources of collision probability, which we assume to be negligible, $P_{hit,0}$ is then:

$$P_{hit,0} = \frac{1}{2} - \frac{1}{2}\,\mathrm{erf}\left(\frac{d|b|}{\sqrt{2P_0}}\right) \quad \text{(no practicing)} \tag{5.22}$$

The shortest possible practicing maneuver in this example is a single movement South and a single movement North, or vice versa. (The actions need not be adjacent in the motion plan, as shown in Figure 5-4.) From (5.13), the resulting parameter

variance after each step is:

$$\begin{aligned} E[P_{y,1}] &= W\left(\frac{1}{b^2}(1+P_0)+\frac{W}{P_0}\right)^{-1} \\ E[P_{y,2}] &= W\left(\frac{1}{b^2}(1+P_0)+\frac{1}{b^2}\left(1+E[P_{y,1}]\right)+\frac{W}{P_0}\right)^{-1} \\ &= W\left(\frac{2}{b^2}+\frac{P_0}{b^2}+\frac{W}{P_0}+\frac{W}{1+P_0+b^2W/P_0}\right)^{-1} \end{aligned} \tag{5.23}$$

and from (5.15) the Y variance is:

$$\begin{aligned} V_y &= E[P_{y,2}]\frac{1}{b^2} \\ &= W\left(2+P_0+\frac{b^2W}{P_0}+\frac{b^2W}{1+P_0+W/P_0}\right)^{-1} \end{aligned} \tag{5.24}$$

Finally, the collision probability is:

$$P_{hit,1} = \frac{1}{2}-\frac{1}{2}\,\mathrm{erf}\left(\frac{d}{\sqrt{2W}}\sqrt{2+P_0+\frac{b^2W}{P_0}+\frac{b^2W}{1+P_0+W/P_0}}\right) \quad \text{(practicing)} \tag{5.25}$$

Combining 5.21 with 5.22 and 5.25, we arrive at the following inequality. When this inequality is satisfied, the vehicle will execute a practicing maneuver:

$$\mathrm{erf}\left(\frac{d}{\sqrt{2W}}\sqrt{2+P_0+\frac{b^2W}{P_0}+\frac{b^2W}{1+P_0+W/P_0}}\right)-\left(1+\frac{\Delta T}{T_0}\right)\mathrm{erf}\left(\frac{d|b|}{\sqrt{2P_0}}\right)-\frac{\Delta T}{T_0}>0 \tag{5.26}$$

We can numerically evaluate the combinations of P_0 and W that will result in a practicing behavior. These combinations form the white region of Figure 5-7. Note that the values used in the example, $P_0 = 1$ and $W = 0.1$, fall into this region.

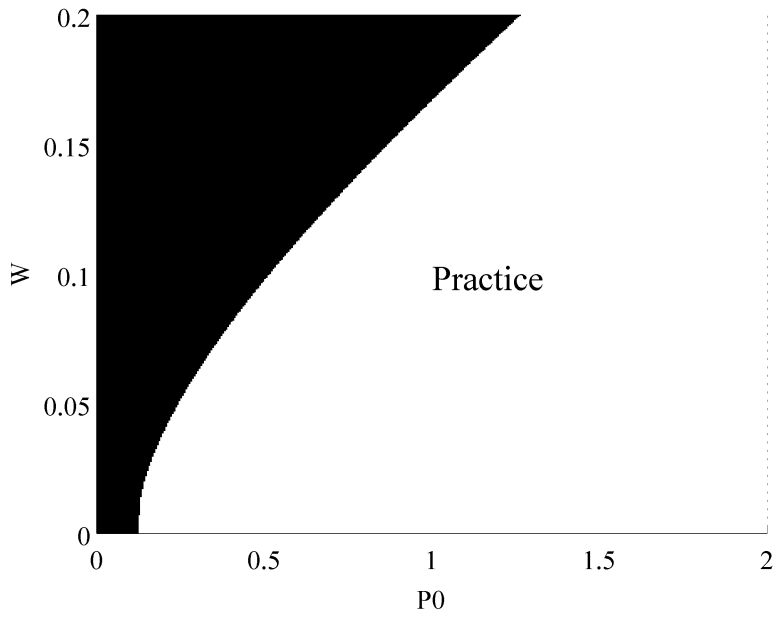

Figure 5-7: The white region represents the combinations of P_0 and W that will result in the practicing behavior shown in Figure 5-6. In the dark regions the shortest path `EEEEN` is the optimal path.

5.3 Predicting the Effects of Model Uncertainty with Hermite Quadrature

In Chapter 4 we derived the parameter covariance evolution for a motion plan for a planar holonomic vehicle. To implement the integrated planning and model learning algorithm described in Section 5.1 we also need to be able to predict the increased collision probability due to model uncertainty. For the simple 2D robot, the error variance is a direct function of the parameter variance; as the Gaussian nature of the error variance is preserved, the collision probability is a simple calculation from the error variance. These calculations are not simple for the planar holonomic vehicle for several reasons:

- Disturbances are a continuous random process, while parameter uncertainty acts as a random but steady effect on each path leg.

- The controller is a linear quadratic regulator designed around the model pa-

rameter estimates after each maneuver. The controller gains can be linearized around small parameter errors, but the effects of those controller gains on the error variance are not analytically computable.

- If the controller is designed around an estimated system that is very different from the true system, then the controller may be unstable when applied to the true system. With Gaussian process and sensor noise models, this situation cannot be ignored.

As in Chapters 3 and 4, we seek analytic predictions of the collision probability given the parameter covariance evolution. This process can be considered to be a function with a normally distributed parameter vector. We are seeking the expected output of that function. This solution can be approximated using numerical quadrature. With numerical quadrature, an integral is approximated by sampling the integrand at specific points and combining the results with a weighting function. Hermite quadrature is a version of numerical quadrature using a Gaussian kernel. As shown below, Hermite quadrature can be used to approximate the expected value of a function with a normally distributed input. Subsequently we will apply this quadrature rule to the planning and learning problem. See [16] for a useful summary of numerical quadrature techniques.

5.3.1 Hermite Quadrature Integration

The Hermite quadrature rule states that the integral of a function $g(x)$ multiplied by a Gaussian function is approximately equal to the weighted sum of the function evaluated at specific points. As the number of evaluation points increases, the accuracy of the approximation improves.

$$\begin{aligned} \int_{-\infty}^{\infty} e^{-x^2} g(x)\, dx &= \sum_{j=1}^{\infty} w_j\, g(x_j) \\ &\approx \sum_{j=1}^{n_H} w_j\, g(x_j) \end{aligned} \tag{5.27}$$

Hermite quadrature uses a Gaussian kernel, while other quadrature rules use different kernels. Next we recall the definition of the expected value of a function $g(x)$ of a random variable x whose probability density function is $f(x)$:

$$E[g(x)] = \int_{-\infty}^{\infty} g(x) f(x)\, dx. \tag{5.28}$$

When x is sampled from a normalized Gaussian distribution, $x \sim N(0,1)$ and $f(x) = e^{-x^2}$, then (5.27) and (5.28) can be combined:

$$E[g(x)] \approx \sum_{j=1}^{n_H} w_j\, g(x_j). \tag{5.29}$$

The quadrature point locations and weights are computed using the following procedure. First, a $n_H \times n_H$ matrix H is constructed as shown below.

$$\mathbf{H} = \begin{bmatrix} 0 & \sqrt{1} & 0 & \cdots & 0 & 0 \\ \sqrt{1} & 0 & \sqrt{2} & \cdots & 0 & 0 \\ 0 & \sqrt{2} & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sqrt{n_H - 1} \\ 0 & 0 & 0 & 0 & \sqrt{n_H - 1} & 0 \end{bmatrix} \tag{5.30}$$

Next we define $\mathbf{V}$ and $\mathbf{D}$ as the eigenvalue decomposition of $\mathbf{H}$, where $\mathbf{D}$ is a diagonal matrix of eigenvalues and $\mathbf{V}$ is the corresponding matrix of eigenvectors. The eigenvalues of $\mathbf{H}$ are the quadrature point locations and the weights are the square of the first element of each eigenvector: for the j'th point, $x_j = D_{jj}$ and $w_j = {V_{1j}}^2$.

The extension of (5.29) to the case where $\mathbf{x}$ is sampled from a multinormal distribution $\mathbf{x} \sim N(\mathbf{0}, \mathbf{I})$ is straightforward using the tensor product. If there are n_H quadrature points in each of m dimensions, then there are $N_H = {n_H}^m$ quadrature points overall. The point locations in the m-dimensional space are defined by the scalar point locations along each dimension, and the point weights are the product

of the component weights. The point locations form a $m \times N_H$ matrix $\mathbf{x_H}$ and the weights form a $N_H \times 1$ vector $\mathbf{w_H}$.

The vectors in $\mathbf{x_H}$ correspond to the multinormal distribution $N(\mathbf{0}, \mathbf{I})$, but the parameters themselves follow the distribution $\boldsymbol{\beta} \sim N(\bar{\boldsymbol{\beta}}, \mathbf{P})$. The following procedure is used to transform a particular point $\mathbf{x}_j$ into its corresponding parameter vector $\boldsymbol{\beta}_j$. First we define $\mathbf{V_P}$ and $\mathbf{D_P}$ as the eigenvalue decomposition of $\mathbf{P}$, where $\mathbf{D_P}$ is a diagonal matrix containing the eigenvalues of $\mathbf{P}$. Next we compute $\boldsymbol{\beta}_j$ as follows:

$$\boldsymbol{\beta}_j = \mathbf{V_P} \sqrt{\mathbf{D_P}}\, \mathbf{x}_j + \bar{\boldsymbol{\beta}}. \tag{5.31}$$

The output we are interested in is the collision probability, P_{hit}. To find the expected collision probability due to the model uncertainty, we need to know the collision probability associated with each quadrature point $\mathbf{x}_j$ (or equivalently $\boldsymbol{\beta}_j$) in the parameter space. The collision probability for each quadrature point can be predicted analytically if we can express the mean error and the error variance as a function of the parameter error for that quadrature point. This problem is studied below.

5.3.2 Error Distribution Predictions for Systems with Model Error

In Section 3.6 we considered the evolution of the mean and variance of the state error when executing a maneuver. In that section, the only effects that contributed to state error were process noise disturbances, underactuation, and transients when the speed control setpoint changed. We can also include the effects of model error if the true dynamic model $\{\mathbf{a}(\boldsymbol{\nu}), \mathbf{b}\}$ and the estimated model $\{\hat{\mathbf{a}}(\boldsymbol{\nu}), \hat{\mathbf{b}}\}$ are known. In this case the LQR controller is designed around the estimated plant, and the feedforward matrix $\mathbf{K_r}(\boldsymbol{\nu})$ in (3.68) becomes:

$$\mathbf{K_r}(\boldsymbol{\nu}) = \left[\begin{array}{cc} \mathbf{K_\nu} - \hat{\mathbf{b}}^{(inv)} \hat{\mathbf{a}}(\boldsymbol{\nu}) & \mathbf{K_\eta} \end{array} \right]. \tag{5.32}$$

Propagating this change, the steady forcing term (3.72) becomes:

$$\mathbf{d}(\boldsymbol{\nu}_\mathbf{r}) \equiv \begin{bmatrix} \left(\mathbf{a}(\boldsymbol{\nu}_\mathbf{r}) - \mathbf{b}\hat{\mathbf{b}}^{(inv)}\hat{\mathbf{a}}(\boldsymbol{\nu}_\mathbf{r})\right)\boldsymbol{\nu}_\mathbf{r} \\ \mathbf{0} \end{bmatrix} \tag{5.33}$$

The subsequent calculation of $\bar{\mathbf{e}}(t)$ and $\boldsymbol{\Sigma}(t)$ proceeds as normal in Section 3.6. The parameter error has a strong affect on the mean state error due to the feedforward term, and it has a weak effect on the error variance due to the design of the LQR controller. Next we present an example to show how these predictions are used.

5.3.3 Two-Parameter Example

Consider the following simple 2-parameter system, which represents a double integrator with damping. The state vector is $\mathbf{x} = [v,\, y]^T$ where y is position and v is velocity.

$$\begin{aligned} \begin{bmatrix} \dot{v} \\ \dot{y} \end{bmatrix} &= \begin{bmatrix} a & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} v \\ y \end{bmatrix} + \begin{bmatrix} b \\ 0 \end{bmatrix} \tau + \begin{bmatrix} w \\ 0 \end{bmatrix} \\ \dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\tau + \mathbf{w} \end{aligned} \tag{5.34}$$

An LQR controller $\mathbf{K}_\mathbf{x}$ is designed around the parameter estimates $\hat{a}$ and $\hat{b}$, which are stored in the parameter vector $\boldsymbol{\beta} = [\hat{a},\, \hat{b}]^T \sim N(\bar{\boldsymbol{\beta}}, \mathbf{P})$. In this example, the covariance matrix $\mathbf{P}$ is diagonal, with $\bar{a} = a = -0.3$, $\bar{b} = b = 1$, $\sigma_a = 0.1$, and $\sigma_b = 0.4$. The variance of w is 1. The LQR matrices are $\mathbf{Q} = \text{diag}([1,\, 0.01])$ and $R = 10$. For a reference state $\mathbf{r}(t) = [v_r,\, v_r t]^T$, the control law is:

$$\begin{aligned} \tau &= \mathbf{K}_\mathbf{r}\mathbf{r} - \mathbf{K}_\mathbf{x}\mathbf{x} \\ \text{where} \quad \mathbf{K}_\mathbf{r} &= [\mathbf{K}_\mathbf{x}(1) - \hat{a}/\hat{b},\ \mathbf{K}_\mathbf{x}(2)]. \end{aligned} \tag{5.35}$$

From the derivation above, we can predict the mean and variance of the path-following error $\mathbf{e} = \mathbf{x} - \mathbf{r}$ due to the process noise when using a particular $\boldsymbol{\beta}_j$. Using Hermite quadrature, we can choose specific sample points $\boldsymbol{\beta}_j$ around $\bar{\boldsymbol{\beta}}$. For a mth

order Hermite quadrature, there are $N_H = m^2$ sample points in the parameter space, each with a corresponding weighting w_j. After a certain amount of time T we can look at the distribution of the position state $y(T)$ from a 10,000-point Monte Carlo simulation and compare it to the distribution predicted by Hermite quadrature. The Hermite distribution is:

$$p(y)^{(q)} = \sum_{j=1}^{N_H} w_j \times \frac{1}{\sqrt{2\pi\Sigma_j}} \exp\left(-\frac{(y-\bar{y}_j)^2}{2\Sigma_j}\right) \tag{5.36}$$

where $\bar{y}_j$ is the mean position for the jth quadrature point and Σ_j is the variance of the position. This distribution is a true PDF because its integral is 1. Next we consider an obstacle at a certain position d_{obs} in the space at $t = T$. If $y(T) > d_{obs}$, then a collision has occurred. Due to the linear construction of the PDF, the collision probability can be combined using the same quadrature rule:

$$P_{hit}{}^{(q)} = \sum_{j=1}^{N_H} w_j \times P_{hit,j} \quad = \quad \sum_{j=1}^{N_H} w_j \times \frac{1}{2}\left(1 - \text{erf}\left(\frac{d_{obs} - \bar{y}_j}{\sqrt{2\Sigma_j}}\right)\right) \tag{5.37}$$

where $P_{hit,j}$ is the collision probability computed from the jth quadrature point.

Figures 5-8 through 5-11 show the quadrature point locations and the resulting PDF for $m = \{1, 2, 4, 6\}$. As m increases, the PDF from (5.36) converges to the PDF as determined by the 10,000-point Monte Carlo simulation. The $m = 1$ case uses a single point at the nominal $\bar{\boldsymbol{\beta}}$, so it does not include any information about model uncertainty; the PDF and the predicted collision probability are from the process noise alone. For large m, some of the quadrature points result in an unstable controller when applied to the true system. The quadrature weights for these points are so low that the resulting large state errors have little effect on the PDF.

