Robot Planning and Execution with Unreliable Models

by

Ellis Marshal Ratner

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering– Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Fall 2022

Robot Planning and Execution with Unreliable Models

Abstract

Robot Planning and Execution with Unreliable Models

by

Ellis Marshal Ratner

Doctor of Philosophy in Engineering– Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Anca Dragan, Co-chair

Professor Claire Tomlin, Co-chair

Many modern robotic systems rely on models to complete their tasks. A model captures all aspects of the environment relevant to the robot, and enables the robot to predict the result of any action it may take. Such a model enables the robot to plan and execute an efficient behavior to complete a given task.

Unfortunately, however, no model is perfect. Many real-world phenomena that a robot may encounter — from contact forces such as friction, to unpredictable human behavior — are exceedingly difficult to model precisely. Instead, we typically make certain simplifying assumptions, which make it easier to build a model of the environment for the robot to use in planning and execution. These assumptions, however, invariably break down somewhere. Since the robot relies on such simplified models for planning and execution, its performance can suffer as a result of these inaccuracies. We refer to such models as being *unreliable*, since the robot's planning and execution systems are unable to rely on the model's predictions to produce a behavior to ensure that the robot completes its task in the real-world with a good level of performance. It is therefore critical to design the robot's planning and execution systems to be as robust as possible to unreliable models.

To that end, we focus on increasing the robustness of two key aspects of a model-based planning and execution system: first, how to choose a model that best captures the robot's current environment; and, second, how to plan with that model, despite the possibility that it is unreliable. In this dissertation, we introduce three new algorithms towards increasing the robustness of a robot's planning and execution system, to unreliable models of the real-world. First, we introduce an approach to enable the robot to choose the best model for its current situation *online*, from a larger set of possible models. However, even this best model may not be perfectly reliable. To address this, we complement our approach with the ability to reason about where the robot should and should not rely on the model, which we

describe in the second part of this dissertation. In doing so, the robot is able to leverage the model where it is accurate, but avoid regions where the model may lead the robot's plan astray. More specifically, we present two implementations of this idea in the second part of this dissertation, the main difference being in the *type* of model that each approach assumes — the first focuses on deterministic (or, certain) models, whereas the second focuses on probabilistic (or, uncertain) models.

Throughout this dissertation, to evaluate the effectiveness of our approaches, we present a set of experiments in simulation, as well as on various real-robots. These robots include a 7 degree-of-freedom robot arm, a non-holonomic ground robot, and a free-flying space robot that operates in zero-gravity. Overall, we show how our approaches improve the robot's ability to plan and execute behaviors with greater robustness to unreliable models. Finally, we summarize several preliminary ideas and future research directions around building a unified framework for model-based planning and execution with unreliable models, bringing together all of the key contributions in this dissertation.

Contents

Contents **i**

List of Figures **iii**

List of Tables **vi**

1 Introduction **1**
1.1 Impact of Unreliable Models on Task Performance 3
1.2 Designing a Better Planning and Execution System when Models are Unreliable 7

I Choosing the Best Model for Planning **11**

2 Efficient Estimation with Adaptive Model Sets **12**
2.1 Background and Related Work 14
2.2 The Adaptive Model Set (AMS) Algorithm 16
2.3 Evaluation in Simulation 19
2.4 Robustness to Misspecified Models 23
2.5 Domain Generalization 28
2.6 Evaluation in Hardware 32
2.7 Discussion 34

II Avoiding Unreliable Regions of a Model in Planning **35**

3 Bias Planning Away from Model Inaccuracies using Control-Level Information **36**
3.1 Related Work 38
3.2 Problem Setup 39
3.3 Approach 41
3.4 Experimental Evaluation 44
3.5 Conclusion and Future Work 52

4 Planning Conservatively with Unreliable Human Models 53
4.1 Problem Setup 55
4.2 Background: Robust versus Intent Prediction 56
4.3 A Robust-Control Framework for Intent-Driven Human Prediction 59
4.4 Prediction Comparisons 63
4.5 Implications for Safe Motion Planning 69
4.6 Conclusion 70

IIIFuture Work and Conclusion 71

5 Future Research Directions 72
5.1 Towards a Unified Framework for Planning and Execution with Unreliable Models 72
5.2 Handling a Broader Set of Unreliable Models 75

6 Conclusion 78

Bibliography 79

List of Figures

1.1 Model-based planning and execution systems typically exhibit the structure illustrated here. (Left) The system relies on a model, which summarizes the robot's knowledge of the real-world, to predict the outcome s' of taking action a at state s. The system may select this model *offline* (i.e., before deploying the robot), or *online* (i.e., after the robot has begun operating). (Center) Using this model, the robot plans a path to the goal using an appropriate planning algorithm. (Right) Finally, the robot executes this plan, using control algorithms to follow the planned path. 2
1.2 (Left) The robot plans a path to move its end effector to a goal pose, shown by the red dashed line. (Right) While executing this path, one of the robot's joints unexpectedly locks (due to someone physically preventing it from moving about that joint). Therefore, the robot is unable to reach the goal along this path. . . 4
1.3 (Left) The Astrobee robot plans a path to goal, shown by the dashed red line, passing directly through the fan, which it neglected to model. (Right) As a result, when the robot executes this path, it is blown into the obstacle along the path shown by the solid red line. 5
1.4 The TurtleBot navigates to a goal in an environment shared with a human. (Left) To avoid collision with the human, the robot uses a model that predicts a probability distribution over the human's possible future paths, shown in blue. Darker blue indicates higher probability, and lighter blue indicates lower probability. (Right) In contrast to the robot's prediction, the human walks straight. As a result, the robot fails to avoid colliding with the human, as shown. 6
1.5 We illustrate each of the three approaches introduced in this dissertation, in the context of the examples presented in this section. Chapter 2 (Left) focuses on how to choose a model. Chapter 3 and Chapter 4 focus on how to plan with unreliable deterministic and uncertain models, respectively. Finally, Chapter 5 provides our perspectives on how to unify these approaches into a single model-based planning and execution framework. 7

2.1 We apply our adaptive estimator in three domains: (left) Manipulator moves into contact with a table, (center) interaction dynamics between the vehicles wheels and the ground change when it moves from dirt to icy terrain, (right) human switches goal locations as they move through a hallway (path shown in color), affecting a nearby robot's motion plan (shown in grey arrows). 13
2.2 An example depicting the operation of our AMS estimator on the Kinova JACO 7 DOF manipulator system. As in an experiment in Sec. 2.6, the end-effector follows a position trajectory. The person immobilizes one of the robot's joint at time step k_1. By time step $k_1 + N_V$, the algorithm detects that the current model set is insufficient to explain the encoder measurements– so it expands the model set appropriately. Later, the algorithm removes the models that are no longer needed to explain the measurements. At time step k_2, the person lets go of the robot, allowing the joint to move freely again. 16
2.3 NASA 6 DOF Europa lander arm, tracking a position trajectory for the end-effector (the scoop). The actuator on the second joint suffers a temporary 75% degradation. Using the nominal model for controlling the manipulator leads to a trajectory that diverges from the reference, since controller is unable to compensate for the degradation. 20
2.4 Performance comparison of an IMM estimator versus our AMS estimator for the 6 DOF arm. Error bars indicate standard deviation. 24
2.5 Evaluating the robustness of our AMS estimator versus an IMM estimator, when sensor noise is mismodeled. The sensor is modeled as having noise with $\sigma_v^2 = 0.01$, whereas it is simulated with $\sigma_v^2 = 0.03$ (i.e. sensor is noisier than modeled). Error bars indicate standard deviation. 26

3.1 (Left) The robot plans a path to the goal through a fan's strong air current not captured by its model. (Right) To avoid the fan, the robot must plan an alternate path. 37
3.2 An illustration of our approach. The planning-level first generates a desired path (dashed line), which the control-level then executes. If the robot's actual path (solid line) deviates, then we update the control-level discrepancy model with the estimated discrepancies $\hat{d}$. The planner uses the new d to bias away from actions likely to result in a discrepancy, using the control-level penalty. 42
3.3 A wire obstructs the robot's path to goal π_1. An alternate path, such as π_2, is required to reach the goal. 46
3.4 A simulated fan exerts a force on the robot, indicated by the blue arrows. Unaware of the fan, the robot initially plans path π_1, but is pushed into the obstacle. An alternate path, such as π_2, is required to complete the task. 47
3.5 3 fans and 1 wire obstruct the robot's path to goal. We show several alternate paths in red. 48
3.6 The mean and standard errors of time to complete the task, for the wire (S1), fan (S2), and 3 fans + 1 wire (S3) scenarios. 50

3.7 Astrobee plans a path to the goal, but the path is obstructed by an unseen wire. 51

4.1 When intent-driven human models are misspecified in Bayesian predictors, robots confidently plan unsafe motions (top). Our approach (bottom) trusts the intent-driven model only to remove completely *unlikely* human actions, resulting in safer robot plans despite a misspecified model. (not depicted here) When planning using the worst-case predictor, the robot has to leave the environment entirely to avoid the predicted human state. 54

4.2 Intent-driven predictions 1.8 seconds into future. Human is a black dot, grey dashed circle is the full FRS. (left) Full state distribution shown in green gradient with dark color indicating higher probability, (center) $\epsilon = 0$ is the full FRS, (right) $\epsilon > 0$ drastically reduces set of states to avoid. 58