The collision probability is computed using (5.37). As we would expect, the accuracy of this prediction increases as the quadrature order increases. Table 5.1 shows the convergence of $P_{hit}{}^{(q)}$ to the 10,000-point Monte Carlo result, which is taken as the true value. We can see that the quadrature prediction is several orders of magnitude faster to compute than the Monte Carlo result. The convergence is

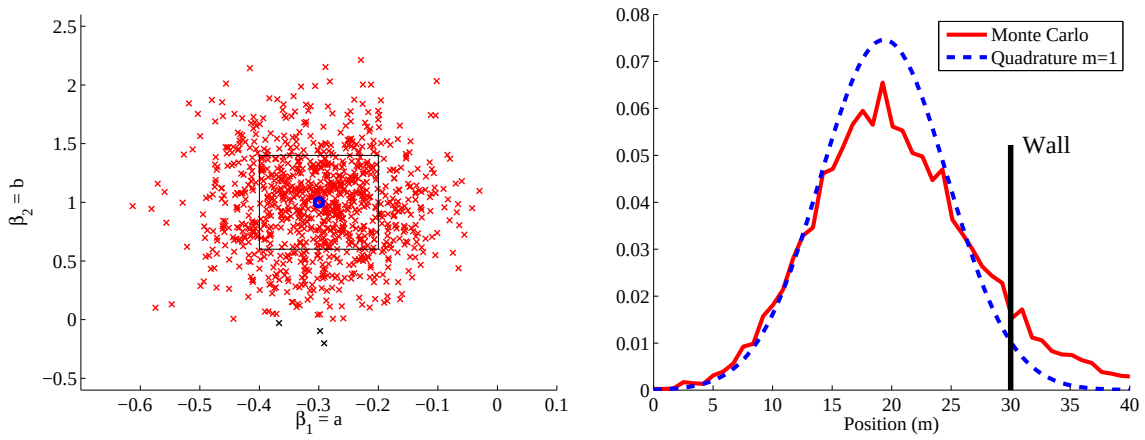

Figure 5-8: Left: The quadrature point location in the parameter space for $m = 1$ (blue circle) and the Monte Carlo parameter distribution. Right: The PDF computed using the quadrature rule (blue dashed), and the PDF computed from the Monte Carlo simulation (red). With $m = 1$ there is no information about model uncertainty in the prediction of the PDF.

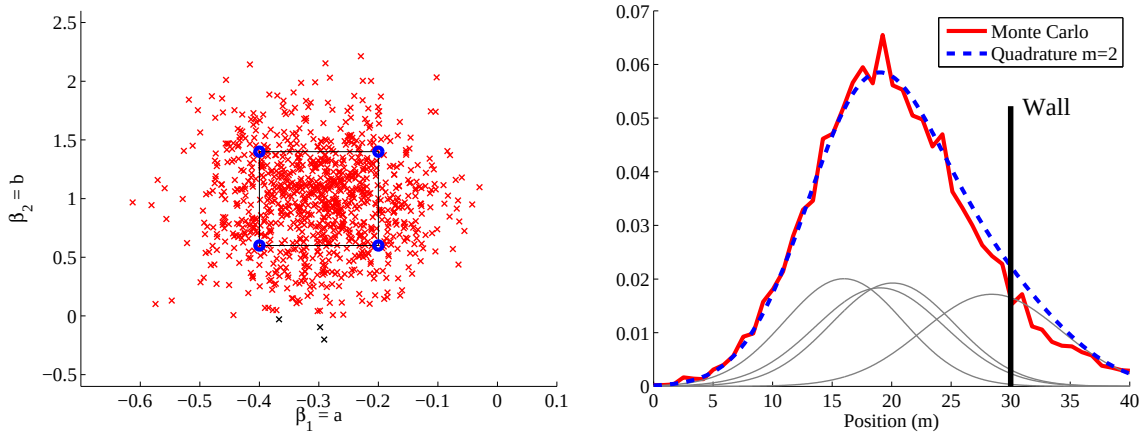

Figure 5-9: Left: The quadrature point locations in the parameter space for $m = 2$ (blue circles) and the Monte Carlo parameter distribution. Right: The PDF computed using the quadrature rule (blue dashed), and the PDF computed from the Monte Carlo simulation (red). The component Gaussian distributions for each quadrature point are shown in gray, scaled by the quadrature weights.

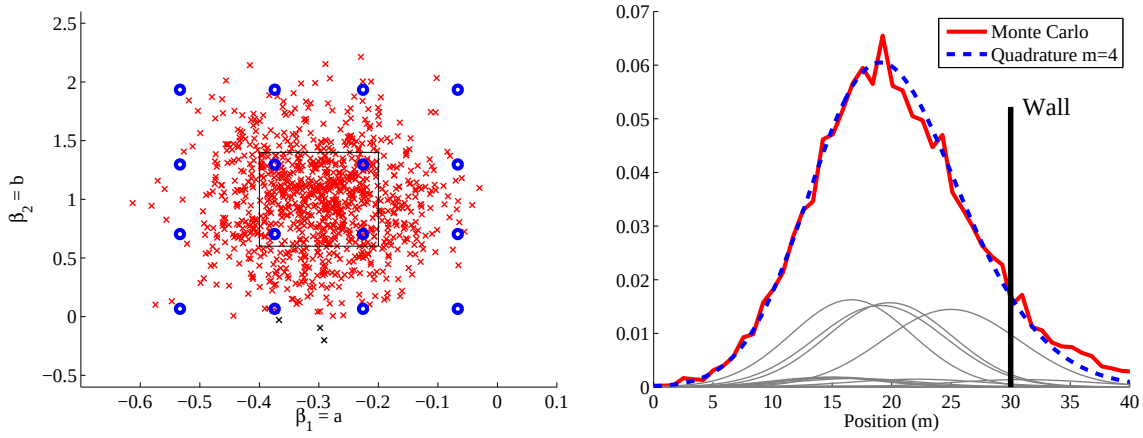

Figure 5-10: The same plots as Figure 5-9 with $m = 4$.

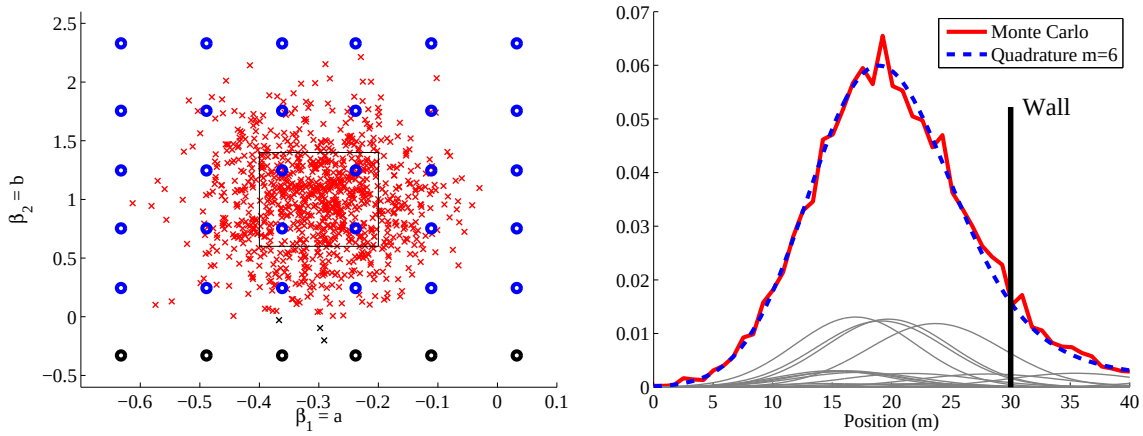

Figure 5-11: The same plots as Figure 5-9 with $m = 6$. The quadrature points shown in black ($\beta_2 < 0$) result in an unstable controller when applied to the true system.

plotted in Figure 5-12.

The procedure outlined in this section can be applied to the planar holonomic vehicle. When the parameter covariance matrix $\mathbf{P}$ changes from maneuver to maneuver, then the collision probability (and the plan cost) incorporates the model learning that will take place during the plan execution. When all of these effects are combined, the result is the integrated planning and model learning algorithm. The full algorithm is described in the next section.

Table 5.1: Convergence of the quadrature prediction of P_{hit} to the Monte Carlo result.

Method	P_{hit}	Time (sec)
Monte Carlo, 100 points	0.1100	4.407
Monte Carlo, 1,000 points	0.1050	37.13
Monte Carlo, 10,000 points	0.1176	334.5
Quadrature, $m = 1$	0.0233	0.0103
Quadrature, $m = 2$	0.1112	0.0285
Quadrature, $m = 3$	0.1496	0.0561
Quadrature, $m = 4$	0.1087	0.0978
Quadrature, $m = 5$	0.1095	0.1481
Quadrature, $m = 6$	0.1232	0.2029
Quadrature, $m = 7$	0.1152	0.3132
Quadrature, $m = 8$	0.1140	0.3529

5.4 Full Algorithm

At this point we have all of the ingredients needed to apply the integrated motion planning and model learning algorithm described in Section 5.1 to a planar holonomic vehicle. In Chapter 3 we described a motion planning framework and a search algorithm to find the optimal motion plan using a cost function based on collision probability. Chapter 4 described how to predict the parameter covariance evolution that results from following a particular motion plan, and in Section 5.3 we described how to compute the collision probability for a maneuver based on the parameter covariance. When all of these ingredients are combined, the planner chooses the best motion plan while considering the model learning and the resulting uncertainty-based

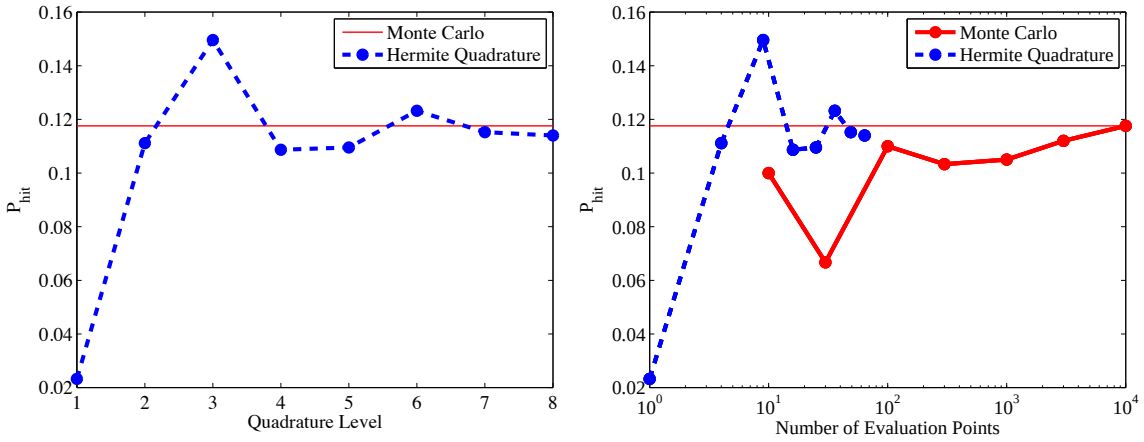

Figure 5-12: Convergence of the quadrature prediction of P_{hit} and the Monte Carlo simulation. The result of the 10,000-point Monte Carlo simulation is taken as the truth value. The $m = 8$ quadrature uses 64 evaluation points and runs approximately 1,000 times faster than the best Monte Carlo simulation, but comparable results are achievable at lower quadrature levels. (Data in Table 5.1.)

cost of each plan. As we saw in Section 5.2, such a planner exhibits automatic active learning behaviors, when necessary, to complete the mission with an optimal balance of risk and efficiency.

This section is devoted to the synthesis of all the ingredients into a robust integrated planner for planar holonomic vehicles. We begin with a high-level view.

5.4.1 High-Level View

The high-level synthesis of the different parts of the planning problem is shown in Algorithm 11. The algorithm includes the adjustments necessary between missions to update the parameter estimates and covariance based on the previous mission's data. Those steps add feedback to the planning process on a mission-to-mission level. A block diagram of Algorithm 11 is shown in Figure 5-13.

Algorithm 11 Integrated Planning and Model Learning Algorithm (High Level).

1: Initialize the *a priori* parameter estimates and the *a priori* parameter covariance matrix.
2: **for** each mission **do**
3: Load the initial state, the goal state, the obstacle positions, the *a priori* parameter estimates $\hat{\boldsymbol{\beta}}_0$ and the *a priori* parameter covariance matrix $\mathbf{P}_0$.
4: Start the A* robust motion planner (Chapter 3).
5: **for** each plan p considered by the planner **do**
6: Predict the error statistics given process noise only (Section 3.6).
7: Predict the parameter covariance evolution through the plan based on those error statistics (Section 4.4).
8: Predict the collision probability given the parameter covariance evolution using Hermite quadrature (Section 5.3).
9: Compute the plan cost $g = T/(1 - P_{hit})$.
10: Compute the heuristic function h (Section 3.8.2) and return the predicted total cost $f = g + h$.
11: **end for**
12: Return the plan that ends at the goal with the lowest cost.
13: Execute the optimal motion plan, implementing the learning algorithm (Section 4.1.4) and updating the controller (Section 3.5.3) after each maneuver.
14: Analyze the prediction error and adjust the posterior parameter covariance (Section 4.5.2).
15: Update $\hat{\boldsymbol{\beta}}_0$ and $\mathbf{P}_0$ for the next mission.
16: **end for**

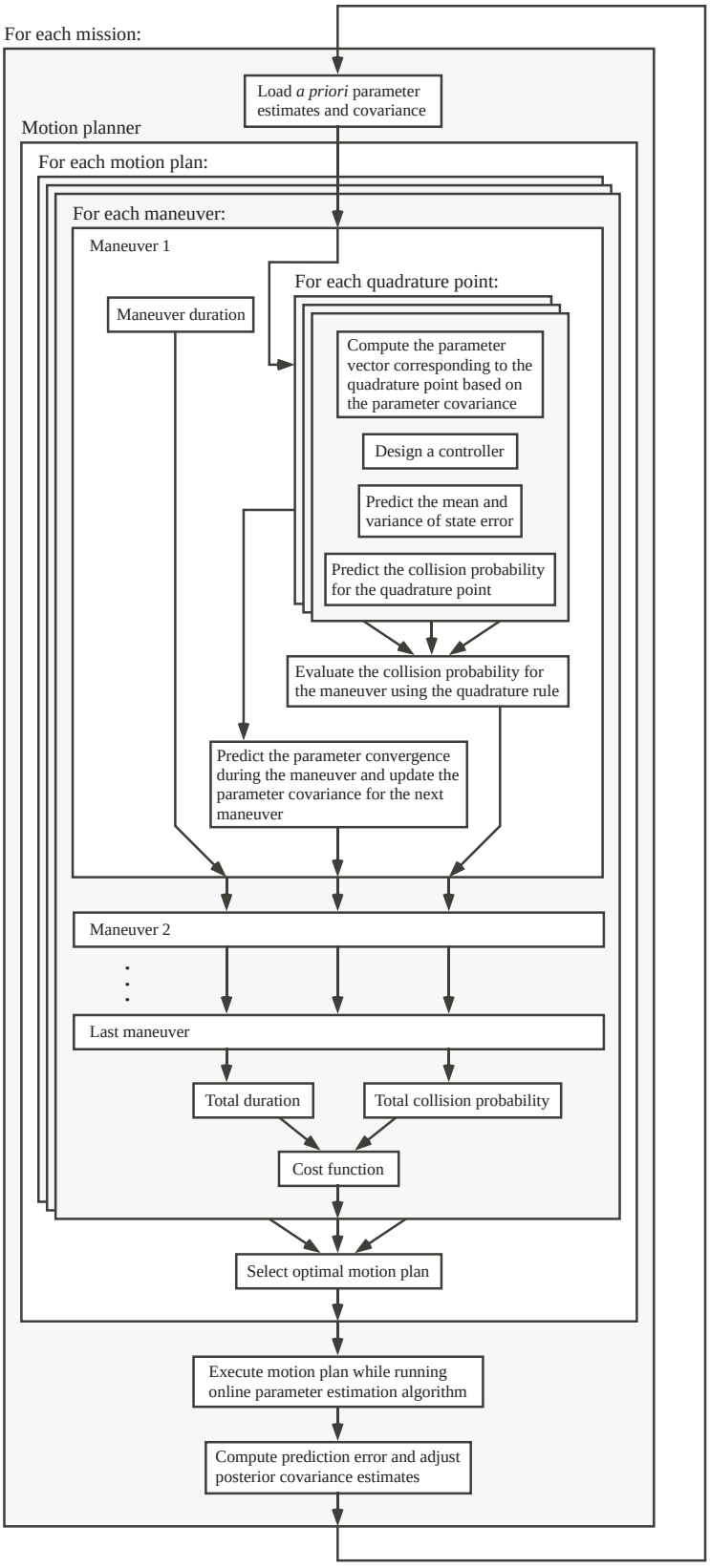

Figure 5-13: A block diagram of Algorithm 11.

5.4.2 Implementation Details

Here we discuss some of the specific issues that arise when implementing Algorithm 11 for the planar holonomic vehicle.

Error Statistics

With the overall algorithm defined, we can now study some of the particulars of the algorithm. In Algorithm 11, the parameter covariance evolution is computed based on the nominal error statistics, that is, computed based on the *a priori* parameter estimate $\hat{\boldsymbol{\beta}}_0$. Strictly speaking the parameter covariance evolution should be based on the error statistics associated with each quadrature point, essentially putting the learning prediction inside the quadrature. However, the learning data is not appreciably affected by the model error at each quadrature point, and simply using the nominal error statistics results in substantial savings in terms of computer memory usage and computation time.

Maneuver Concatenation

Next we note that each plan in Algorithm 11 is analyzed in its entirety. However, plans are constructed by adding maneuvers to existing plans, so it is only necessary to consider the effect of the new maneuver. All of the algorithms are conveniently partitioned by maneuver, so adding a maneuver and computing the incremental plan cost is straightforward.

Quadrature Points Mapped to Parameter Vectors

The quadrature points formed by the tensor product exist in the normalized orthogonal m-dimensional space; these are the $\mathbf{x}_j$ vectors from Section 5.3.1. Meanwhile, the corresponding parameter vectors change as the parameter covariance matrix changes. Consider second-order ($n_H = 2$) quadrature for $m = 2$; the four quadrature points

are shown below, with each column representing one quadrature point.

$$\mathbf{x_H} = \begin{bmatrix} -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 \end{bmatrix} \tag{5.38}$$

If $\bar{\boldsymbol{\beta}}_0 = [1, 0]^T$ and $\mathbf{P}_0 = \text{diag}([4, 4])$, then the corresponding parameter vectors are computed using (5.31):

$$\hat{\boldsymbol{\beta}}^{(q)}_{all,0} = \begin{bmatrix} -1 & -1 & 3 & 3 \\ -2 & 2 & -2 & 2 \end{bmatrix} \tag{5.39}$$

Now suppose that after one maneuver of the motion plan the estimated parameter covariance matrix is $\mathbf{P}_1 = \text{diag}([1, 0.25])$. While the quadrature point locations $\mathbf{x_H}$ remain the same in the normalized orthogonal space, we apply (5.31) again using $\mathbf{P}_1$ to obtain the new parameter vectors:

$$\hat{\boldsymbol{\beta}}^{(q)}_{all,1} = \begin{bmatrix} 0 & 0 & 2 & 2 \\ -0.5 & 0.5 & -0.5 & 0.5 \end{bmatrix} \tag{5.40}$$

Throughout the motion plan, the parameter vectors associated with each quadrature point converge toward the *a priori* parameter vector $\bar{\boldsymbol{\beta}}_0$.

Learning a Subset of Parameters

The full parameter vector for the planar holonomic vehicle has 10 parameters. It is straightforward to learn all of the parameters at once, and the associated parameter convergence predictions are simple as well. However, it is computationally infeasible to predict the effects of model uncertainty using quadrature with all 10 parameters. For the simplest useful quadrature, $n_H = 2$, the number of quadrature points is $N_H = {n_H}^m = 2^{10} = 1024$. While the calculation of the collision probability for each quadrature point has an analytic solution, the memory requirements for storing 1024 control matrices and covariance matrices for each of hundreds of motion plans, each with several maneuvers, is prohibitive.

The workaround for this issue is to only learn a subset of the parameters at once, as described in Section 4.1.3. For example, if it is known that there are no wind effects, then the wind parameters f_U and f_V need not be learned. Similarly, if it is known that the vehicle has no cross-coupling between sway and yaw damping, then a_{23} and a_{32} can be set to zero and not learned. If some of the parameters have already been learned with a high degree of confidence and it is known that they will not change during the mission, then those parameters can be fixed as well. In this manner the vehicle may only need to learn a few parameters; if three parameters are learned, then the planner only needs $2^3 = 8$ quadrature points for second-order Hermite quadrature.

Collision Probability Accuracy

As we will see in section 5.5.1, the predicted collision probability often differs from the collision probability measured by a Monte Carlo simulation by up to a factor of two. This occurs even when the mean and variance predictions are accurate. The reason for the discrepancy is that the error prediction and the parameter convergence prediction use several simplifying assumptions. For example, the error prediction assumes small state errors so that the small-angle approximation is valid, and the parameter convergence prediction uses only the first two terms of the Taylor series expansion of the inverse of the information matrix. The final distribution of trajectories is then modeled as a Gaussian function; the result of the approximations is that the tails of the distribution are not well modeled. As there are likely to be obstacles in the tails of the distribution, this means that the true collision probability is higher than predicted. However, the prediction captures all of the effects of *a priori* learning data, state excitation, process noise, state correlation, etc. and the differences between different motion plans will be revealed by the prediction as well as the collision probability measurement. For this reason we accept collision probability errors of up to a factor of two with the understanding that the various effects of learning, uncertainty and disturbances have been properly taken into account.

5.5 Simulations and Experiments

In this section we present simulation and experimental results for the integrated planning and learning algorithm. While the algorithm can be used for any motion planning problem with any initial parameter distributions, to highlight the features of the algorithm the same mission is performed four times according to the procedure shown in Figure 5-14. Throughout four missions, the model uncertainty decreases as the learning algorithm is exposed to more data. However, during the third mission the true dynamic model is changed; depending on the resulting prediction error, the planner may choose to actively reduce its confidence to relearn the parameters during the last mission. Note that each mission could be performed on a different map with a different start and goal location, but for clarity we use the same map for each mission in these examples.

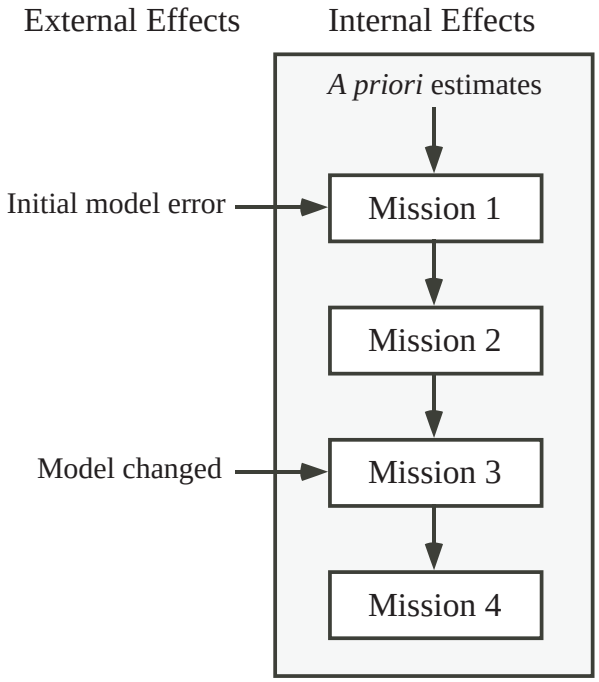

Figure 5-14: Overview of the testing procedure for the simulations and experiments.

The vehicle used in the experiments is described in Appendix A. First we present a simulation of a planning scenario in which the vehicle learns the yaw thrust gain b_3. Next we present an experiment to learn the yaw drag parameter in calm water.

The second experiment investigates the mean environmental effects in the presence of waves. Finally, using those learned environmental forcing estimates we perform a set of missions in which we learn four parameters at once.

In each simulation and experiment, the planner uses five-second trims to construct motion plans and model uncertainty is approximated with second-order quadrature. The learning data is acquired at 1 Hz in the simulation and 1.67 Hz (approximately the data rate of the vehicle's positioning system, as described in Section A-4) in the experiments. The *a priori* dynamic model and the noise model are provided in Section A.6 and the LQR weighting matrices are provided in Section A.7.

5.5.1 Simulation Learning One Parameter

To show how the integrated planning and learning algorithm applies to a planar holonomic vehicle, consider the following example. For simplicity, the only free parameter in the model is b_3, the yaw thrust gain. The *a priori* parameter value is $b_3 = -0.0527$ and the *a priori* standard deviation for b_3 is 0.05, meaning that the *a priori* standard deviation is approximately equal to the true parameter value. The task is to drive from the start position indicated in Figure 5-15 to the goal marked by a green square. For simplicity we do not let the planner use reverse maneuvers, restricting it to the trims `a-f`.

Simulation results for each mission are described below. A summary of the results is shown in Table 5.2.

Mission 1

With the problem parameters described above, the planner chooses the indicated motion plan `ecaabaw` with a duration of 32.4 seconds and a predicted collision probability of 4.6%, resulting in a cost $g = 34.0$. This plan was found after 51 iterations of the A* algorithm in 15.1 seconds on a 2.33 MHz Intel Core 2 Duo processor. A Monte Carlo simulation with 1,000 trajectories returns a collision probability of 8.8%. Figure 5-16 shows the predicted mean and standard deviation of the trajectories.

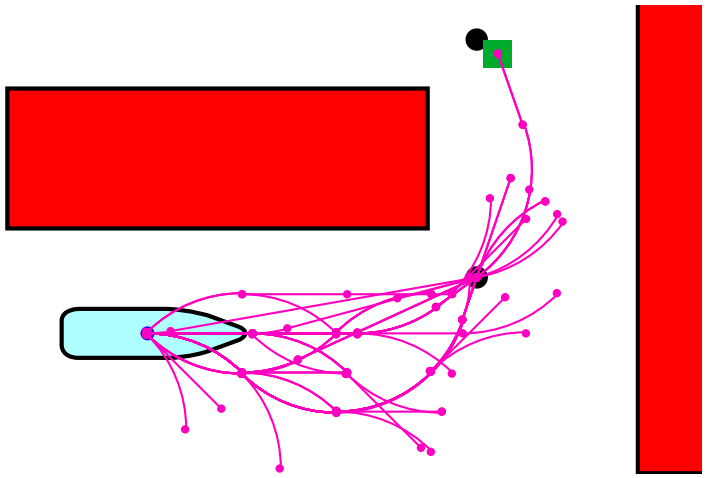

Figure 5-15: The mission is to drive from the indicated start position to the green square. The black circles are waypoints available to the planner. This figure shows the search tree expanding toward the goal.

Table 5.2: Summary of Missions for Figure 5-15.

Mission	Motion Plan	Param.	$\hat{\beta}_0$	$\sqrt{\text{var}(\beta)_0}$	$\hat{\beta}_N$	$\sqrt{\text{var}(\beta)_N}$
1	`ecaabaw` T = 32.4 sec P_{hit} = 4.6%	b_3	-0.0527	0.05	-0.031	0.0243
2	`ebbawaw` T = 28.0 sec P_{hit} = 11.2%	b_3	-0.031	0.0243	-0.042	0.0182
3	`ebbawaw` T = 28.1 sec P_{hit} = 4.6% Increased thrust gain	b_3	-0.042	0.0182	-0.044	0.0778
4	`edfcaabaw` T = 42.6 sec P_{hit} = 8.9%	b_3	-0.044	0.043	-0.077	0.0208

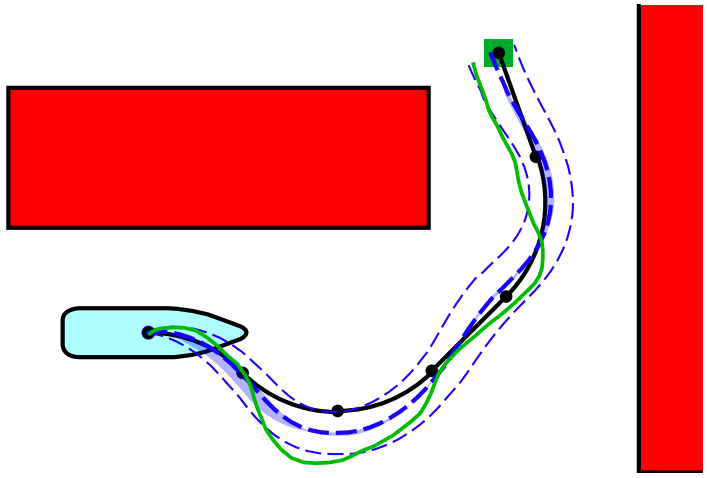

Figure 5-16: (Mission 1) With a large initial yaw parameter uncertainty, the planner chooses a motion plan that will improve the parameter estimate and the controller before driving through the passage to the goal. The blue dashed lines show the predicted mean and standard deviation of the trajectories given no parameter uncertainty, and the shaded blue region indicates the spread of the mean quadrature trajectories. A single plan execution is drawn in green.

A single plan execution is shown in Figure 5-16. For this trajectory, the final parameter estimate is $\hat{b}_3 = -0.031$. The predicted final standard deviation is 0.0243. For this run, the prediction error is within the expected bounds, so $\alpha = 1$ and the posterior standard deviation is also 0.0243. Note that the true parameter value, $b_3 = -0.0527$, is within one standard deviation of the estimate.

If the evolution of the model uncertainty is not taken into account, then the optimal plan is `ebbawaw`, with a duration of 28.1 seconds. However, including the effects of parameter uncertainty its predicted collision probability is 21.7%, resulting in a cost $g = 35.9$. The collision probability measured from a 1,000-point Monte Carlo simulation is 32.0%. This motion plan is shown in Figure 5-17.

Mission 2

Next, the planner is run a second time with the *a priori* parameter values and estimates taken from the posterior values from the first plan execution. This time, the increased confidence from the first plan execution causes the planner to choose the shorter motion plan `ebbawaw`, as shown in Figure 5-18. This plan has a duration

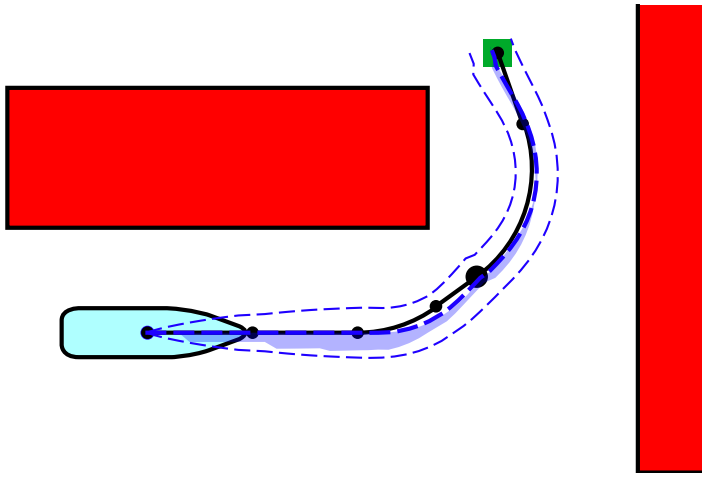

Figure 5-17: The optimal motion plan not considering model learning drives directly to the goal. This motion plan is shorter than the plan shown in Figure 5-16, but it has a higher collision probability because of the large parameter errors near the obstacles (shaded blue region).

of 28.0 seconds and a predicted collision probability of 11.2%, resulting in a cost of $g = 31.5$. The plan was found in 7.8 seconds after 29 iterations. It has a shorter duration than the plan shown in Figure 5-17 because of the new parameter estimate for b_3 results in a smaller predicted turning radius for the vehicle. The collision probability calculated from a 1,000-point Monte Carlo simulation is 23.1%.