4.3 Effect of the belief and the δ-threshold on the admissible set of controls (shown in upper-left inset) and the overall predictions (shown in pink) for 3 seconds into the future. 62

4.4 (left) Initial set in z-space (in grey). Likely control distribution for $\delta = 0.01$ shown projected in $h_x - h_y$ plane. Comparisons of the resulting joint state if the human moves directly towards g_1 (in red) versus towards g_2 (in blue). (right) 4 seconds BA-FRS and its projection into x_H-space. 63

4.5 Comparisons of Bayesian and BA-FRS predictions for static vs. time-varying human intent. Dashed lines are the full FRS. Predictions are for 2 seconds for the static parameter and 1.8 seconds for time-varying. For the Bayesian predictor, we choose ϵ to capture the $(1 - \delta)$ most likely states. 64

4.6 Comparison of the intent-driven Bayesian, our BA-FRS, and the full FRS predictions for three scenarios. In the first row the human moves towards one of the modelled goals. In the middle the human moves towards an *unmodelled* goal. In the last row the human is moving towards a modelled goal (g_2) but they take a suboptimal path under our model because they are avoiding an *unmodelled* obstacle on the ground (shown in grey circle). The belief over g_1 is visualized over time in the lower-left inset plot. 66

4.7 Simulated human moves to modelled g_1 but with varied optimality from $\sigma = 0$ (optimal) to π (random). Both predictors use a fixed $\sigma = \pi/4$. 68

4.8 (top) Simulated human moves to one of 7 *unmodelled goals* in an optimal trajectory. Results are in increasing misspecification of the goal. (bottom) Simulated human moves to modelled goal g_2 but has their straight-line path obstructed by an *unmodelled obstacle*. Results from 7 unmodelled obstacles are shown in increasing deviation from the straight trajectory. 69

5.1 We illustrate one way to unify our approaches in Chapter 2, Chapter 3, and Chapter 4, into a single model-based planning and execution framework. 73

List of Tables

2.1 3 DOF manipulator, all models available to estimation algorithm, position estimation and tracking errors. Mean and standard error in parentheses. 24
2.2 Experiment for H3 (a): 3 DOF manipulator, sensor is noisier than modeled in estimation algorithm (position). 27
2.3 Experiment for H3 (b): 3 DOF manipulator, link masses are slightly different than model used by estimator (position). 27
2.4 Experiment for H3 (c): 3 DOF manipulator, estimator has no access to model for system mode (position). 27
2.5 Comparison of performance in the driving domain. 30
2.6 Comparison of performance in the robot navigation domain. Mean values reported with standard errors in parenthesis. 32

3.1 Mean distance traveled (m) by the robot in each scenario, with standard error in parentheses. 49
3.2 Mean planning time in seconds, with standard error in parentheses. 49
3.3 Mean number of replans (standard error in parentheses). 50

Acknowledgments

There are many people to whom I am incredibly grateful for all of their support and encouragement throughout my PhD. I would like to make a few specific acknowledgements here.

To my advisors Anca and Claire, thank you for providing me with invaluable guidance through all the bumps along the way, and affording me the freedom to explore and grow as a researcher.

To my committee members, Terry and Ken, thank you for all of your valuable feedback that has greatly helped me to shape this dissertation. Thank you Terry for the many fascinating discussions and research guidance, and for introducing me to numerous incredibly exciting research problems at NASA in the area of robotics and autonomy.

Thank you Max for being an incredible mentor and collaborator throughout the years, and for introducing me to the exciting world of robotics research in the first place. I am excited to continue working with you in the future.

During my PhD, I have had the privilege of being part of two great labs at Berkeley; thank you to all members of the Interact Lab and the Hybrid Systems Lab for this incredible opportunity.

Thank you to the Intelligent Robotics Group at the NASA Ames Research Center, and specifically to Arno Rogg, Trey Smith, Brian Coltin, Bob Morris, and Marina Moreira for all of your guidance, and for providing me with the opportunity to work on some really amazing problems in space robotics.

Finally, thank you to all my family and friends for providing me with an incredible amount of support and encouragement "behind the scenes" throughout my PhD.

This work was supported by a NASA Space Technology Research Fellowship (grant 80NSSC18K1147).

Chapter 1

Introduction

Many modern robotic systems rely on model-based planning and execution to complete their tasks. The *model* predicts the outcome of the robot taking an action at a particular state. Regardless of where this model comes from — whether it is derived through first-principles, acquired through a data-driven or machine learning method, and so on — no model can perfectly capture the real-world. Every model will make an incorrect prediction for some states and actions. A challenge, however, arises when these wrong predictions have negative downstream effects on the robot's planning and execution, such as sub-optimal performance, or in the more extreme case, complete failure at completing the task, or intermittent failures over several repetitions of the task. In these cases, we refer to the model as being *unreliable*, in that the robot could not rely on the model to complete its task.

For example, accurately modeling friction and contact is challenging (e.g., [23]), the dynamics of non-rigid bodies very challenging (e.g., [85, 4, 84, 31]), and modeling the complex decision-making processes of humans is still an open research question (e.g., [70]). In these cases, it is difficult to avoid models that will sometimes poorly predict the outcomes of the robot's actions for at least some regions of the state space where the robot operates. As a result, if the robot's planner relies on these models, the quality of the robot's plan can suffer from these poor predictions — the robot may come up with a plan to complete the task that proves to be sub-optimal, or even infeasible, when executed in the real-world.

To avoid these possibilities, it is important to design a robot's planning and execution system to be as robust as possible to operating with an *unreliable model* of the real-world. Two key aspects of designing a model-based planning and execution system are:

1. the choice of *model*, which predicts the outcomes of the robot's possible actions, as described earlier; and,

2. the choice of *planner*, which utilizes the aforementioned planner to devise a behavior that enables the robot to complete the task when executed.

In this dissertation, we investigate how to improve both aspects of the system to better prepare a robot to complete tasks despite these inevitable unreliable models. Fig. 1.1 summa-

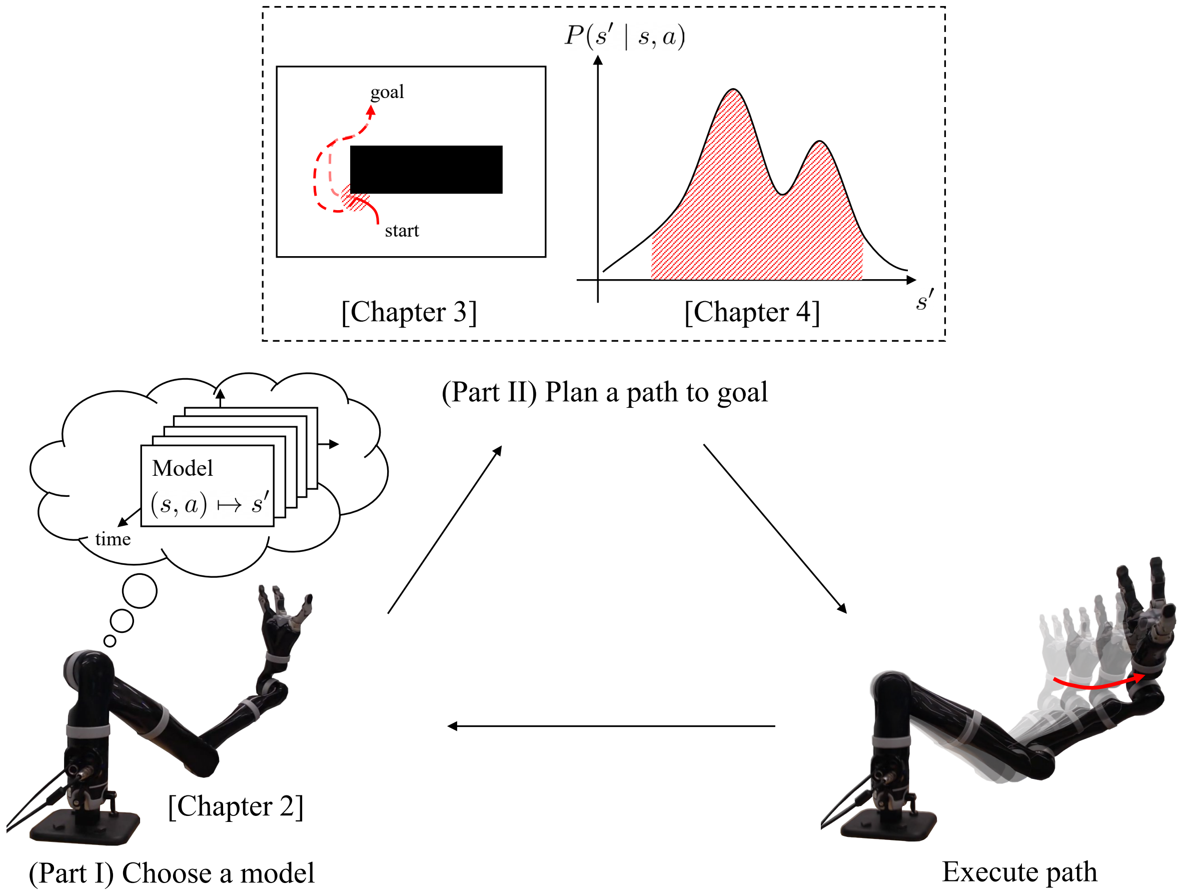

Figure 1.1: Model-based planning and execution systems typically exhibit the structure illustrated here. (Left) The system relies on a model, which summarizes the robot's knowledge of the real-world, to predict the outcome s' of taking action a at state s. The system may select this model *offline* (i.e., before deploying the robot), or *online* (i.e., after the robot has begun operating). (Center) Using this model, the robot plans a path to the goal using an appropriate planning algorithm. (Right) Finally, the robot executes this plan, using control algorithms to follow the planned path.

rizes how our key contributions in this dissertation fit into the flow of a typical model-based planning and execution framework.