A single plan execution is shown in Figure 5-18. For this trajectory, the final parameter estimate is $\hat{b}_3 = -0.042$. The predicted final standard deviation is 0.0182. Again, the prediction error is within the expected bounds, so $\alpha = 1$ and the posterior variance is also 0.0182.

Mission 3

The planner is run a third time with the *a priori* parameter values and estimates taken from the posterior values from the second plan execution, and it returns the same motion plan, `ebbawaw`. However, this time we increase the true parameter value by a factor of two to $b_3 = -0.1054$. This change could model a different thruster being installed on the vehicle or a different amplifier gain. The execution of this plan is shown in Figure 5-19. Due to the *a priori* confidence in the old parameter

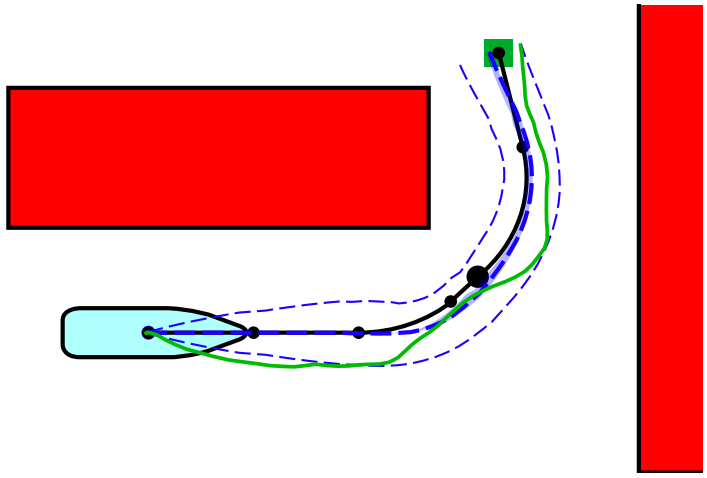

Figure 5-18: (Mission 2) Because the parameter uncertainty was reduced in the first run (Figure 5-16), the planner chooses a more direct path to the goal. A single plan execution is drawn in green.

estimate, the new estimate does not change much: now $\hat{b}_3 = -0.044$. However, this value results in a large prediction error. The posterior correction factor computed from the learning data is $\alpha = 23.8$, so the posterior standard deviation is adjusted from the predicted value of 0.0159 to 0.0778. This adjusted value more than accounts for the difference between the parameter estimate and the new true parameter value.

Mission 4

Finally, we run the planner a fourth time to find a motion plan that will best learn the new parameter value. However, because we can see that the posterior variance estimate from the previous plan execution would result in an unstable controller at one of the quadrature points ($\hat{b}_3^{(q)} = -0.044 \pm 0.0778$ for second-order quadrature), we reduce the parameter standard deviation to 0.043. In this way all of the quadrature points will be stable and the error analysis can proceed. The planner returns the motion plan `edfcaabaw` shown in Figure 5-20. This motion plan includes two stationary maneuvers at the beginning of the plan to learn the parameter before driving away from the start point. The rest of the plan is the same as in Figure 5-22. After this motion plan is executed, the new parameter estimate is $\hat{b}_3 = -0.077$ with a predicted posterior standard deviation of 0.0208. When the planner is run again, it returns

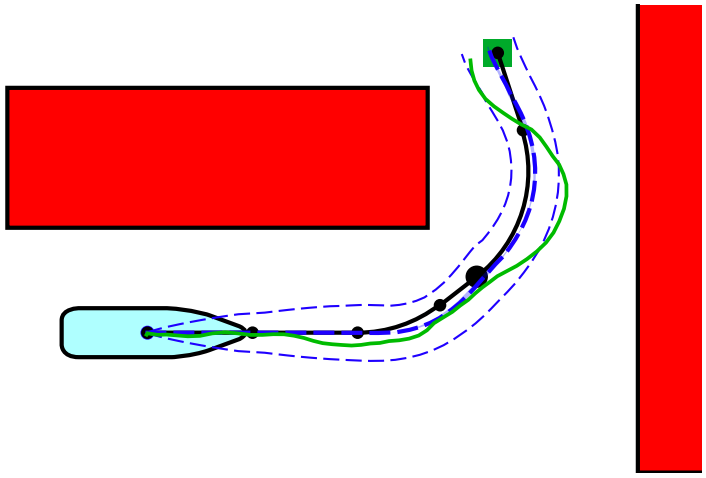

Figure 5-19: (Mission 3) After executing this plan, the learning algorithm recognizes that the true parameter value has changed. The confidence in the learned parameter value is reduced so that an active learning strategy will be chosen the next time the planner is run. A single plan execution is drawn in green.

the plan shown in Figure 5-18 because the new parameter value has been learned adequately.

In the next section we present experimental results showing a similar sequence of motion plans.

5.5.2 Experiment Learning One Parameter

The planning task for the first batch of experiments is shown in Figure 5-21. These four missions follow the same sequence as in the previous section. The vehicle must drive through a channel and make a quick left turn to arrive at the goal. Because these experiments were performed in calm water, the process noise was reduced by a factor of 100. We learn a single parameter, the yaw drag a_{33}. A summary of the results of the four missions is given in Table 5.3.

Mission 1

The *a priori* value for a_{33} is -0.1047 with a standard deviation of 0.5. In the first mission, the planner chooses the motion plan `eacwbadw` shown in Figure 5-22. This plan was found in 26.8 seconds after 150 iterations of the A* algorithm. The plan was

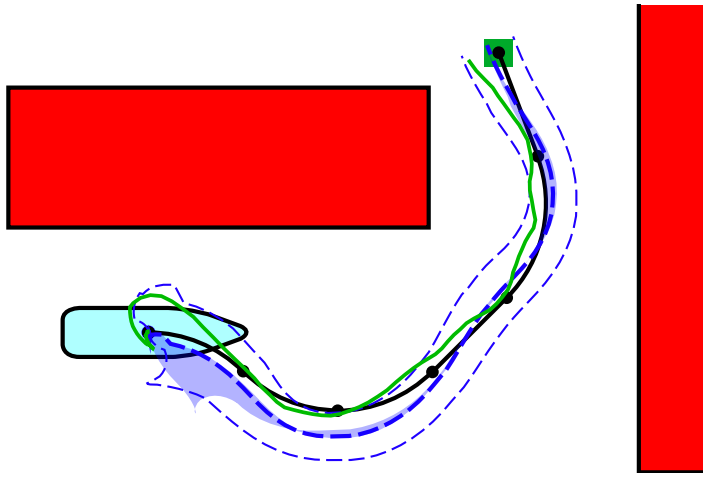

Figure 5-20: (Mission 4) The planner chooses the motion plan `edfcaabaw`, which causes the vehicle to turn back and forth in place to learn the new parameter and reduce the uncertainty before driving to the goal. The shaded blue region indicates the spread of the mean quadrature trajectories. A single plan execution is drawn in green.

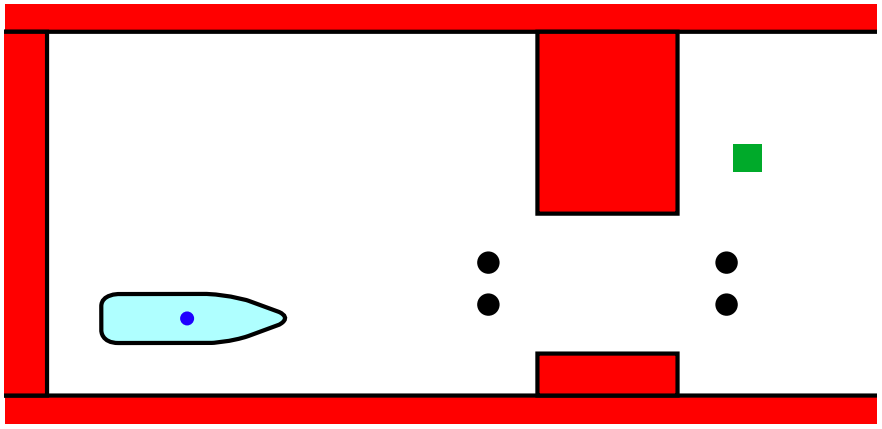

Figure 5-21: The planning task for the first batch of experiments. The goal is shown with a green square. The black circles are waypoints available to the planner. The main obstacles are one meter wide and the channel gap is one meter.

Table 5.3: Summary of Missions for Figure 5-21.

Mission	Motion Plan	Param.	$\hat{\beta}_0$	$\sqrt{\text{var}(\beta)_0}$	$\hat{\beta}_N$	$\sqrt{\text{var}(\beta)_N}$
1	`eacwbadw` T = 34.8 sec P_{hit} = 0.082%	a_{33}	-0.1047	0.5	-0.259	0.257
2	`ebawbaw` T = 30.0 sec P_{hit} = 0.0002%	a_{33}	-0.259	0.257	-0.237	0.235
3	`ebawbaw` T = 29.9 sec P_{hit} = 0.0042% Added drag	a_{33}	-0.237	0.235	-0.713	1.31
4	`ebawbawdw` T = 43.1 sec P_{hit} = 0.63%	a_{33}	-0.713	1.31	-1.334	0.209

executed using the vehicle shown in Figure A-1. The resulting trajectory is shown in the Figure 5-22.

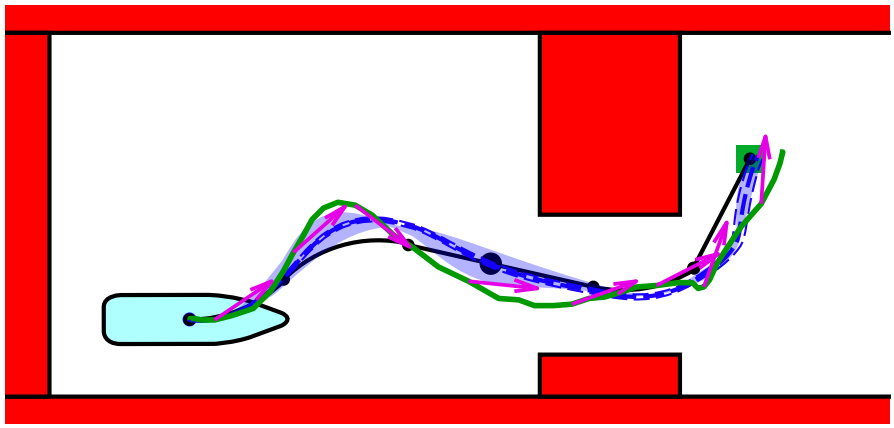

Figure 5-22: (Mission 1) The motion plan `eacwbadw` takes two turns at the beginning of the plan to learn the yaw drag parameter before entering the channel. The actual vehicle trajectory is shown in green and the heading is shown in five second increments.

After executing the first motion plan, the new parameter value is $a_{33} = -0.2585$. With no external disturbances, the prediction error is within the expected bounds ($\alpha = 1$) and the posterior parameter standard deviation is 0.257.

Mission 2

When the planner is run a second time with the improved uncertainty from the first mission, it chooses the more direct route `ebawbaw` shown in Figure 5-23. The vehicle executed this motion plan with the trajectory shown in the figure. The parameter value after this plan execution is $a_{33} = -0.2370$ with a posterior standard deviation of 0.235. Again, the prediction error is small and $\alpha = 1$.

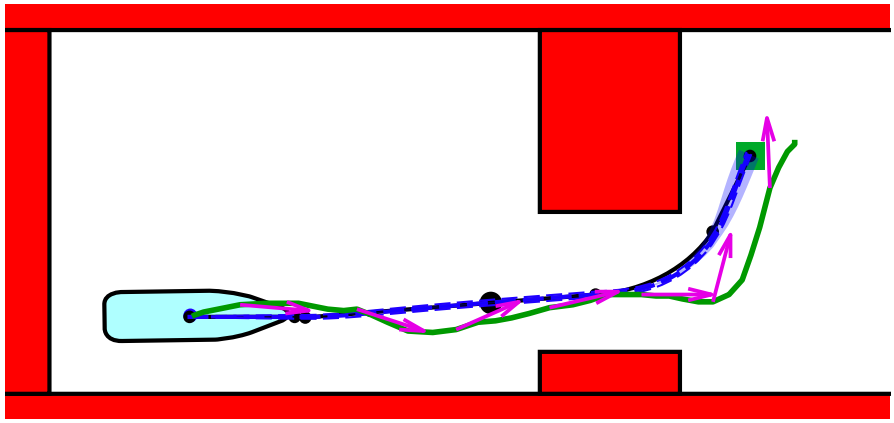

Figure 5-23: (Mission 2) The motion plan `ebawbaw` drives directly to the goal. The measured vehicle trajectory is shown in green.

Mission 3

Next we consider how the learning algorithm identifies system changes. To increase the yaw drag, we add rope to the stern of the vehicle as shown in Figure 5-24. With the increased confidence in the parameter value from the second experiment, the motion plan to get to the goal remains the same as in Figure 5-23. With the added drag, the vehicle trajectory is shown in Figure 5-25. The turning performance is reduced and the path errors are larger.

While executing the motion plan, the learning algorithm changes the parameter value to $a_{33} = -0.7134$. Afterward, the posterior variance prediction described in Section 4.5.2 identifies that the prediction error is higher than expected; as a result, it increases the posterior variance by a factor of $\alpha = 36.2$, from 0.0475 to 1.72 (a standard deviation of 1.31). This ensures that the next time the planner is run, it

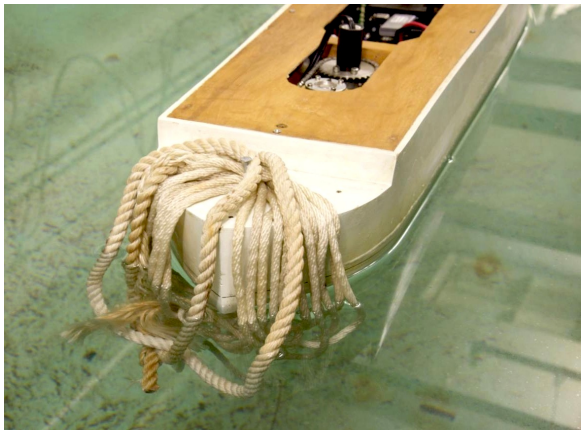

Figure 5-24: Rope was added to the stern of the vessel to increase the yaw drag for the third experiment. The resulting trajectory is shown in Figure 5-25.

will choose a plan that quickly learns the new parameter value and it will put more weight on new data.