In such systems, it is typical to try designing one model, prior to the robot being deployed, which captures everything that the robot may encounter throughout its lifetime. This can be a very challenging modelling problem, however; building a single model to handle a broad set of scenarios that the robot may encounter is very difficult. As an alternative, in the first part of this dissertation (Chapter 2), we propose to have the robot choose a model *online*, i.e., while the robot is already operating in the real-world. To do so, we enumerate a *large* *set of possible models* prior to deploying the robot, and enable the robot to choose the best model based on the latest information available.

Unfortunately, however, not even the best model that the robot chooses online is perfect; the robot will invariably encounter scenarios where that model is no longer reliable. In many cases, however, the model is not inaccurate everywhere— instead, the model is inaccurate only in smaller regions of the robot's state and action space. Motivated by this, in the second part of this dissertation (Chapers 3 and 4), we present planning approaches that rely on the model *only where we believe it to be an accurate representation of the real-world.* In this part, we present two approaches that implement this key idea. The first approach, in Chapter 3, focuses on *deterministic* or *certain* models (i.e., models that predict a single outcome for a given state and action). We present a way to use control-level information gathered during execution to determine where the model is inaccurate, which we ultimately use to bias the planner away from these inaccuracies in future replanning. The second approach, in Chapter 4, focuses instead on *stochastic* or *uncertain* models (i.e., models that predict a probability distribution over outcomes for a given state and action). Specifically, we focus on uncertain models for human motion prediction. Relying on the model's predicted distribution over outcomes enables the robot to find an efficient plan to complete the task. However, if parts of that distribution are unreliable, in that the probabilities do not well-characterize the human's behavior in the real-world, then the robot's performance will suffer. To avoid this, our approach relies on the model to predict only what is very unlikely— which enables the robot to behave conservatively with respect to the all predictions the model makes, which we do not believe to be accurate.

Since the first part of this dissertation (Chapter 2) focuses on the *choice of model* aspect, and the second part (Chapters 3 and 4) focuses on the *choice of planner* aspect, it seems natural to combine these three approaches into a single planning and execution framework. To that end, in Chapter 5, we explore some preliminary thoughts on how to build a model-based planning and execution framework that unifies all of these approaches. Specifically, we discuss several directions of future research that would be helpful towards realizing this unified framework. Following that, we discuss some limitations of the current work, and examine some possible ways to extend our approach to handle a broader set of unreliable models. While many of these ideas are not yet complete, we believe they form promising avenues for future research. Finally, in Chapter 6, we conclude by summarizing the contributions in this dissertation work.

1.1 Impact of Unreliable Models on Task Performance

What is an unreliable model?

When viewed from the perspective of planning, we can characterize models as being either *reliable* or *unreliable*. A reliable model is one that predicts the outcomes of actions with

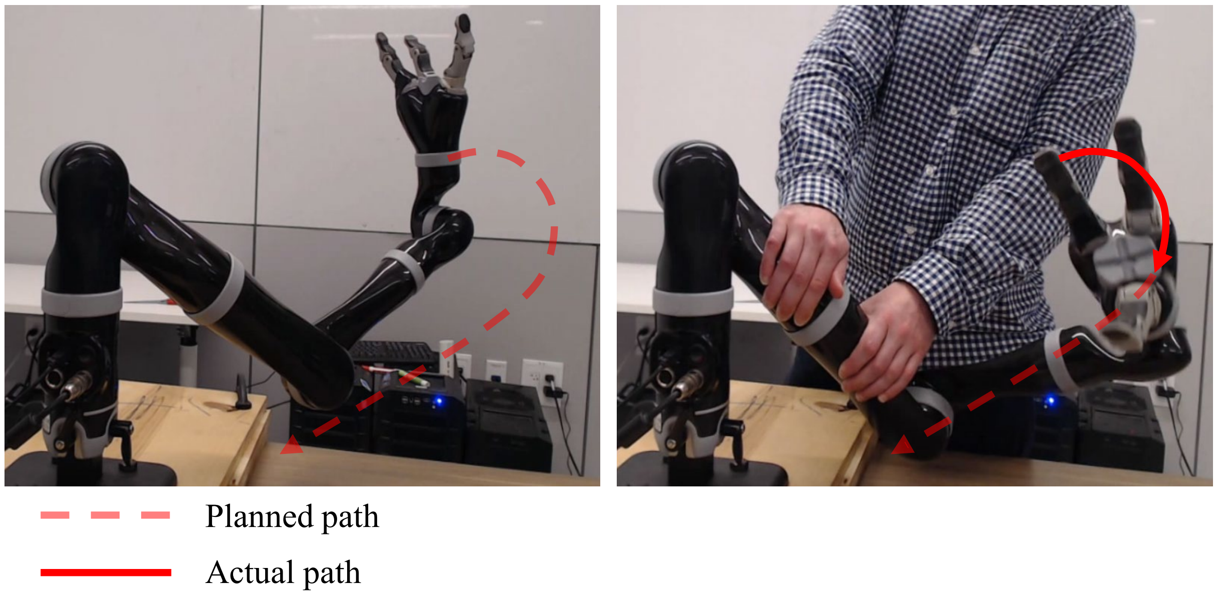

Figure 1.2: (Left) The robot plans a path to move its end effector to a goal pose, shown by the red dashed line. (Right) While executing this path, one of the robot's joints unexpectedly locks (due to someone physically preventing it from moving about that joint). Therefore, the robot is unable to reach the goal along this path.

sufficient accuracy so that the robot can compute a plan that is successful— both from the perspective of task performance and completion— when executed in the real-world. Importantly, a reliable model need not be perfect; loosely speaking, it only needs to be "good enough" for the task. On the other hand, an unreliable model is one whose predictions are inaccurate in ways that lead to a plan that is unsuccessful at completing the task with a sufficient level of performance during execution.

Examples of unreliable models in planning

To make the definition of an unreliable model more concrete, we present three example tasks, and highlight how an unreliable model impacts overall task performance. We will return to these examples in later chapters to demonstrate how our approaches improve upon a standard model-based planning and execution system to more appropriately handle each.

For the first example, consider the task of moving a robot's end effector to a desired pose, as shown in Fig. 1.2. A standard approach is to use a motion planner — which utilizes a model of the robot's kinematics and dynamics, as well as of the environment (e.g., where dynamic and static obstacles are) — to find a collision-free path to the goal, satisfying any additional constraints such as joint position, velocity, acceleration limits, and so on. We

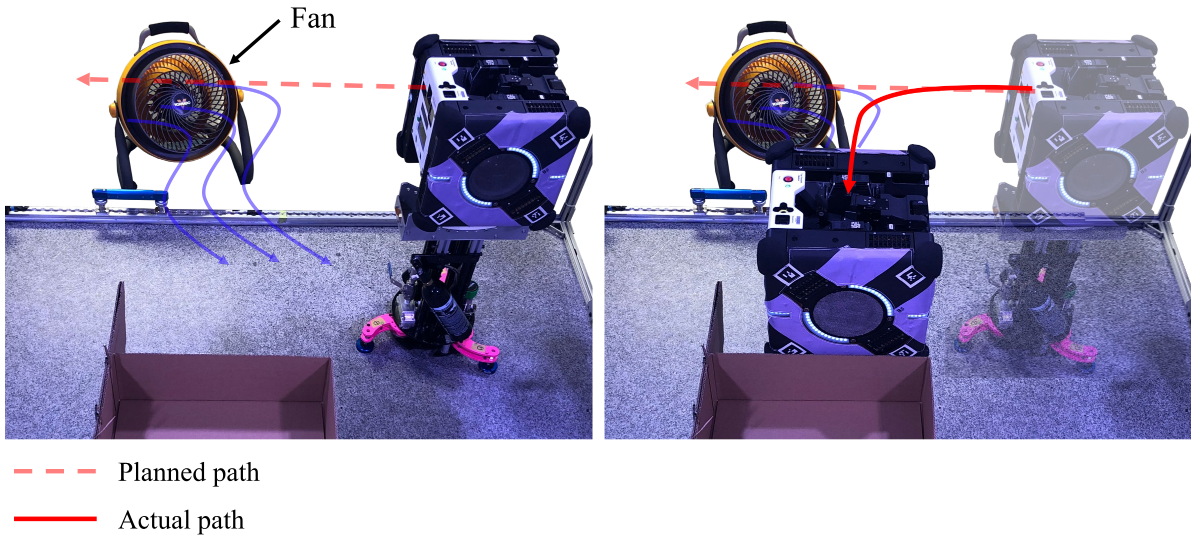

Figure 1.3: (Left) The Astrobee robot plans a path to goal, shown by the dashed red line, passing directly through the fan, which it neglected to model. (Right) As a result, when the robot executes this path, it is blown into the obstacle along the path shown by the solid red line.

show such a path by the dashed red line in Fig. 1.2 (Left). In this example, when the robot begins executing this path, one of the robot's joints unexpectedly becomes locked, due to someone physically holding the robot, as in Fig. 1.2 (Right). As a result, the robot gets stuck, and can no longer complete the task of reaching its goal. This is because the motion planner's model was *unreliable*, in that it failed to predict the robot's locked joint.