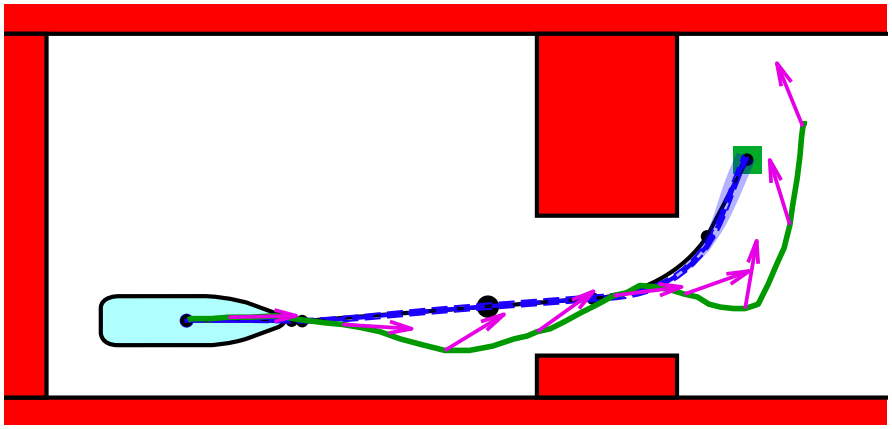

Figure 5-25: (Mission 3) The turning performance of the vehicle suffers due to added yaw drag. After executing the plan, the learning algorithm identifies that the true parameter value has changed and reduces its confidence in that parameter value accordingly. The measured vehicle trajectory is shown in green.

Mission 4

Finally, we run the planning algorithm a fourth time using the new *a priori* data obtained from the previous run. Because the drag parameter estimate has been increased, the planner knows that it is harder to turn the vehicle. The optimal

motion plan is `ebawbawdw`, which is shown in Figure 5-26. This plan does not include any maneuvers to actively learn the parameter value because it is expected that with the added drag the vehicle will not deviate very far from the straight-line path through the channel. The parameter uncertainty affects the error variance on the far side of the channel when the vehicle turns in place, but this maneuver can be safely performed despite the uncertainty because there are no obstacles nearby to hit. In other words, the shaded blue region in the figure is far enough away from obstacles that the planner accepts the risk. After executing this motion plan, the new parameter estimate is $\hat{a}_{33} = -1.334$ with a standard deviation of 0.457.

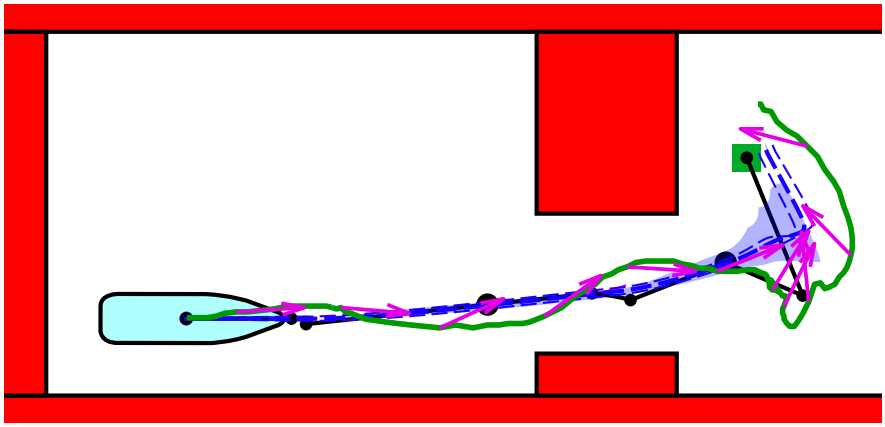

Figure 5-26: (Mission 4) The planner chooses to take a direct path to the goal, `ebawbawdw`, rather than explicitly learn the parameter, because the yaw drag estimate is large. The effects of the parameter uncertainty are only seen during the sharp turn at the end of the plan where the spread of the quadrature means (shaded blue) is large. The measured vehicle trajectory is shown in green.

5.5.3 Experiment Learning Environmental Forcing Parameters

In this experiment, we learn the steady environmental forcing terms f_U and f_V when driving through waves. The motion planning task is shown in Figure 5-27. The vehicle must back away from the wall then turn and drive into the slip. The optimal motion plan is `eihawawcw` as shown in the figure. Because we do not use any position feedback when driving in reverse (the actual vehicle is unstable in reverse) the error

variance grows rapidly. However, as soon as the vehicle starts driving forward toward the goal the error variance contracts.

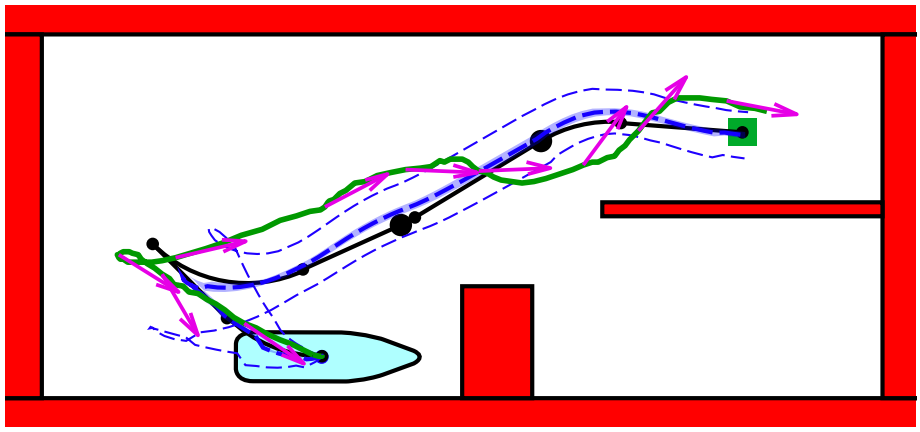

Figure 5-27: The motion plan `eihawawcw` backs the vehicle away from the wall before turning toward the goal. There is no position feedback when driving in reverse, so the error variance grows during maneuvers `i` and `h`. The vehicle trajectory is shown in green and the heading is shown in five second increments.

Without any initial knowledge of the mean environmental force, we set the *a priori* estimates to $f_U = f_V = 0$ with an *a priori* standard deviation of 0.001. The controller does not compensate for the wind effects, so the planner will never choose an active learning strategy to learn these parameters. (Learning these parameters online does not contribute to reducing the collision probability for this mission, which is the only effect considered by the planner.) However, we use second-order quadrature to predict the effects of the wind parameter uncertainty. The predicted collision probability is 30.7%, mostly from the initial maneuver to pull away from the wall.

The vehicle executed this plan in the presence of 2.4-Hz waves with a wave height of 2 cm, the same sea state as the experiment in Section 3.9. The waves traveled from left to right in Figure 5-27. The vehicle's trajectory is plotted in the figure. Note that the trajectory mostly falls within one standard deviation of the predicted mean path.

The environmental parameter estimates after executing this motion plan are $\hat{f}_U = -1.1 \times 10^{-3}$ and $\hat{f}_V = -5.3 \times 10^{-4}$. It is not guaranteed that these parameters truly reflect the mean environmental force; in fact, the nonlinear wave drift force ought to

result in a positive value for f_U for these waves. Rather, these parameters indicate the mean drift that the vehicle will experience when following this motion plan and similar motion plans. This drift may be caused by other model error, thruster lag, sensor filters, or a number of other reasons. As such, we can use these parameters when finding similar motion plans as these effects are likely to also be similar.

5.5.4 Experiment Learning Two Parameters

In this batch of experiments we learn two parameters at once: the the yaw drag a_{33} and the yaw thrust gain b_3. The planner uses second-order quadrature to evaluate the collision risk due to modeling error. The environmental parameters are fixed to the values derived above. As in the earlier batch of experiments, we perform the same mission four times with the same procedure described at the beginning of this section. The planning task is shown in Figure 5-28; it is similar to the previous example.

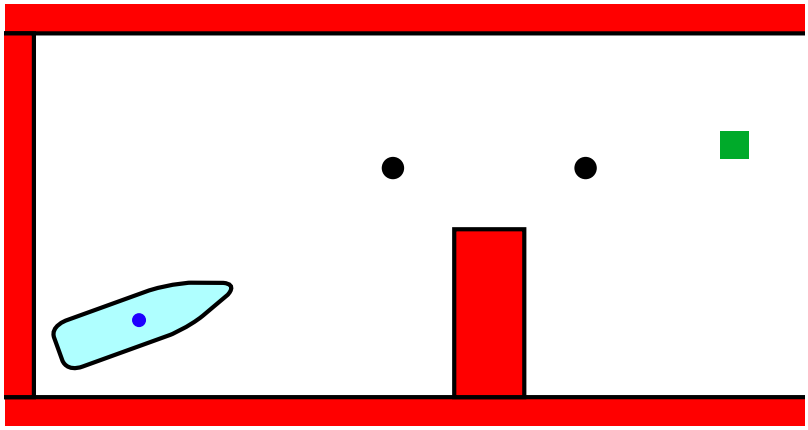

Figure 5-28: The planning task is to drive from the indicated start position to the green square. The waypoints available to the planner are shown as black circles.

For each parameter the *a priori* standard deviation is on the same order of magnitude as the *a priori* estimate. The initial estimates and standard deviations are shown in Table 5.4.

Table 5.4: Summary of Missions for Figure 5-28.

Mission	Motion Plan	Param.	$\hat{\beta}_0$	$\sqrt{\text{var}(\beta)_0}$	$\hat{\beta}_N$	$\sqrt{\text{var}(\beta)_N}$
1	`eacbcw`	a_{33}	-0.105	0.5	-0.281	0.268
	T = 31.5 sec	b_3	-0.0527	0.03	-0.0432	0.0179
	$P_{hit} = 4.1\%$					
2	`eawcaw`	a_{33}	-0.281	0.268	-0.538	0.228
	T = 30.8 sec	b_3	-0.0432	0.0179	-0.0348	0.0146
	$P_{hit} = 8.0\%$					
3	`eacbcaw`	a_{33}	-0.538	0.228	-0.333	0.203
	T = 31.9 sec	b_3	-0.0348	0.0146	-0.0237	0.0120
	$P_{hit} = 18.3\%$					
	Reduced thrust					
4	`eacbcaw`	a_{33}	-0.333	0.203	-0.262	0.181
	T = 31.9 sec	b_3	-0.0237	0.0120	-0.0101	0.0097
	$P_{hit} = 29.2\%$					

Mission 1

To learn the parameters before driving near the obstacles, the planner chooses the motion plan `eacbcw`, which has a duration of 31.5 seconds and a predicted collision probability of 4.1%. This plan is shown in Figure 5-29. Note the large spread of the quadrature trajectories after the turning maneuvers. This plan was executed by the vehicle in the same sea state as described in the previous section. The resulting trajectory is plotted in the figure. The parameter estimates at the end of the mission are shown in Table 5.4. The yaw drag parameter estimate increases by a factor of three, and the yaw thrust gain reduces slightly. These changes mean that the vehicle experiences more sideslip during turns than predicted by the *a priori* model.

Mission 2

After the parameter uncertainty was reduced in Mission 1, the planner chooses a more direct route to the goal. The motion plan `eawcaw` has a duration of 30.8 seconds and a predicted collision probability of 8.0%. This plan is shown in Figure 5-30 along with the actual vehicle trajectory. The parameter estimates are shown in Table 5.4.

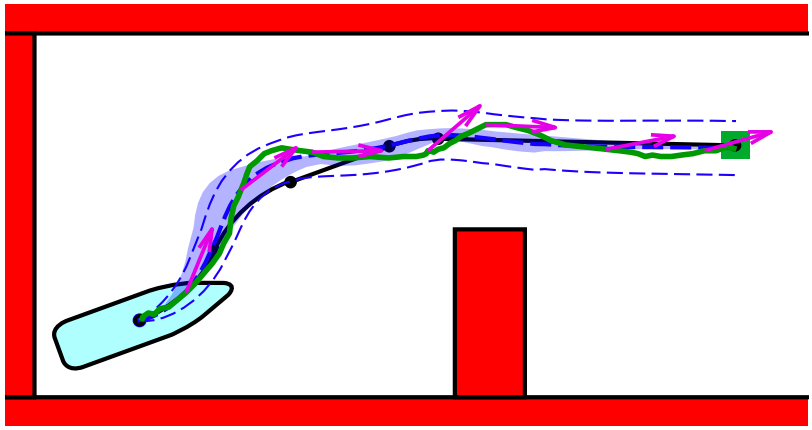

Figure 5-29: (Mission 1) The motion plan `eacbcw` exposes the vehicle yaw data early in the mission so that the controller performance is improved.

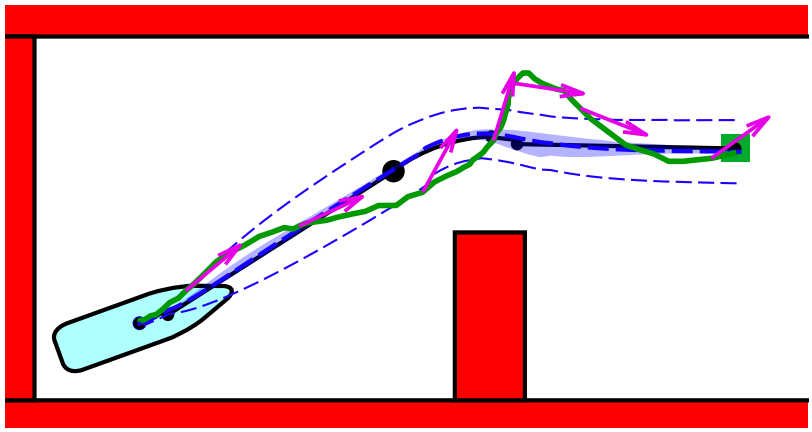

Figure 5-30: (Mission 2) The motion plan `eawcaw` drives directly to the goal.

Mission 3

Next we change the vehicle dynamics by reducing the thrust by 25%. Using the posterior variance estimates from Mission 2, the optimal motion plan is slightly different: now it is `eacbcaw`, with a duration of 31.9 seconds and a predicted collision probability of 18.3%. The yaw thrust gain estimate reduces by 32%, but the prediction error based on the final parameter estimate is not enough to increase the posterior variance. In other words, the learning algorithm has enough trust in the new parameter estimate that it continues to increase its confidence. The new estimates are shown in Table 5.4. The motion plan and the measured trajectory are shown in Figure 5-31.

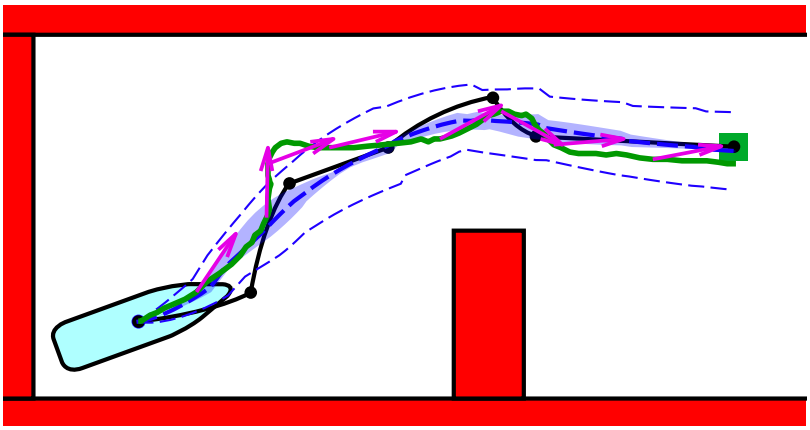

Figure 5-31: (Mission 3) The thrust gain has been reduced by 25% while following the motion plan `eacbcaw`.

Mission 4

The new parameter estimates reflect the fact that the vehicle has poor turning performance with the reduced thrust gain. The planner chooses the same motion plan `eacbcaw` as shown in Figure 5-32. Because the thrust is lower, it takes longer for the vehicle to get up to speed and the error variance increases. This motion plan has the same duration of 31.9 seconds but a larger predicted collision probability, 29.2%, due to the larger error variance. The measured trajectory is shown in the figure.

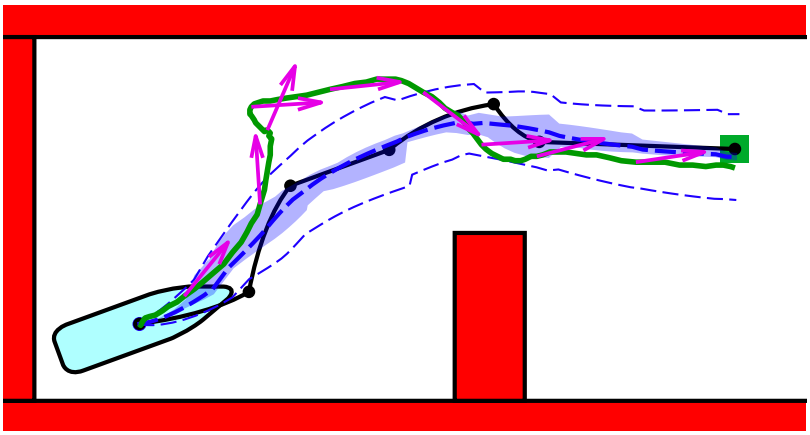

Figure 5-32: (Mission 4) The errors are larger when following this plan than when following the identical plan in Figure 5-31 because the vehicle's speed is lower.

5.6 Summary

In this chapter we have brought the various pieces of the motion planning and model learning problems together in a single integrated strategy. The resulting algorithm is based on the robust motion planner described in Chapter 3 and it uses the predictions of the parameter convergence from Chapter 4 to relate active learning strategies to the planner's cost function. The final algorithm has very few parameters that need to be tuned by the operator and it is robust to changes in the true vehicle model parameters.

We have provided simulations of this algorithm applied to a simple 2D grid-based robot and simulations and experiments of the algorithm applied to a marine surface vessel, which is representative of a large class of planar holonomic vehicles. The experimental results show how the learning algorithm gains confidence in the parameter estimates by executing the motion plans, and how the planner uses that confidence to find motion plans that will be successful in the presence of model uncertainty. High-level feedback is achieved by adjusting the parameter covariance estimates based on the prediction error during the execution of the motion plans. In this way the algorithm does not become too confident in incorrect parameter estimates.

In Chapter 6 we summarize the novel contributions in this thesis and we highlight fruitful areas of future work.

Chapter 6

Conclusions

In Chapter 5 we presented the integrated motion planning and model learning algorithm that ties together the various components of this thesis. In this chapter we summarize the novel contributions in this thesis and we suggest avenues of future research. While much of this thesis is devoted to specific calculations applicable to a planar holonomic vehicle such as a marine surface vessel, the overall algorithm can be applied to a wide variety of mobile robots. Much of the future work involves extending the specific components of the full algorithm to new types of vehicles. In the last section we finish with some concluding remarks.

6.1 Novel Contributions

This thesis includes new contributions to several disparate research areas in the construction of the integrated algorithm, which is itself a novel high-level strategy. The various specific contributions are summarized below.

6.1.1 Robust Motion Planner

The first new contribution in this thesis is the robust motion planning algorithm. Building on a standard cost-based motion planner to find the shortest sequence of actions to bring a mobile robot through a cluttered environment to a goal configura-

tion, this robust motion planner discourages plans that have a low chance of success. Success is defined by the avoidance of obstacles, so in a stochastic context the planner incorporates the probability of collisions into the cost function.

Evaluating the collision probability is the key step when determining the value of a motion plan. This probability could be computed using a Monte Carlo simulation, but that approach requires a great computational effort to achieve an accurate and reliable result. Instead, we develop fast and efficient algorithms to approximate the collision probability for each motion plan based on the estimated dynamic model for the system, the noise models, and the obstacle locations. These various algorithms are described below.

6.1.2 Error Variance Predictions

For a planar holonomic vehicle driving under closed-loop velocity and/or position control in the presence of stochastic disturbances, under certain conditions it is possible to express the variance of the state error as a linear variance evolution equation, which is a version of the matrix Riccati equation. This equation can be solved analytically to predict the error covariance matrix at any moment in the execution of the motion plan. In a parallel problem, we can predict the mean error due to transient effects and underactuation by solving a first-order differential equation using the matrix exponential. These approximations are valid as long as the path-following error (the heading error in particular) is not too large, i.e. the small-angle approximation applies.

One feature of the prediction of the mean and variance of the state error is that it can account for the effects of a controller designed around incorrect parameter estimates. This effect is particularly strong in the feedforward velocity term. This prediction on its own is not particularly useful because if the model error is known then the true parameter values are known (and, in that case, it would be possible to design a controller around the correct values). However, the calculation is the foundation of the numerical quadrature technique described below.

6.1.3 Model-Based Collision Probability Predictions

Given the distribution of state errors around the reference trajectory and the locations of the obstacles, it is not a simple task to evaluate the collision probability. The continuous dynamic model of the system leads to state correlations through time, so the collision probability evaluation is an infinite-dimensional conditional probability problem. We related this problem to the particle absorption problem from statistical physics, and we solved the problem for the case of driving past a single continuous obstacle. To deal with multiple obstacles and non-continuous sampling of the error statistics, it was necessary to make a rough approximation in which the obstacle locations are sampled at a certain rate. This sampling rate is determined by the vehicle dynamics, so the collision probability prediction improves as the model estimate improves. While the predicted collision probability for a motion plan does not exactly match the result of a Monte Carlo simulation, the prediction is a useful tool for comparing different motion plans.

6.1.4 Model Uncertainty Effects through Numerical Quadrature

Model uncertainty affects the collision probability in a fundamentally different way than disturbances; instead of being a continuous random process, model uncertainty makes the collision probability a deterministic function of a random variable. Numerical quadrature techniques are ideal for this type of problem. The output PDF is approximated by sampling the function (the collision probability) at specific input points (vectors in the parameter space) and combining the results with a weighting rule. Each point in the parameter space corresponds to a vector of parameter error. As described above, the prediction of the mean and variance of the state error can be calculated for a specific parameter error vector. The net effect of random parameter error is approximated by the quadrature rule.

6.1.5 Parameter Convergence Predictions

The learning algorithm used in this thesis is least squares regression, which is a standard algorithm in parameter estimation. If the sensor noise model and the input data to the learning algorithm are known in advance, then it is possible to predict the parameter covariance matrix before any output data is collected. However, this problem is more complicated when the input data is stochastic rather than deterministic, and it is even more difficult when the input data has correlations through time. Because the learning algorithm takes as an input the vehicle's velocity and control values, and those states are the output of a system excited by a stochastic process, the learning data is indeed correlated and stochastic. Using a Taylor series expansion applied to matrix data, we developed a second-order prediction of the parameter covariance matrix evolution.

6.1.6 Integrated Motion Planning and Model Learning Algorithm

The various predictions outlined above are combined in the integrated motion planning and model learning algorithm. The key link is made when the parameter covariance at the beginning of each maneuver in the motion plan is used to define the quadrature point locations (parameter error vectors) used by the numerical quadrature rule. In this way, the state excitation resulting from different maneuvers is reflected in the parameter convergence, the collision probability, and ultimately the cost of the motion plan. Active learning strategies appear naturally out of this process.

There is a high-level feedback that occurs between missions when the prediction error is used to adjust the *a priori* parameter covariance for the next mission. This feedback corrects for noise model errors and changes to the true parameter values. Over the course of many missions, the dynamic model and the collision probability calculations become more accurate.

The integrated motion planning and model learning algorithm can apply to any

mobile robot system; the only necessary ingredients are a prediction of the parameter convergence for each motion plan, and a way to relate that parameter convergence to the planner's cost function. With those two ingredients, any robotic system can be made to plan ahead with active learning strategies.

6.2 Future Research

This thesis leads to many avenues of future research. The integrated motion planning and model learning algorithm is very general, while the calculations of the error variance and collision probability for the planar holonomic vehicle are very specific. Consequently, the main thrust of future work should be to extend the calculations to more general classes of mobile robots. Some specific directions are suggested below.

6.2.1 Extensions to 6 DOF Vehicles, Non-Holonomic Robots, and Non-Gaussian Disturbances

The overall integrated algorithm is general enough that it can be applied to a wide range of mobile robots, as discussed below. However, some of the predictions derived in this thesis are limited to 3 DOF planar holonomic vehicles. The coordinate transformation used in Section 3.6 results in a constant state transition matrix $\mathbf{A_0}$ for a planar vehicle, but in 6 DOF the corresponding matrix is not constant. The calculation of the mean error using the matrix exponential and the error variance using the Riccati equation require a constant $\mathbf{A_0}$ matrix, so these approaches do not work for the 6 DOF vehicle. This problem must be explored further to find a way to predict the error statistics analytically for a more general vehicle.

The sampled approach to predicting the collision probability can be applied to a 6 DOF vehicle as well as a planar vehicle, but the calculation of the optimal sample time would need to be adapted to a 6 DOF vehicle operating in a 3D environment. A simple approach would be to map the 6 DOF problem to a one-dimension-plus-time (1D+T) problem, just as the 3 DOF planar holonomic vehicle problem is mapped to

a 1D+T problem in Section 3.7.

This work is focused on holonomic vehicles, meaning that there are no velocity-based constraints. A car-like robot is the most common non-holonomic vehicle: without wheel slip, the turn rate and sideways motion are dependent on one another. An obvious extension of the work in this thesis would be to extend the predictions to non-holonomic mobile robots. Such an extension would bring the error distribution predictions and collision probability predictions to unmanned ground vehicles, rovers, and other wheeled robots.

Finally, the analytic predictions of the error statistics, the collision probability and the parameter convergence rely on Gaussian models for the process noise, sensor noise and uncertainty. A useful extension of this thesis would be to derive predictions for other noise models, including bounded disturbances and parameter error distributions with semi-infinite support (a Rayleigh distribution, for example).

6.2.2 Adaptive Sparse Grid Quadrature

For simplicity, the planar holonomic vehicle examples presented in Chapter 5 used second-order quadrature. The quadrature scheme was defined using a full tensor grid, meaning that the number of quadrature points in each dimension was the same. For higher-order quadratures this approach leads to a very large number of quadrature points, and the speed of the planning algorithm will be slow as a result. If the full tensor grid is replaced by an adaptive sparse grid, then similar accuracy can be achieved with fewer quadrature points. The adaptivity places the quadrature points where they are needed most. However, because the quadrature points are fixed in a normalized orthogonal space for each motion plan, the adaptivity would have to take place over the entire motion plan rather than on a maneuver-by-maneuver basis. In other words, each maneuver requires the same quadrature point locations in the normalized space.

6.2.3 Generalize the Integrated Algorithm

As discussed above, the integrated motion planning and model learning algorithm can be applied to any mobile robot system. The simple 2D robot simulation and the marine surface vessel are only two examples of the applicability of the algorithm. For the simple 2D robot the predictions were straightforward, but for the marine surface vessel and other vehicles with complicated dynamics the predictions are more difficult to obtain. However, if those predictions are available without resorting to Monte Carlo simulations then the algorithm can be applied to a wide range of vehicles. In fact, if precision, repeatability and computational cost are not an issue then Monte Carlo simulations could indeed be used in the integrated algorithm as well.

6.2.4 Investigate Emergent Practicing Behavior

Section 5.2.8 contained a brief discussion of the conditions under which a robot will choose to practice maneuvers. In the discretized planning framework used in this thesis, the practicing behavior is quantized by the discrete maneuvers. If a continuous planning approach was used instead, then one would expect to observe a continuous gradient of practicing behavior; in other words, the robot would practice just enough to exactly minimize the overall cost function. Equation (5.26) offers a glimpse of what that gradient might look like for general mobile robots.

Next, POMDPs should be applied to the model learning problem to see if similar practicing behaviors emerge. POMDPs result in a policy, not specific motion plans, so they may offer new insights into the motivations for practicing. However, it is possible that the motion planning problem using POMDPs is too computationally expensive to be practical for this problem.

6.3 Concluding Remarks

The thesis goal stated at the beginning of Chapter 1 has a wide scope, but through the construction of the integrated motion planning and model learning algorithm and

the careful derivation of the various components, we have achieved the goal. Using this algorithm a marine surface vessel or any mobile robot with similar dynamics can find feasible and safe motion plans through a known environment. If the vehicle model changes due to damage, wear, or configuration changes, then the planner will automatically take steps to learn about the new model while executing the next mission.

After initially defining the speed controller setpoints, the LQR control cost and an approximate *a priori* model, the human operator of the system only needs to provide the robot with maps and destinations. As more and more mobile robots are used in practical applications, this paradigm will be very useful. An entire fleet of autonomous vehicles could be programmed in a factory and delivered to customers; then each vehicle could be configured by the customer for his or her specific application, and the vehicle would teach itself about the dynamic model corresponding to that configuration. In this way, the end user does not need to know anything about planning algorithms, learning algorithms, or control theory. If designers of robotic systems continue with this strategy, then autonomous vehicles will gain a greater acceptance in the field.

Appendix A

Autonomous Surface Vessel

A.1 General Description

The autonomous surface vessel used in the experiments in this thesis is a 1.25-meter wooden model of an icebreaking ship outfitted with an electric propulsion system, a computer system, and digital sensors. The vessel is shown in Figure A-1. The various subsystems are described in the following sections.

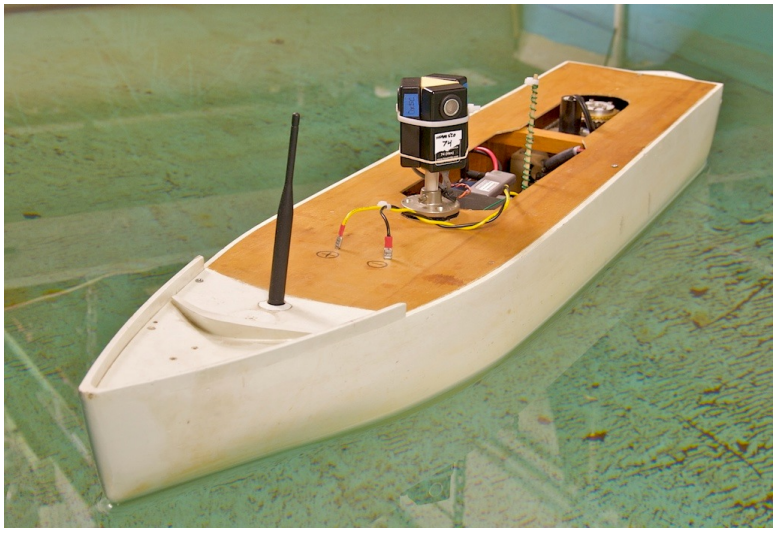

Figure A-1: The autonomous surface vessel used in the experiments. It is a wooden model of an icebreaker. From left to right, the four masts are (1) the radio modem used for the ultrasonic positioning system, (2) the ultrasonic beacons, (3, partially hidden) the computer's wireless router, and (4) the radio control antenna.

A.2 Propulsion

The vehicle is propelled by a single azimuthing thruster, a West Marine 2140W bilge pump motor rated at 410 gallons per hour drawing 1.2 amps at 13.6V. Its propeller is a 2-bladed plastic model aircraft propeller. The thruster is mounted to the bottom of a vertical stainless steel post that turns inside a plastic tube passing through the hull. The unit is shown in Figure A-2.

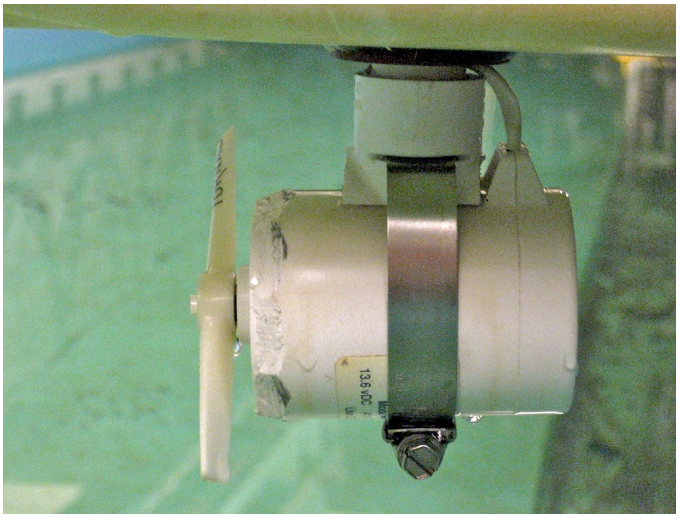

Figure A-2: The azimuthing thruster.