For the second example, consider NASA's Astrobee free-flying robot [10], as shown in Fig. 1.3. Astrobee is designed to operate in a zero-gravity environment, such as a space station, and uses an impeller-based propulsion system to move; here we show the robot operating in a specialized lab that enables the robot to move in the plane with very little friction. The robot's task is to move from its current location to a goal location at the other end of the lab space. As in the previous example, the robot uses a model-based motion planner to find a collision-free path to the goal. However, in this example, the robot is unaware of the fan. Therefore, the planner's model neglects to capture the effect of the fan on the robot's motion. So, the robot plans a path that crosses the fan, as shown by the dashed line in Fig. 1.3 (Left). Because this path passes directly through the fan's air current, the robot is pushed off of the planned path, instead taking the path shown by the solid red line in Fig. 1.3 (Right), and the robot is unable to reach its goal. Because the model failed to accurately predict the effect of the fan on the motion of the robot around its air current,

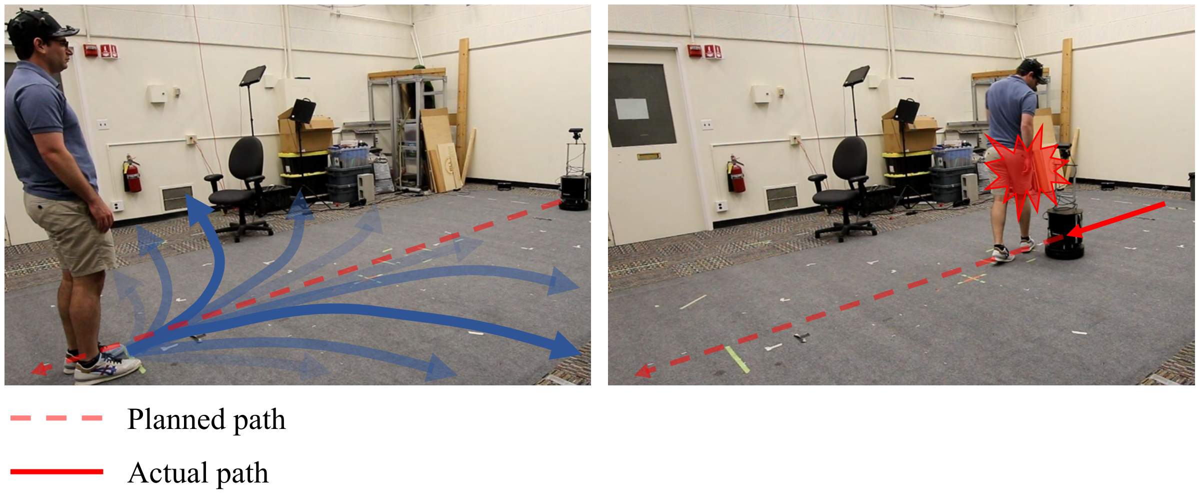

Figure 1.4: The TurtleBot navigates to a goal in an environment shared with a human. (Left) To avoid collision with the human, the robot uses a model that predicts a probability distribution over the human's possible future paths, shown in blue. Darker blue indicates higher probability, and lighter blue indicates lower probability. (Right) In contrast to the robot's prediction, the human walks straight. As a result, the robot fails to avoid colliding with the human, as shown.

it is *unreliable*. In this example, that the planner used an unreliable model caused the robot to be unable to complete its task of reaching the goal.

For the third example, we consider the TurtleBot navigating to a goal in a shared human robot environment, as shown in Fig. 1.4. As in the previous two examples, we use a model-based motion planner to find a path to goal for the robot. However, here we use an *uncertain* model, which provides a probability distribution over the possible future motion of the human. In this example, the model predicts, with high probability, that the human will either walk to the right or the left, and will continue walking forward with low probability, as shown in Fig. 1.4 (Left). Here, darker blue indicates high probability, and lighter blue indicates low probability. Unfortunately, however, this probability distribution is *uncalibrated*, in the sense that the predicted probabilities do not match the human's actual behavior. As a result, when the human unexpectedly moves straight, the robot does not have sufficient time to avoid a collision, as shown in Fig. 1.4 (Right). Again, because the model was *unreliable*, the robot failed to achieve the task of safely reaching its goal. This example also illustrates that our notion of reliability applies to both deterministic (i.e., certain) and uncertain models alike.

In reality, few models are reliable; furthermore, the reliability of a model is often not known until actually observing the outcomes of taking actions in the real-world, so as to truly evaluate the quality of the robot's planned behavior. Despite this, a majority of

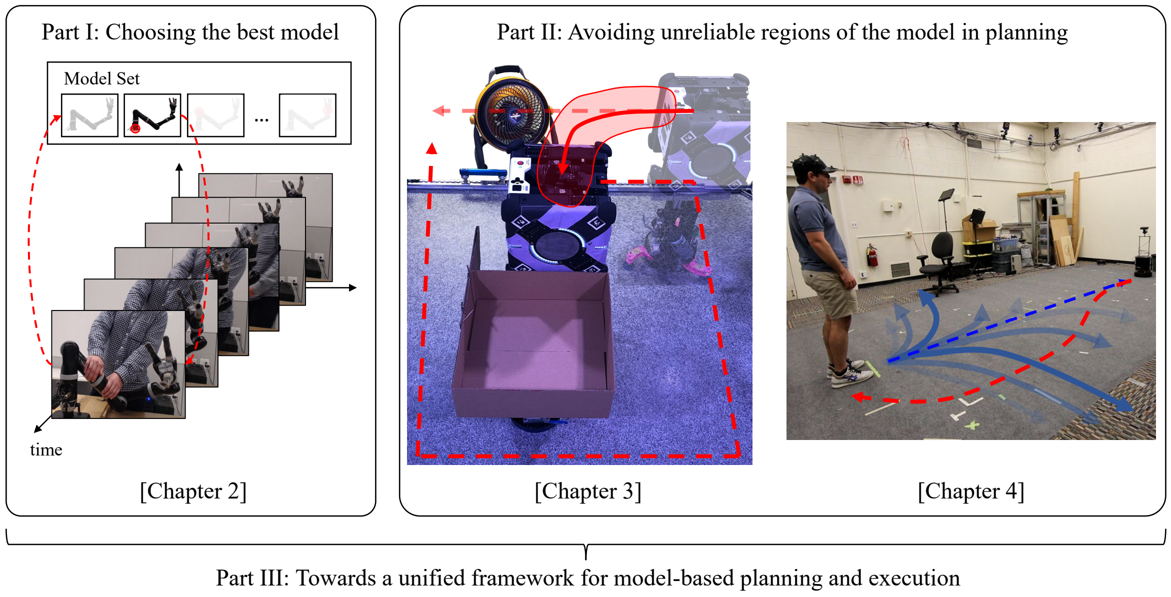
Part I: Choosing the best model
Model Set
time
[Chapter 2]
Part II: Avoiding unreliable regions of the model in planning
[Chapter 3]
[Chapter 4]
Part III: Towards a unified framework for model-based planning and execution

Figure 1.5: We illustrate each of the three approaches introduced in this dissertation, in the context of the examples presented in this section. Chapter 2 (Left) focuses on how to choose a model. Chapter 3 and Chapter 4 focus on how to plan with unreliable deterministic and uncertain models, respectively. Finally, Chapter 5 provides our perspectives on how to unify these approaches into a single model-based planning and execution framework.

research in planning — both classical planning and planning under uncertainty — relies on the model that it uses in generating a plan or policy; relatively less work has focused on settings where the model is not fully reliable. The focus of this dissertation, therefore, is to enable robots to complete tasks despite having access only to unreliable models in planning and execution.

1.2 Designing a Better Planning and Execution System when Models are Unreliable

So far, we have examined several examples of unreliable models. We have seen how the models' inability to accurately predict the outcome of the robot's actions in the real-world causes the robot's task performance to suffer. How, then, can we improve a standard model-based planning and execution system in settings such as these examples? In this dissertation, we propose new approaches to improve the two key aspects illustrated in Fig. 1.1.

Choose the Best Model from a Large Set

In some settings, it is very difficult to specify a single model that reliably characterizes the environment everywhere the robot operates over its lifetime. In Sec. 1.1, we examined the example of a robot's joint locking unexpectedly (Fig. 1.2). Since this event was caused by some unpredictable factors (i.e., a person holding the robot), it may be exceedingly difficult to model exactly *when* and *where* the joint will lock beforehand. More generally, intermittent and unexpected changes in the environment or operation of the robot's sensors and actuators can be difficult to predict prior to deployment, and may require significantly different models to effectively plan over.

Rather than choosing a single model beforehand, as shown in Fig. 1.1 (Left), we can detect during execution when our current model is no longer reliable, and then switch to a new model selected from a larger set of candidates. We can then use that new model to replan a more suitable path to the goal.

Prior work has introduced several ways to choose the best model from a set— notably Multiple Model Adaptive Estimation (MMAE) [50] algorithms, Interactive Multiple Model (IMM) algorithms [61, 9], and Generalized Pseudo-Bayesian (GPB) algorithms [1, 13, 33]. Unfortunately, however, these methods suffer with respect to computational efficiency, particularly when the full set of possible models is large. We address this issue in Chapter 2 by proposing a new algorithm for performing the model selection more efficiently.

Specifically, in Chapter 2, we introduce an estimator that leverages an *adaptive model set* to efficiently estimate the best model at each point in time. While adaptive model sets have been used in prior work, e.g. [83, 41, 67, 45, 44, 43], designing a general mechanism for adapting the model set has remained challenging. To that end, our key idea is to start with a small set of models, and to only expand that set when none of its models sufficiently explain the observations. Our approach compares favorably to competing approaches with respect to computation time, without sacrificing estimation accuracy. One additional advantage of our approach is that it can readily be integrated with models *learned* — either online or offline — from data, e.g., using model learning [57], model-based reinforcement learning [63], or learning-based control [21]. We discuss avenues for future work around utilizing learned models in Chapter 5.

Avoid Unreliable Regions of the Model when Planning

Despite our best efforts to choose the best model available, no model is perfect, and invariably will be unreliable in some regions of the state and action space. The model's inaccurate predictions, in turn, may prevent the robot from being able to complete its task. We address this in the second part of this dissertation, where we propose approaches that enable a planner to avoid regions in the state and action space where we believe the model to be inaccurate. Chapters 3 and 4 present specific implementations of this idea when the model is deterministic or stochastic, respectively.

Biasing the planner away from unreliable parts of a deterministic model

Consider again the example illustrated in Fig. 1.3 in Sec. 1.1. Here, the robot is planning a path to the goal, but its model neglects to capture the effect of the fan on the robot. Consequently, the robot plans to pass through the fan, and as a result the robot is pushed into an obstacle before it can reach the goal. One possible solution would be to learn a better model, i.e., one that accurately predicts the effect of the fan on the robot (using, e.g., model learning [57], model-based reinforcement learning [63], or learning-based control [21]). This may be a very challenging learning problem, however, and in fact not necessary— we observe that the robot can still reach the goal if it were to take an alternate route that *avoids* the fan all together.

Motivated by this, in Chapter 3, we introduce an approach to *bias* the planner away from regions where we believe the model to be inaccurate. The challenge, however, is how to construct such a bias. Prior work has proposed to bias the planner by introducing an additional cost on taking states and actions where we believe the model to be inaccurate [81, 80], or to utilize a learned classifier to predict where the model will be inaccurate inside the planning loop [51, 55]. These approaches, however, neglect potentially valuable information gathered at the control-level that can better inform the planner's bias. Motivated by this, our key idea in this work is to observe the performance of the path following controller, and use this information to infer where the model may be inaccurate. We then translate that information back up to the planner in the form of a bias, which then guides replanning to find an alternate path avoiding these inaccurate regions. Chapter 3 explains in detail how our approach achieves this, and presents an experimental evaluation in simulation and in real-robot experiments on NASA's Astrobee [10] robot.

Planning conservatively with an unreliable stochastic model

When planning around other agents, such as humans, in the environment, we often use *stochastic* (or, *uncertain*) models to predict how those agents will behave in the future; for example, a large body of prior work focuses on leveraging a model of the humans' intent to predict what they will do in the future [6, 65, 86, 34, 12]. Optimistically trusting these predictions, the planner can come up with efficient behaviors for the robot that avoid undesirable situations, such as collisions or other conflicts. This planned behavior will lead to good performance when executed by the robot only if the predictions are indeed correct. Unfortunately, however, such behaviors are often very brittle if this is not the case — if the agent does something unexpected under the model's prediction, the planner has no way of accounting for that possibility, and may lead the robot to perform poorly, as in the example illustrated in Fig. 1.4.

One way to address this issue is to be completely conservative with respect to the model's predictions. That is, to ensure that the planner avoids conflicts under any possible future behavior of the agents, e.g., using a reachability analysis to determine all possible states the agent may take in the future [53, 19]. Such a high level of conservatism, however, may have a negative effect on efficiency of the planned behavior at best, or may lead to the robot failing to complete its task all together.

In Chapter 4, we propose an approach to find a balance between the optimistically relying on the model, and conservatively guarding against possibilities not captured by the model's predictions in planning. Our key idea is to only rely on the model to predict what is exceedingly unlikely, and otherwise plan conservatively with respect to all other possibilities under the model. We show how to implement this idea in a reachability computation, and how to use the resulting predictions in planning to enable the robot to better navigate in situations such as the example in Fig. 1.4. We present evaluations of our approach in simulation and experiments involving a TurtleBot navigating to a goal in an environment shared with a human.

Part I

Choosing the Best Model for Planning

Chapter 2

Efficient Estimation with Adaptive Model Sets

Whether operating on the road, on a deep space exploration mission to a distant world such as Europa, or in a household around people, robots frequently face changing environments and operating conditions of their sensors, actuators, and so on. These changes arise for a variety of reasons, such as traversing changing terrains, faults induced by wear and tear from extreme operating conditions, or navigating around people with uncertain and changing intentions.

While obtaining a single model that captures all of these possible situations is often prohibitively difficult, we can often more easily construct a separate model for each of these operating modes, for example by first-principles or data-driven approaches. Planning for the robot, however, requires a single model (or, in some cases, a distribution over some set of models). Therefore, we must estimate the model based on the latest sensor measurements. This estimation problem can be posed as a general filtering problem over the space of possible models; however, this set is typically large (or even infinite) in practice. Furthermore, since the dynamics switch over time, an optimal estimator must track all possible mode sequences, the number of which grows geometrically in the time step. While there exist approximate estimators [61, 9], these typically still require large model sets at estimation time, which can be computationally expensive to manage.

In this chapter, we propose to use only a small subset of models for estimation at each time step. The fundamental challenge with such an approach, however, is that the best model may not be in this subset. To address this, our key idea is that the robot can detect when none of the current models sufficiently explain the sensor measurements, which in turn it can use as an indicator of when to expand the model set. Equipped with this idea, we propose a multiple model estimator with a novel mechanism for *adapting* the model set. Starting with a small subset of models, our algorithm only expands that set when the true observations are assigned low likelihood under *all* models in our current model set. To determine which model to add, we measure the predictive performance of each model currently *not* in the set and add the one from the set which assigns highest likelihood to the true observations.

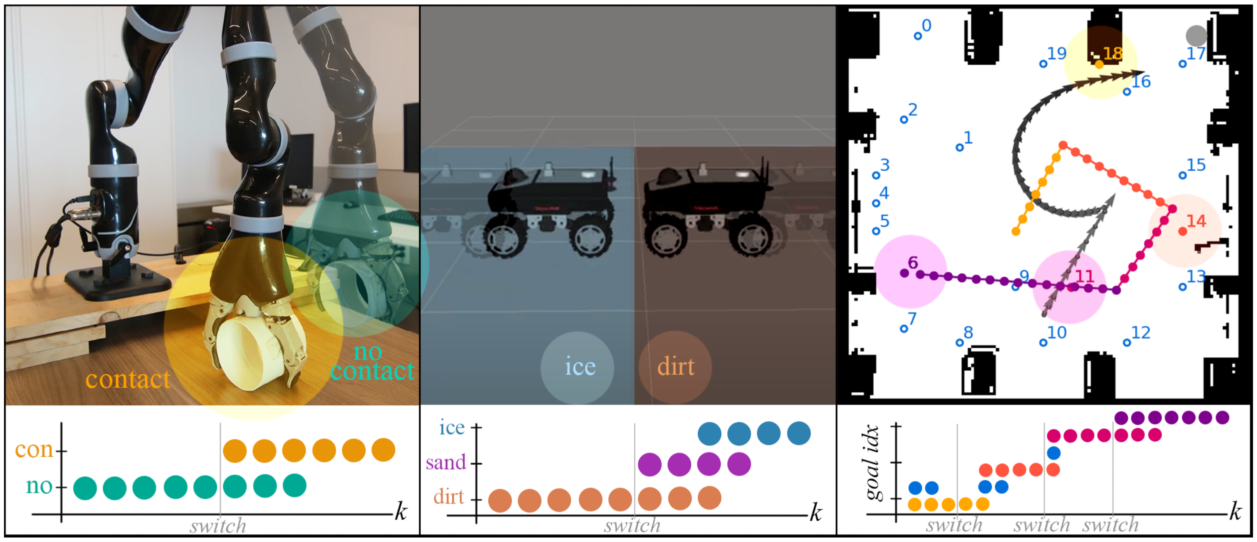

Figure 2.1: We apply our adaptive estimator in three domains: (left) Manipulator moves into contact with a table, (center) interaction dynamics between the vehicles wheels and the ground change when it moves from dirt to icy terrain, (right) human switches goal locations as they move through a hallway (path shown in color), affecting a nearby robot's motion plan (shown in grey arrows).

Further, to keep the subset as small as possible, we remove any models with low posterior probability which indicates that they are no longer needed to explain the measurements; should they become necessary again, we will detect this and add them in later on.