A.2.1 Speed Controller

The propulsion motor is controlled with a Vantec RFR825 pulse width modulated (PWM) speed controller. Its input is a standard servo PWM control signal, and the output is a 12-volt PWM whose duty cycle is proportional to the throttle setting of the control input. The power source is a 12V lead acid battery that also powers the CPU and sensors.

A.2.2 Azimuthing Servo

The vertical post of the thruster connects to a Tonegawa SSPS-105 servo through a chain drive. The servo runs off the 12V lead acid battery that powers the CPU and

sensors. It has a maximum rotation rate of 95°/sec with an angular sweep of ±180°. The maximum load is 27.5 in-lb. The servo has an internal position controller; a standard servo PWM control signal sets the reference position and the servo converges to that position without overshoot. The servo connects to the top of the thruster's vertical post with a chain drive.

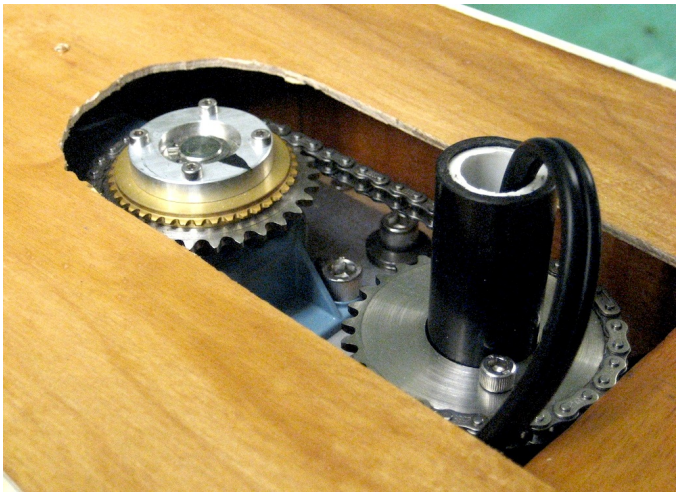

Figure A-3: The azimuthing servo (left) connected to the vertical post of the thruster with a chain drive.

A.3 Sensors

A.3.1 Ultrasonic Positioning

The ultrasonic positioning system consists of five Hexamite HX900ASIO modules: two onshore *pilots* and three onboard *beacons*. The pilots simultaneously send a 40 kHz pulse out to the vehicle; as soon as any of the beacons on the vehicle hears one of the pulses, it sends a pulse back. The total time of flight measured by the pilots determines the distance from each pilot to the beacon, and the beacon position is computed by triangulation. The rated maximum range is 16 meters, but the maximum useful range is around 10 meters. The update rate is around 1.75 Hz. A pair of radio modems are used to send the position data back to the vehicle.

The two pilots are connected to each other by a two-meter phone cable. This cable provides serial communication between the two pilots and power to the second pilot. Through a Y-adapter the phone cable continues on to a radio modem. The radio modem's power supply feeds into the phone cable to power both HX900 units. The three pilots are connected to the 12V bus of the onboard computer. They are mounted to a short mast; the three units point towards the stern, 60° to port, and 60° to starboard. This mast is shown in Figure A-4.

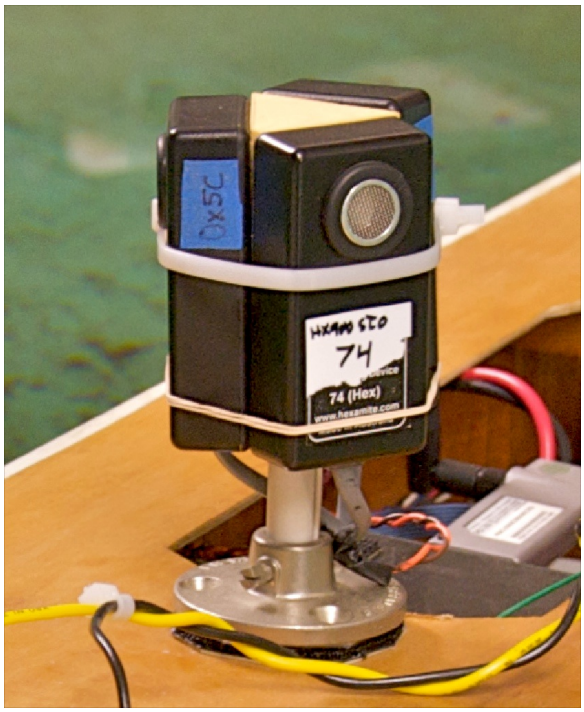

Figure A-4: Three ultrasonic positioning beacons are mounted on a short mast on the deck of the vessel. They each have an effective angular coverage of 120°, so these three units together act as an omnidirectional beacon.

The radio modems are Max Stream 9XStream 900 MHz 9600 baud modules. The onshore modem is powered by a 12V AC/DC supply and the onboard modem is powered by the onboard computer's 12V bus. The onboard modem is mounted in the bow with its antenna protruding up through the deck.

A.3.2 Compass

The compass is a PNI Corporation TCM2.6 tilt-compensated magnetometer. It uses an integrated tilt sensor to correct for the orientation of the magnetometer. The compass sends the heading angle, the roll angle and the pitch angle to the computer at 8 Hz. The unit is mounted in the bow of the vessel, far from the magnets in the propulsion motor.

A.4 Computer System

The computer system is a PC/104 stack with a CPU board, a counter/timer board, a power board, and a flash disk module. The CPU is a Diamond Systems Athena PC/104 computer, model ATH660-128, with a 660 MHz VIA Eden processor (equivalent to a Pentium III) and 128 MB of RAM. It has four serial ports and four USB 1.1 ports. There are sixteen 16-bit analog inputs, four 12-bit analog outputs, and twenty-four digital I/O channels. The system uses a solid state hard drive, the Emphase FDM4400 2 GB Flash Disk Module. The counter-timer PC/104 board is a Diamond Systems GPIO-MM. It has 48 digital I/O channels and ten 16-bit counter/timers, five each on 2 AM9513A-equivalent chips. Power is delivered to the computer from a 7 AH 12V sealed lead-acid battery that connects to a Tri-M Engineering HE104 PC/104 power supply board. The Athena CPU, GPIO-MM, HE-104, breakout boards, and Emphase Flash Disk Module are housed in a Diamond Systems 5-inch Pandora enclosure, stacked in that order. The front panel of the enclosure has the computer connectors and the back panel is perforated to allow air circulation.

A.5 Remote Control

A radio control system is used to drive the boat under manual control and switch between different modes of operation while the boat is underway. The transmitter is a 6-channel 75 MHz FM Airtronics VG600, shown in Figure A-5, and the receiver is a 7-channel 75 MHz FM Airtronics 92875. The transmitter has two 2-channel joysticks,

a two-position switch, and a three-position switch. The left stick controls the throttle, and the right stick controls the azimuth angle. The three-position switch is used to select the mode of operation of the vehicle and to start the planner.

Figure A-5: The radio control transmitter used to manually drive the vehicle. The switch on the upper right corner is used to start the planner and activate automatic control.

A.6 Dynamic Model

The following parameter vector was computed from a series of simple system identification tests using least squares regression.

$$\boldsymbol{\beta}_{sysid} = \begin{bmatrix} a_{11} \\ b_1 \\ a_{22} \\ a_{23} \\ b_2 \\ a_{32} \\ a_{33} \\ b_3 \\ f_U \\ f_V \end{bmatrix} = \begin{bmatrix} -0.03716 \\ 0.04247 \\ -0.08013 \\ 0.006497 \\ 0.002850 \\ 0.07146 \\ -0.1047 \\ -0.05270 \\ 0 \\ 0 \end{bmatrix} \tag{A.1}$$

Using the parameters in $\boldsymbol{\beta}_{sysid}$, the dynamic model for the icebreaker is:

$$\dot{\boldsymbol{\nu}} = \underbrace{\begin{bmatrix} -0.03716 & 0 & 0 \\ 0 & -0.08013 & 0.006497 \\ 0 & 0.07146 & -0.1047 \end{bmatrix}}_{\mathbf{a}(\boldsymbol{\nu})} \boldsymbol{\nu} + \underbrace{\begin{bmatrix} 0.04247 & 0 \\ 0 & 0.002850 \\ 0 & -0.05270 \end{bmatrix}}_{\mathbf{b}} + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}}_{\mathbf{f}_{\boldsymbol{\nu}}(\psi)} + \mathbf{w}_{\boldsymbol{\nu}} \tag{A.2}$$

The process noise vector $\mathbf{w}_{\boldsymbol{\nu}}$ is sampled from a zero-mean diagonal multinormal distribution with a covariance matrix $\mathbf{W}_{\boldsymbol{\nu}}$ shown below:

$$\mathbf{W}_{\boldsymbol{\nu}} = \begin{bmatrix} 2 \times 10^{-5} & 0 & 0 \\ 0 & 8 \times 10^{-4} & 0 \\ 0 & 0 & 3 \times 10^{-3} \end{bmatrix} \tag{A.3}$$

The process noise was evaluated by measuring the variance of the surge, sway and yaw accelerations while the vehicle encountered 2.4-Hz waves with a 2-cm wave height and a wavelength of 27 cm.

The learning algorithm must have an estimate of the noise $\mathbf{w_s}$ that affects the measurement of the accelerations $\dot{u}$, $\dot{v}$ and $\dot{r}$. This noise arises more from the relatively slow sample rates of the sensors than sensor noise itself; three position or orientation data points are needed for one acceleration measurement using a finite difference approach, and it is difficult to sync the resulting acceleration measurement to the input variables of velocity and thrust. Online learning algorithms do not have the benefit of post-processing and bi-directional smoothing, so we must account for the measurement lag with a sensor noise model. The noise levels were measured by collecting position data while driving the vehicle in smooth water, then calculating the acceleration through numerical differentiation, removing low-frequency effects, and taking the variance of the resulting data. Using this method, the sensor noise covariance matrix $\mathbf{W_s}$ is:

$$\mathbf{W_s} = \begin{bmatrix} 0.0035 & 0 & 0 \\ 0 & 0.019 & 0 \\ 0 & 0 & 0.11 \end{bmatrix} \tag{A.4}$$

A.7 Controller

The vehicle uses an LQR controller to follow the reference trajectory. This controller has two forms, depending on whether or not position feedback will be applied. For all forward maneuvers, the state cost matrix is:

$$\mathbf{Q}_{lqr} = \text{diag}([1,\ 1,\ 1,\ 1,\ 2,\ 1]) \tag{A.5}$$

For all other maneuvers, the state cost matrix is shown below. There is no cost on position error, and the cost on velocity errors is increased to keep the overall bandwidth approximately the same.

$$\mathbf{Q}_{lqr} = \text{diag}([2,\ 2,\ 2,\ 0,\ 0,\ 0]) \tag{A.6}$$

For all maneuvers, the control cost matrix is:

$$\mathbf{R}_{lqr} = \begin{bmatrix} 0.05 & 0 \\ 0 & 0.05 \end{bmatrix} \tag{A.7}$$

The nominal forward speed for the vehicle is 0.15 m/sec and the nominal yaw rate is 9°/sec.

Appendix B

Numerical Procedures

B.1 Continuous Algebraic Riccati Equation

The continuous algebraic Riccati equation is used to design an LQR controller. For a square system matrix A, input matrix B, control cost matrix R_{lqr}, and state cost matrix Q_{lqr}, the feedback matrix is ${R_{lqr}}^{-1}B^TP$ where the symmetric positive-definite matrix P is the solution to the following equation:

$$A^TP + PA - PB{R_{lqr}}^{-1}B^TP + Q_{lqr} = 0 \tag{B.1}$$

This equation can be solved in Matlab with the command `P = care(A,B,Qlqr,Rlqr)`. The solution is outlined in [2]. First a new matrix M is constructed from the original matrices:

$$M = \begin{bmatrix} A & -B{R_{lqr}}^{-1}B^T \\ -Q_{lqr} & -A^T \end{bmatrix} \tag{B.2}$$

Next we find the Schur decomposition of M, that is, the orthogonal matrix U and the upper triangular matrix S so that $S = U^TMU$. However, the Schur decomposition is not unique, and we need the particular U and S for which the eigenvalues in the upper-left quadrant of S are all in the left half-plane. Once the proper decomposition

is found, P is computed as follows:

$$\begin{aligned} U &= \begin{bmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{bmatrix} \\ P &= U_{21}{U_{11}}^{-1} \end{aligned} \tag{B.3}$$

To find the appropriate U, we note that the Schur decomposition can be found using the following procedure:

$$\begin{aligned} D &= V^{-1}MV \qquad \texttt{[V,D] = eig(M)} \\ V &= UR \qquad\quad\ \texttt{[U,R] = qr(V)} \\ D &= R^{-1}U^TMUR \\ RDR^{-1} &= U^TMU \end{aligned} \tag{B.4}$$

RDR^{-1} is an upper-triangular matrix and U is orthogonal, so (B.4) represents a Schur decompositon. The positions of the eigenvalues along the diagonal of RDR^{-1} depend on their positions in D. However, because D is strictly diagonal, it is easy to reorder the eigenvalues in D (and the corresponding eigenvectors in V). The procedure is listed below. Note that n is the size of A.

```
[V,D] = eig(M);
Ddiag = diag(D);
[Dsort, isort] = sort(real(Ddiag));
Ddiag = Ddiag(isort);
D = diag(Ddiag);
V = V(:, isort);
[U,R] = qr(V);
U1 = U(1:n, 1:n);
U2 = U(n+1:2*n, 1:n);
P = U2 * inv(U1);
```

This procedure can be performed in Java because both matrix packages, JAMA

and Jampack, have QR decomposition routines.

B.2 Discrete-time Lyapunov Equation

The discrete-time Lyapunov equation is used in the correlated prediction of $E[R^{-1}]$ (4.92) in the parameter convergence problem. The problem is to find the matrix X that satisfies:

$$X - AXA = C \tag{B.5}$$

If A is invertible, then (B.5) can be rewritten as follows:

$$A^{-1}X - XA = A^{-1}C \tag{B.6}$$

This form of the equation resembles the non-symmetric continuous algebraic Riccati equation. We can solve the equation be creating an M matrix similar to (B.2):

$$M = \begin{bmatrix} A & 0 \\ -A^{-1}C & A^{-1} \end{bmatrix} \tag{B.7}$$

Once M is constructed, the solution X is computed using the same Schur decomposition procedure shown in Section B.1. However, this procedure does not work when A is singular because A^{-1} does not exist. In the application (4.92) A is often singular due to the construction of the problem. To deal with this, we first take the eigenvalue decomposition of A, `[V,D] = eig(A)`, so that $A = VDV^{-1}$. Plugging this into the original problem we have the following:

$$\begin{aligned} X - VDV^{-1}XVDV^{-1} &= C \\ V^{-1}XV - DV^{-1}XVD &= V^{-1}CV \qquad \text{(B.8)} \\ \longrightarrow \quad X_V - DX_VD &= C_V \qquad \text{(B.9)} \end{aligned}$$

When A is singular, D only has nonzero values on some of the diagonal elements. We assume that D (and V) have been arranged so that all of the nonzero values

come first on the diagonal and all the zero elements on the diagonal are in the lower right corner. If there are m nonzero diagonal elements (equivalently, the rank of A is m), then the second term of (B.9) has zeros everywhere except for an $m \times m$ submatrix in the upper left quadrant. In the other three quadrants, $X_V = C_V$. The upper left quadrant of X_V is the solution outlined above for the invertible case if M is constructed using $D(1:m, 1:m)$ instead of A and $C_V(1:m, 1:m)$ instead of C. Once all four quadrants of X_V have been formed, then the final solution is extracted using $X = VX_VV^{-1}$.

B.3 Continuous-time and Discrete-time State Space Models

The path-following error statistics for the vehicle are computed using a continuous-time state space model of the vehicle's dynamics. However, the learning rate predictions use a discrete-time model. The following table shows the conversion of a simple continuous-time state space model (B.10) to a discrete-time model (B.11) with a sample time Δt. The zero-mean Gaussian process noise vector w_c has a covariance matrix W_c. The covariance of w_k is W.

$$\dot{x}(t) = a_c x(t) + b_c + w_c(t) \tag{B.10}$$

$$x_{k+1} = ax_k + b + w_k \tag{B.11}$$

Table B.1: Conversion from continuous-time to discrete-time state space system matrices.

Continuous	Discrete
a_c	$a = e^{a_c \Delta t}$
b_c	$b = (a - I){a_c}^{-1} b_c$
W_c	$W = W_c \Delta t$

Bibliography

[1] A. Aguiar and A. Pascoal. Modeling and control of an autonomous underwater shuttle for the transport of benthic laboratories. In *OCEANS '97*, volume 2, pages 888–895, 1997.

[2] W. F. Arnold and A. J. Laub. Generalized eigenproblem algorithms and software for algebraic riccati equations. *Proceedings of the IEEE*, 72(12):1746–1754, 1984.

[3] E. Bach, S. Coppersmith, M. P. Goldschen, R. Joynt, and J. Watrous. One-dimensional quantum walks with absorbing boundaries. *Journal of Computer and System Sciences*, 69(4):562–592, 2004.

[4] J. Barraquand and P. Ferbach. Motion planning with uncertainty: the information space approach. In P. Ferbach, editor, *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 1341–1348, 1995.

[5] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

[6] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. Pappas. Symbolic planning and control of robot motion. *IEEE Robotics and Automation Magazine*, pages 61–70, March 2007.

[7] L. Blackmore. A probabilistic particle control approach to optimal, robust predictive control. *Proceedings of the AIAA Guidance, Navigation and Control Conference*, 2006.

[8] L. Blackmore and B. Williams. Finite horizon control design for optimal discrimination between several models. In *45th IEEE Conference on Decision and Control*, pages 1147–1152, 2006.

[9] J. C. Bongard and H. Lipson. Automated robot function recovery after unanticipated failure or environmental change using a minimum of hardware trials. In *NASA/DoD Conference on Evolvable Hardware*, pages 169–176, 2004.

[10] R. W. Brockett. Asymptotic stability and feedback stabilization. *Differential Geometric Control Theory*, 27:181–191, 1983.

[11] M. Brodie, I. Rish, S. Ma, and N. Odintsova. Active probing strategies for problem diagnosis in distributed systems. In *International Joint Conference on Artificial Intelligence*, volume 18, pages 1337–1338, 2003.

[12] T. W. Burkhardt. Absorption of a randomly accelerated particle: Recent results for partially absorbing and inelastic boundaries. *Physica A: Statistical Mechanics and its Applications*, 306:107–116, 2002.

[13] B. Burns and O. Brock. Sampling-based motion planning using predictive models. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 3120–3125, 2005.

[14] G. Buskey, J. Roberts, and G. Wyeth. A helicopter named dolly-behavioural cloning for autonomous helicopter. In *Proceedings of the Australasian Conference on Robotics and Automation*, Brisbane, Australia, 2003.

[15] S. Dasgupta and Y-F. Huang. Asymptotically convergent modified recursive least-squares with data-dependent updating and forgetting factor for systems with bounded noise. *IEEE Transactions on Information Theory*, 33(3):383–392, 1987.

[16] P. J. Davis and P. Rabinowitz. *Methods of Numerical Integration*. Academic Press, New York, 1975.

[17] R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of A*. *Journal of the ACM*, 32(3):505–536, 1985.

[18] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[19] M. W. M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localization and map building (SLAM) problem. *IEEE Transactions on Robotics and Automation*, 17(3):229–241, 2001.

[20] K. Djouani and Y. Hamam. Ship optimal path planning and artifical neural nets for berthing. In *OCEANS '94*, volume 1, pages 785–790, 1994.

[21] K. D. Do and J. Pan. Robust path-following of underactuated ships: Theory and experiments on a model ship. *Ocean Engineering*, 33(10):1354–1372, 2006.

[22] L. Elden. Algorithms for the regularization of ill-conditioned least squares problems. *BIT Numerical Mathematics*, 17(2):134–145, 1977.

[23] M. Erdmann. Using backprojections for fine motion planning with uncertainty. In *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 549–554, 1985.

[24] H. J. S. Feder, J. J. Leonard, and C. M. Smith. Adaptive mobile robot navigation and mapping. *International Journal of Robotics Research*, 18(7):650–668, 1999.

[25] A. Felner, R. Stern, A. Ben-Yair, S. Kraus, and N. Netanyahu. PHA*: Finding the shortest path with A* in an unknown physical environment. *Journal of Artificial Intelligence Research*, 21:631–670, 2004.

[26] T. R. Fortescue, L. S. Kershenbaum, and B. E. Ydstie. Implementation of self-tuning regulators with variable forgetting factors. *Automatica*, 17(6):831–835, 1981.

[27] T. I. Fossen. *Marine Control Systems: Guidance, Navigation, and Control of Ships, Rigs and Underwater Vehicles*. Marine Cybernetics, Trondheim, Norway, 2002.

[28] T. I. Fossen, M. Breivik, and R. Skjetne. Line-of-sight path following of under-actuated marine craft. In *Proceedings of the IFAC Conference on Maneuvering and Control of Marine Craft*, pages 244–249, Girona, Spain, 2003.

[29] E. Frazzoli. Maneuver-based motion planning and coordination for single and multiple UAV's. In *AIAA 1st Technical Conference and Workshop on Unmanned Aerospace Vehicles*, Portsmouth, Virginia, 2002.

[30] E. Frazzoli. Explicit solutions for optimal maneuver-based motion planning. In *42nd IEEE Conference on Decision and Control*, volume 4, pages 3372–3377, 2003.

[31] E. Frazzoli, M. A. Dahleh, and E. Feron. Robust hybrid control for autonomous vehicle motion planning. In *39th IEEE Conference on Decision and Control*, volume 1, pages 821–826, Sydney, Australia, 2000.

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

[33] E. Frazzoli, M. A. Dahleh, and E. Feron. Maneuver-based motion planning for nonlinear systems with symmetries. *IEEE Transactions on Robotics and Automation*, 21(6):1077–1091, 2005.

[34] G. Freiling. A survey of nonsymmetric Riccati equations. *Linear Algebra and its Applications*, 351-352:243–270, 2002.

[35] A. Gelb. *Applied Optimal Estimation*. MIT Press, Cambridge, MA, 1974.

[36] M. Greytak. *High Performance Path Following for Marine Vehicles Using Azimuthing Podded Propulsion*. Massachusetts Institute of Technology, Department of Mechanical Engineering, Cambridge, MA, 2006.

[37] M. Greytak and F. Hover. Robust motion planning for marine vehicle navigation. In *Proceedings of the 18th International Offshore and Polar Engineering Conference*, volume 2, pages 399–406, Vancouver, Canada, 2008.

[38] M. Greytak and F. Hover. Underactuated point stabilization using predictive models with application to marine vehicles. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3756–3761, Nice, France, 2008.

[39] M. Greytak and F. Hover. Analytic error variance predictions for planar vehicles. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation*, pages 471–476, Kobe, Japan, 2009.

[40] M. Greytak and F. Hover. Motion planning with an analytic risk cost for holonomic vehicles. In *Proceedings of the 2009 IEEE Conference on Decision and Control*, Shanghai, China, 2009.

[41] M. Greytak and F. Hover. Planning to learn: Integrating model learning into a trajectory planner for mobile robots. In *Proceedings of the 2009 IEEE International Conference on Information and Automation*, pages 18–23, Zhuhai, China, 2009.

[42] S. Harris. Steady, one-dimensional brownian motion with an absorbing boundary. *The Journal of Chemical Physics*, 75(6):3103–3106, 1981.

[43] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[44] S. Huang, N. M. Kwok, G. Dissanayake, Q. P. Ha, and G. Fang. Multi-step look-ahead trajectory planning in SLAM: Possibility and necessity. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 1091–1096, 2005.

[45] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

[46] F. Kaplan and P-Y. Oudeyer. Intrinsically motivated machines. In *50 Years of Artificial Intelligence*, pages 303–314. Springer, Heidelberg, 2007.

[47] S. Koenig and M. Likhachev. Improved fast replanning for robot navigation in unknown terrain. In *IEEE International Conference on Robotics and Automation*, pages 968–975, 2002.

[48] T. Kollar and N. Roy. Trajectory optimization using reinforcement learning for map exploration. *International Journal of Robotics Research*, 27(2):175–196, 2008.

[49] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report 98-11, Computer Science Dept., Iowa State University, 1998.

[50] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[51] S. M. LaValle and S. Hutchinson. An objective-based framework for motion planning under sensing and control uncertainties. *The International Journal of Robotics Research*, 17(1):19–42, 1998.

[52] S. M. LaValle and R. Sharma. A framework for motion planning in stochastic environments: Modeling and analysis. In *Proceedings of the 1995 IEEE International Conference on Robotics and Automation*, volume 3, pages 3063–3068, 1995.

[53] J. Leitner, A. Calise, and J. V. R. Prasad. Analysis of adaptive neural networks for helicopter flight control. *Journal of Guidance, Control, and Dynamics*, 20(5):972–979, 1997.

[54] J. J. Leonard and H. F. Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *Proceedings of the 1991 IEEE/RSJ International Workshop on Intelligent Robots and Systems.*, volume 3, pages 1442–1447, 1991.

[55] N. E. Leonard. Control synthesis and adaptation for an underactuated autonomous underwater vehicle. *IEEE Journal of Oceanic Engineering*, 20(3):211–220, 1995.

[56] M. Likhachev, G. Gordon, and S. B. Thrun. ARA*: Anytime A* with provable bound on sub-optimality. In *Advances in Neural Information Processing Systems*. MIT Press, 2003.

[57] L. J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3):293–321, 1992.

[58] J. F. Montgomery and G. A. Bekey. Learning helicopter control through "teaching by showing". In *Proceedings of the 37th IEEE Conference on Decision and Control*, volume 4, pages 3647–3652, 1998.

[59] E. W. Montroll and H. Scher. Random walks on lattices: Continuous-time walks and influence of absorbing boundaries. *Journal of Statistical Physics*, 9(2):101–135, 1973.

[60] R.J. Muirhead. *Aspects of Multivariate Statistical Theory*. John Wiley & Sons, Inc., Hoboken, NJ, 2005.

[61] Department of the Navy. *The Navy Unmanned Surface Vehicle (USV) Master Plan*. 2007.

[62] A. Y. Ng, H. J. Kim, M. I. Jordan, and S. Sastry. Autonomous helicopter flight via reinforcement learning. *Advances in Neural Information Processing Systems*, 2004.

[63] D. J. Park, B. E. Jun, and J. H. Kim. Fast tracking RLS algorithm using novel variable forgetting factor with unity zone. *Electronics Letters*, 27(23):2150–2151, 1991.

[64] K. Y. Pettersen and T. I. Fossen. Underactuated dynamic positioning of a ship - Experimental results. *IEEE Transactions on Control Systems Technology*, 8(5):856–863, 2000.

[65] K. Y. Pettersen and E. Lefeber. Way-point tracking control of ships. In *Proceedings of the 40th IEEE Conference on Decision and Control*, volume 1, pages 940–945, 2001.

[66] K. Y. Pettersen and H. Nijmeijer. Underactuated ship tracking control: Theory and experiments. *International Journal of Control*, 74(14):1435–1446, 2001.

[67] S. Prentice and N. Roy. The belief roadmap: Efficient planning in linear POMDPs by factoring the covariance. In *Proceedings of the 13th International Symposium of Robotics Research*, 2007.

[68] D. Rathbun, S. Kragelund, and A. Pongpunwattana. An evolution based path planning algorithm for autonomous motion of a UAV through uncertain environments. In *21st Digital Avionics Systems Conference*, volume 2, 2002.

[69] T. RayChaudhuri and L. G. C. Hamey. Active learning for nonlinear system identification and control. In *Proceedings of the 13th IFAC World Congress*, 1996.

[70] M. Rickert, O. Brock, and A. Knoll. Balancing exploration and exploitation in motion planning. In *Proceedings of the 2008 IEEE International Conference on Robotics and Automation*, pages 2812–2817, 2008.

[71] I. Rish, M. Brodie, N. Odintsova, Ma Sheng, and G. Grabarnik. Real-time problem determination in distributed systems using active probing. In *IEEE/IFIP Network Operations and Management Symposium*, volume 1, pages 133–146, 2004.

[72] N. Roy and S. Thrun. Coastal navigation with mobile robots. *Advances in Neural Processing Systems*, 12:1043–1049, 1999.

[73] S. Schaal and C. G. Atkeson. Robot juggling: Implementation of memory-based learning. *IEEE Control Systems Magazine*, 14(1):57–71, 1994.

[74] J. Schneider. Exploiting model uncertainty estimates for safe dynamic control learning. In *Neural Information Processing Systems 9*, pages 1047–1053, 1996.

[75] T. Schouwenaars, B. Mettler, E. Feron, and J. How. Hybrid architecture for full-envelope autonomous rotorcraft guidance. *American Helicopter Society 59th Annual Forum*, 2003.

[76] T. Schouwenaars, B. Mettler, E. Feron, and J. How. Robust motion planning using a maneuver automation with built-in uncertainties. In *Proceedings of the 2003 American Control Conference*, volume 3, pages 2211–2216, 2003.

[77] R. Skjetne, O. Smogeli, and T. I. Fossen. Modeling, identification, and adaptive maneuvering of Cybership II: A complete design with experiments. In *Control Applications in Marine Systems*, pages 203–208, Ancona, Italy, 2004.

[78] J. J. E. Slotine and W. Li. *Applied Nonlinear Control*. Prentice-Hall, Upper Saddle River, New Jersey, 1991.

[79] A. Stentz. Optimal and efficient path planning for partially-known environments. In *IEEE International Conference on Robotics and Automation*, volume 4, pages 3310–3317, San Diego, CA, 1994.

[80] R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224. Morgan Kaufmann, 1990.

[81] C. S. Tan, R. Sutton, and J. Chudley. A quasi-random, manoeuvre-based motion planning algorithm for autonomous underwater vehicles. In *16th IFAC World Conference*, Prague, 2005.

[82] Sebastian B. Thrun. Probabilistic algorithms in robotics. *AI Magazine*, 21(4):93–109, 2000.

[83] A. Vahidi, A. Stefanopoulou, and H. Peng. Recursive least squares with forgetting for online estimation of vehicle mass and road grade: Theory and experiments. *Vehicle System Dynamics*, 43(1):31–55, 2005.

[84] P. Vega, C. Prada, and V. Aleixandre. Self-tuning predictive PID controller. *IEE Proceedings D*, 138(3):303–311, 1991.

[85] J. M. Watkins and K. Kiriakidis. Adaptive control of time-varying systems based on parameter set estimation. In *Proceedings of the 37th IEEE Conference on Decision and Control*, volume 4, pages 4002–4007, 1998.

[86] M. P. Wellman, M. Ford, and K. Larson. Path planning under time-dependent uncertainty. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*, Montreal, Canada, 1995.

[87] Y. Wen-Chun and S. Neng-Yih. Bi-loop recursive least squares algorithm with forgetting factors. *IEEE Signal Processing Letters*, 13(8):505–508, 2006.

[88] V. Wieland. Learning by doing and the value of optimal experimentation. *Journal of Economic Dynamics and Control*, 24(4):501–534, 2000.