We experimentally evaluate the performance, with respect to efficiency and accuracy, and the robustness of our algorithm on a set of simulations on the following domains: trajectory tracking for a 3 DOF and a 6 DOF manipulator, which encounter actuation faults; trajectory tracking for a skid-steering vehicle driving across uncertain and changing terrains; and, trajectory planning for a Dubins' car-like robot navigating around a human with changing intentions. Through these evaluations, we show that our adaptive estimation algorithm is computationally favorable when compared to non-adaptive estimators without sacrificing on estimation performance. Additionally, we found that our adaptive estimator was actually more accurate at predicting the true system mode in all experiments. We attribute this to an additional layer of filtering in our mechanism for adapting the model set, providing more stability in predicting the most likely mode.

This chapter is based on research previously published in [66], in collaboration with Andrea Bajcsy, Terry Fong, Claire Tomlin, and Anca Dragan.

2.1 Background and Related Work

We wish to estimate the state x_k at time k, given the sequence of measurements observed, $y_{0:k}$, and controls taken, $u_{0:k}$. We assume the system evolves according to:

$$\begin{aligned} x_{k+1} &\sim P(\cdot \mid x_k, u_k, m_k) \\ y_{k+1} &\sim P(\cdot \mid x_{k+1}, m_{k+1}) \\ m_{k+1} &\sim P(\cdot \mid m_k) \end{aligned} \tag{2.1}$$

where $m_k \in \{1, 2, \ldots, N\}$ is the *mode* of the system at time k. We assume access to a set of candidate models that could characterize the dynamics mode of the robot, obtained from first-principles or identified via data-driven approaches. We further assume that the state and mode are initially *unknown.* So, at any given time k, we must *simultaneously estimate the state and mode* of the system. Note that we do, however, assume full knowledge of the system dynamics in Eq. 2.1.

Consider the state estimator $\hat{x}_k$ of x_k, defined to be the expected state given the sequence of measurements taken up to time k. Then

$$\hat{x}_k := E[x_k \mid y_{0:k}] = \sum_{m_{0:k}} P(m_{0:k} \mid y_{0:k}) E[x_k \mid y_{0:k}, m_{0:k}]. \tag{2.2}$$

Observe that such an estimator requires enumerating *all possible mode sequences of length* $k+1$, or N^{k+1} sequences, and thus typically cannot be implemented [30, 67, 42]. Even in the case where we make the simplifying assumption that we know the state x_k, and only need to estimate m_k, this geometric complexity remains.

Due to these issues, prior work has focused on developing *approximate* methods, which do not track all N^{k+1} mode sequences to estimate the state and/or mode. We broadly divide such related work into *fixed model set*, *online model learning*, *skill learning*, and *adaptive model set* approaches.

Fixed model set

There has been a large amount of research on estimation algorithms that maintain a fixed set of models (pre-defined or learned a priori) to simultaneously estimate the state and mode of the system. The Multiple Model Adaptive Estimation (MMAE) algorithm [50] keeps a bank of estimators, one per mode, each of which computes the *mode conditioned estimate* $E[x_k \mid y_{0:k}, m_{i,k}]$; traditionally, these are implemented as Kalman filters. To handle tracking of the mode sequence, the MMAE algorithm makes the simplifying assumption that the mode is *fixed* but unknown at $k = 0$, and therefore the estimator in Eq. 2.2 simplifies to:

$$\hat{x}_k = \sum_{i=1}^{N} P(m_k = i \mid y_{0:k}) E[x_k \mid y_{0:k}, m_k = i]. \tag{2.3}$$

While computationally convenient, such an approximation often does not perform well when the mode evolves over time, as in Eq. 2.1. To address this, the Interacting Multiple Model (IMM) algorithm [61, 9] adds an additional *mixing step* to the estimator update each time step, to account for the mode switches that may occur over time. While still an approximation, IMM estimation typically outperforms the MMAE algorithm, without considerable overhead [32]. Several other algorithms have been designed to handle such switches, a notable class of those being the generalized pseudo-Bayesian (GPB) estimators [1, 13, 33]. Finally, Cully et al. leverage an offline precomputation to enable the system to detect and compensate for failures (e.g. the loss of an actuator) [17]; while no new models are learned online, their system is still capable of adapting to novel situations at execution time. Our method is complementary to these approaches, as we introduce a mechanism for *adapting* the set of models, rather than keeping a fixed set for all time.

Online model learning

In Event-Triggered Learning (ETL), the algorithm *learns a new model* when there is a mismatch between what the existing model predicts, and what is actually measured [73]. Harris et al. introduce a method for measuring the overall performance of a control system for the purposes of indicating that a chosen controller may be insufficient for the task at hand [26]. MOSAIC [28] simultaneously learns the set of all possible models, as well as how to select the subset relevant for the current environment in a single learning-based framework. In contrast to these approaches, our method selects from an *existing* set of models acquired a priori. These online learning approaches, however, are complementary: a combined algorithm could determine whether to choose from this existing set of models or identify a new model online.

Learning new skills

Within robot skill learning, some prior work focuses on how to combine existing model sets with new model sets (e.g. of how to accomplish a task). Koert et al. manage a set of skills, represented by Gaussian Mixture Models, and introduce a mechanism for adding and removing skills from this set, analogous to our procedures from adding and removing models from a model set [36]. Similarly, Maeda et al. introduce a mechanism for deciding whether to rely on previously learned skills (represented as Gaussian Process motion primitives) to accomplish a task, or to signal learning of a new skill [48]. While these approaches employ a similar idea of managing a set of models via adding, removing, and updating mechanisms, they do not apply directly to the simultaneous mode and state estimation problem that we address in this chapter.

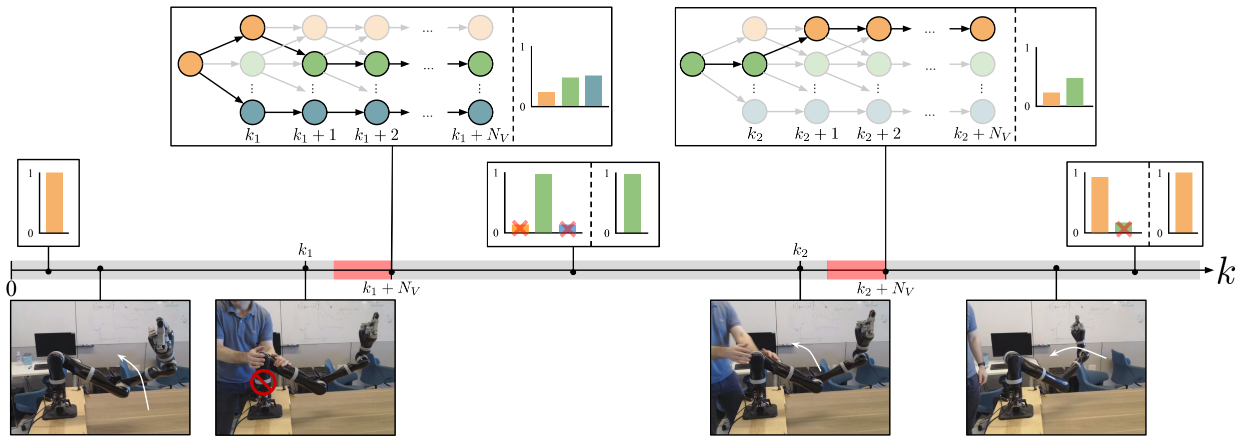

Figure 2.2: An example depicting the operation of our AMS estimator on the Kinova JACO 7 DOF manipulator system. As in an experiment in Sec. 2.6, the end-effector follows a position trajectory. The person immobilizes one of the robot's joint at time step k_1. By time step $k_1 + N_V$, the algorithm detects that the current model set is insufficient to explain the encoder measurements– so it expands the model set appropriately. Later, the algorithm removes the models that are no longer needed to explain the measurements. At time step k_2, the person lets go of the robot, allowing the joint to move freely again.

Adaptive model set

In some cases, such as when computational resources are constrained and the number of possible modes is high, it is desirable to *adapt* the model set over time. Estimators that choose a model set from an existing set of models are generally referred to as Variable Structure Multiple Model (VSMM) algorithms [83, 41]; our approach resides in this category. One approach is to frame the questions around how to adapt the model set as statistical hypothesis tests [67, 45]. However, we need to perform 2^N of these, which is computationally prohibitive. To address this, certain methods leverage structure in the system, such as the Model-Group Switching (MGS) algorithm [44, 43]. In many robotics settings, however, we often cannot assume such structure (e.g. sensing and actuation faults are unpredictable). Our proposed approach, in contrast, scales *linearly* (rather than *geometrically*) with N, and does not assume any structure on the mode switching dynamics.

2.2 The Adaptive Model Set (AMS) Algorithm

Overview

Our proposed Adaptive Model Set (AMS) estimator applies the IMM algorithm with the addition of a model set update via Alg. 1 at each step. We wish to use only a small subset of all possible models at each time step in order to mitigate computational expense.

When to expand the model set

Our key idea is to expand the current model set only if the current models are insufficient to explain the sensor measurements. We implement this in line 8, which checks a threshold on the measurement likelihood

$$p(y_k \mid y_{0:k-1}, m_k = i) > \beta.$$

If *none* of the current models can explain the current measurement, then Alg. 1 expands the model set.

Majority voting

Instead of deciding to expand the model set on the basis of a single measurement, our algorithm considers a majority vote over this decision across N_V time steps (lines 8-13). For example, in Fig. 2.2, the algorithm expands the model set only after a majority of votes between time step k_1 and $k_1 + N_V$ agree that it is necessary to do so. In Fig. 2.2, grey indicates a vote *against* adding models, and red indicates a vote *for* adding models at that time step.

If $N_V > 1$, then the system may have switched modes at some point during the past N_V time steps. To account for this, we enumerate all mode sequences of length N_V, illustrated by the trees of mode sequences at time steps $k_1 + N_V$ and $k_2 + N_V$ in Fig. 2.2. Then, for each mode sequence M, we instantiate a filter to compute the state estimate at time step k conditioned on mode sequence M between time steps $k - N_V + 1$ and k. To do so, we instantiate the filter with the state estimate at $k - N_V + 1$ (line 17), and then for each time step up until k, we update the filter with the input and measurements that were received at that time step (lines 18-20). In order to perform these updates, we must store u_k, y_k, as well as some additional information about the estimates, for the last N_V time steps (e.g. for a Kalman filter, we would store the previous state estimate and estimation error covariances).

While we enumerate all mode sequences of length N_V, there exist several possible optimizations. For example, we could assume that the system switched modes only once during the voting period.

Which models to add

Once it has constructed all candidate filters, Alg. 1 selects the best performing models, with respect to an *evaluation function* q. This evaluation function is specific to the type of filter that we are using, and provides a measure of how well the filter predicts the current observed measurements. For example, if we are using Kalman filters, then $q(F_M)$ is proportional to the filter's *residual* (i.e. difference between predicted and observed measurement), where i is the final mode in the mode sequence M. Finally, Alg. 1 adds the best performing model

F_M^* (line 23), as well as all other filters F_M whose performance is close to F_M^* with respect to q, to the new filter set M_k (lines 24-26).

Removing models

If a model has low *a posteriori* probability $P(m_{k-1} = i \mid y_{0:k-1})$ (denoted as p_i in the pseudocode) at the previous time step $k-1$, then the algorithm removes it (lines 4-6); hence, models not needed to explain the data seen thus far are not kept in the model set. While prematurely removing a model temporarily degrades the estimator performance, if that model is important, it will simply be added again at a later time step.

Algorithm 1 Updates the model set at time step k, given the latest measurement and control input.

```
1: function AMS-UpdateModels($u_k, y_k, M_{k-1}, \{p_i\}$)
2:     $M_k \leftarrow M_{k-1}$
3:
4:     for each model $i$ in $M_{k-1}$ do
5:         if $p_i < \alpha$ then
6:             Remove model $i$ from $M_k$
7:
8:     if $p(y_k \mid y_{0:k-1}, m_k = i) > \beta$ for all $i$ in $M_{k-1}$ then
9:         Vote for adding new models at time step $k$
10:    else
11:        Vote against adding new models at time step $k$
12:
13:    Check majority vote over past $N_V$ time steps
14:    if majority vote favors adding models then
15:        filters $\leftarrow \emptyset$
16:        for each mode sequence $M$ of length $N_V$ do
17:            Construct $F_M$ at time step $k - N_V + 1$
18:            for $l = 1, 2, \ldots, N_V$ do
19:                Switch internal model of $F_M$ according to $M$
20:                Update $F_M$ with $u_{k-N_V+l}$ and $y_{k-N_V+l}$
21:            Add $F_M$ to filters
22:
23:        $F_M^* \leftarrow$ one-of $\underset{F_M \in \text{filters}}{\operatorname{argmin}} \; q(F_M)$
24:        for each $F_M$ in filters s.t. $|q(F_M^*) - q(F_M)| < \gamma$ do
25:            Add the last mode in $M$ to $M_k$
26:            Assign $F_M$ as the filter associated with mode $M$
27:    return $M_k$
```

Parameters

Threshold for removing models (α)

The parameter α (line 5) determines when to remove a model, based on the *a posteriori* probability of that particular model best representing the system mode. In our experiments, we use $\alpha = 0.01$.

Threshold for expanding the model set (β).

The parameter β is used to determine when to expand the model set (line 8), and may in fact vary over time. For example, in our experiments, since we use Kalman filters that assume Gaussian measurement likelihood distributions, we set β to be the likelihood of a measurement that is two standard deviations away from the predicted measurement.

Threshold for adding models (γ)

The parameter γ determines how the algorithm "groups" similarly performing models (line 24). An alternative approach would add only the models i where q-value is above a threshold. In the case where none of the models adequately explain the measurements with respect to this threshold, no models would be added, causing poor performance. Therefore, it is better to add the best models we have relative to one another, rather than with respect to a fixed threshold. In practice, the value of γ depends on the scale of the evaluation function q, and should typically be small.

Number of votes (N_V)

The parameter N_V determines the number of votes before expanding the model set. This parameter captures a trade-off between robustness (explored experimentally in Sec. 2.4), and computational efficiency, since the number of mode sequences is on the order of N^{N_V}.

2.3 Evaluation in Simulation

Overview

We first evaluate if our AMS estimator enables high estimation accuracy, while decreasing the computational load. We evaluate our estimator on 2 simulated systems: a planar 3 DOF manipulator, and a 6 DOF manipulator, proposed for a future NASA lander mission to Europa [25] (see Fig. 2.3).

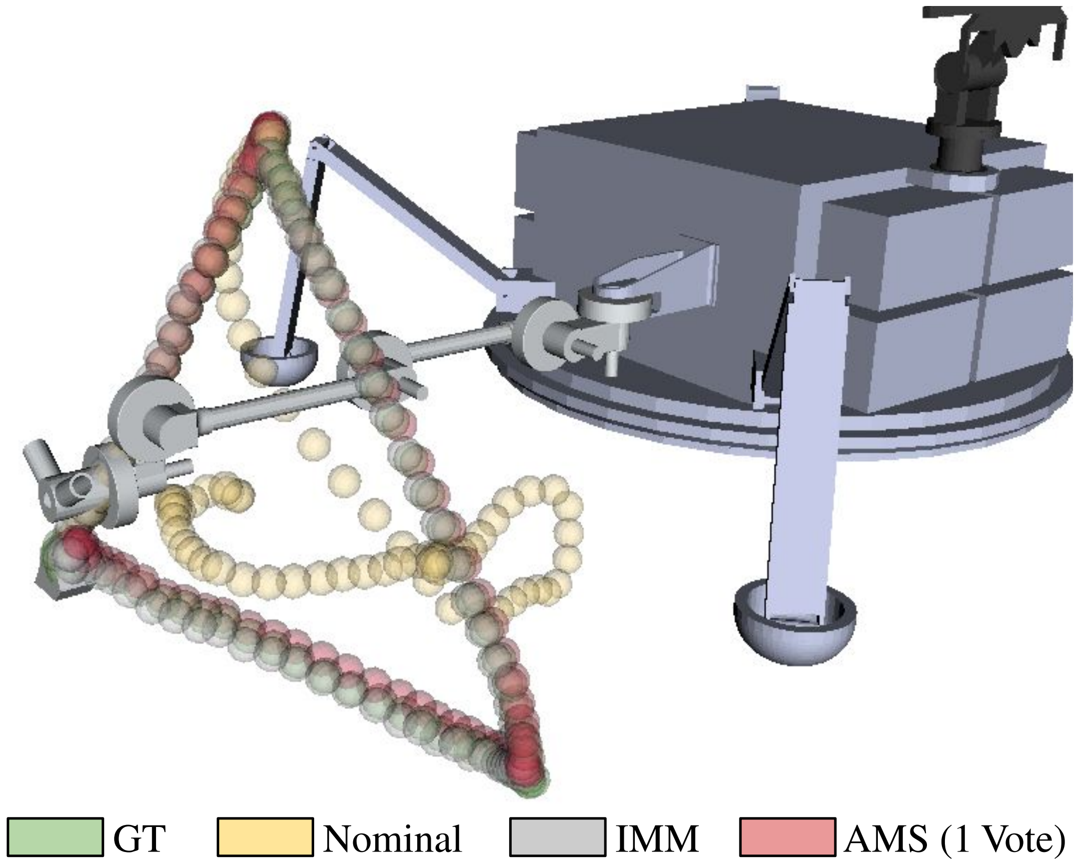

Figure 2.3: NASA 6 DOF Europa lander arm, tracking a position trajectory for the end-effector (the scoop). The actuator on the second joint suffers a temporary 75% degradation. Using the nominal model for controlling the manipulator leads to a trajectory that diverges from the reference, since controller is unable to compensate for the degradation.

Experimental Setup

Simulated Behavior

Each experiment is 3.1 s long, with a time step of 0.01 s. The system starts in mode *nominal*, and switches to another modes after 0.75 s, and then switches back to *nominal* after 1.75 s.

Behavior Modes

We consider models of the following behavioral modes for the 3 DOF and 6 DOF manipulators.

1. *Nominal:* Manipulator is operating normally. We apply a first-order Euler discretization in time to the manipulator equations (see [56]) to derive an equation of the form in Eq. (2.1).
2. *Locked joint:* A single joint is completely immobile.
3. *Free-swinging joint:* All input torque at the free-swinging joint is set to zero.
4. *Degraded actuator:* The input torque at the degraded actuator is multiplied by a scalar *degradation factor* in the interval $(0, 1)$. We consider degradation factors of $0.25, 0.50$ and 0.75.

Each of these modes, other than the nominal mode, can occur at each joint; furthermore, we consider a single model for each behavioral mode. Therefore, to count the total number of models, we multiply by the number of joints. So, in these experiments, we consider 17 possible models for the 3 DOF arm, and 32 models for the 6 DOF arm.

Control Objective

We consider two control objectives:

1. a *jointspace* tracking task, where the goal is for the manipulator to follow a time-varying, sinusoidal sequence of joint positions, velocities, and accelerations;
2. and, a *taskspace* tracking task, where the goal is for the end-effector to follow a time-varying sequence of positions and velocities in taskspace.

Control Law

For both control objectives, we use a Computed Torque Control (CTC) law [56]. We design the CTC input based on the model with highest *a posteriori* probability at each time step.

Independent Variables

We manipulate whether we adapt the model set or keep it fixed; this leads to three non-adaptive baselines to compare with our adaptive estimator.

1. *Ground-Truth (GT):* A Kalman filter using the ground-truth model at each time step.
2. *Nominal (N):* A Kalman filter using the *nominal* internal model.
3. *Interacting Multiple Model (IMM):* The IMM algorithm described in Sec. 2.1. Note that the model set includes all possible models at each time step.
4. *Adaptive Model Set (AMS):* Our proposed AMS estimator, described in Alg. 1. For the AMS algorithm, we also manipulate N_V.

Dependent Measures

We consider the following dependent measures.

1. *Mode Prediction Accuracy:* The percentage of time steps for which the estimator correctly predicts the system mode. Here, the estimator predicts the mode with maximum *a posteriori* probability.
2. *State Estimation Error:* Defined for each time step to be the norm of the difference between the actual state and the estimated state.
3. *Position/Velocity Tracking Error:* Defined as the norm of the difference between the desired position/velocity and the actual position/velocity. For the jointspace control objective, position/velocity is the joint angles/velocities. For the taskspace control objective, position/velocity is the end-effector position/velocity.
4. *Estimator Update Time:* Defined for each time step as the amount of time (in seconds) needed to update the state estimate. All measures reported in the tables in this chapter are mean values, with standard error reported in parenthesis.

Hypotheses

H1 Adapting the model set, via our AMS algorithm, performs better with respect to computation time than a non-adaptive estimator.

H2 Adapting the model set, via our AMS algorithm, performs at least as well as a non-adaptive estimator with respect to state estimation and trajectory tracking error.

Trials

For each algorithm, we conduct 50 trials, each with a different random seed, every combination of manipulator, true mode the system switches to, and control objective.

Analysis

For the 3 DOF and 6 DOF systems, our AMS estimator is faster than the IMM estimator on both control objectives, supporting *H1*. Fig. 2.4 shows the update times for the 6 DOF experiments, and the 3 DOF experiments follow the same trend. Our estimator also performs similar or better than the IMM estimator with respect to estimation and tracking errors on both control objectives, shown in Table 2.1, in support of *H2*. We note that AMS *outperforms* IMM in mode prediction accuracy, shown in Fig. 2.4.

We observe no effect of voting for the 3 DOF manipulator, in that the performance of our estimator with and without voting is comparable; however, in the 6 DOF manipulator experiments, we see about a 10% increase in mode prediction accuracy when using our estimator, compared to using the IMM estimator. This is likely because voting is designed to help when there is ambiguity among which model best describes the system mode. These ambiguities are more prone to occur in the 6 DOF experiments, where there are 32 possible models, compared to in the 3 DOF experiments where there are only 17 possible models.

We also observe that all estimators perform better on the jointspace objective than on the taskspace objective. In the jointspace objective, the manipulator follows a sinusoidal trajectory; it is likely that this trajectory more persistently excites the system when compared to the taskspace trajectory, leading to better estimator performance.

Summary

We find that our AMS algorithm is more computationally efficient than baselines, while not compromising on performance. In some cases our method outperforms baselines, most notably in mode prediction accuracy. This is due in part because our AMS estimator provides an additional layer of filtering: we only increase the model set when the existing models are poorly explaining the measurements. This prevents the maximum *a posteriori* probability model from switching as frequently as in the IMM estimator.

2.4 Robustness to Misspecified Models

Overview

Our first experiment analyzed the performance of our AMS algorithm in situations where the models perfectly describe the dynamics of the system. Next, we verify that working with an adaptive model set, rather than the full one as in the IMM algorithm, does not negatively affect accuracy in situations where the models are imperfect: where the noise model is inaccurate, where the link masses are imperfectly characterized, or where we are missing a model of the correct mode altogether. We keep the same 3 DOF manipulator system, as well as the same models, control objective, control law, trials, and dependent measures as in Sec. 2.3.

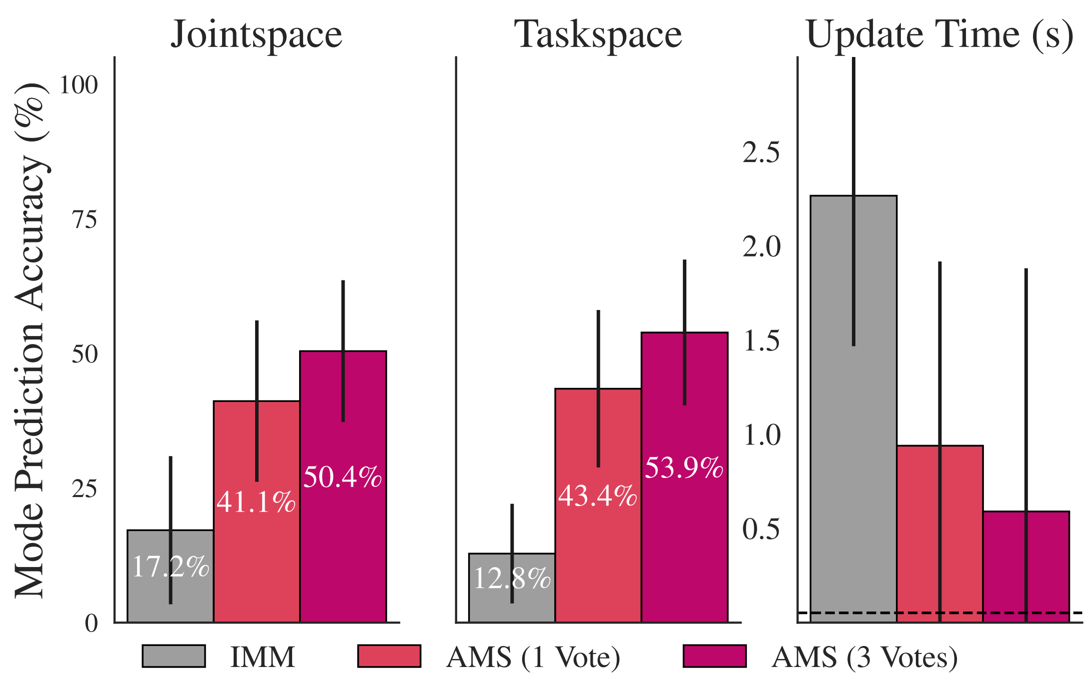

Figure 2.4: Performance comparison of an IMM estimator versus our AMS estimator for the 6 DOF arm. Error bars indicate standard deviation.

3DOF Manipulator				
	Estimation Error		Tracking Error	
	Joint (rad)	Task (m)	Joint (rad)	Task (m)
N	1.178 (4.7e-3)	0.202 (7.1e-4)	1.008 (4.6e-3)	0.070 (2.7e-4)
IMM	0.047 (4.8e-5)	**0.052 (5.2e-5)**	0.093 (3.0e-4)	**0.019 (7.2e-5)**
AMS	**0.045 (5.1e-5)**	0.053 (6.3e-5)	**0.082 (3.0e-4)**	0.020 (1.1e-4)
GT	*0.044 (4.8e-5)*	*0.044 (4.7e-5)*	*0.074 (3.0e-4)*	*0.012 (6.3e-5)*

Table 2.1: 3 DOF manipulator, all models available to estimation algorithm, position estimation and tracking errors. Mean and standard error in parentheses.

Experimental Setup

Independent Variables

In addition to the estimation algorithms described in Sec. 2.3, we further manipulate the following three variables in separate experiments to evaluate robustness: the standard deviation of the *simulated* sensor noise (in this case $\sigma_v^2 = 0.03$) versus the standard deviation of the *modeled* sensor noise (in this case $\sigma_v^2 = 0.01$); whether the link masses are mismodeled (i.e. the actual simulated masses are perturbed randomly in the range $[-0.2, 0.2]$ kg from their modeled values used by the estimator); and, whether or not the estimator has access to a model of the mode to which the system switches.

Hypotheses

H3 Our AMS estimator does not lead to further degradation in performance when faced with (a) mismodeling of the measurement noise; (b) mismodeling of the link masses; and (c) a completely unknown system mode.

Analysis

For $N_V = 1$, while the errors for our estimator are in some cases greater than for the IMM estimator, it is never by more than 1 degree on average, as shown in Table 2.2, 2.3, and 2.4. Increasing N_V greater than 1 showed no effect on performance for the second and third experiments, and hence the results are not shown in the respective tables. For the first experiment, however, increasing N_V to $3, 5$ and 7 provided additional robustness to the mismodeling of sensor noise as measured by mode prediction accuracy, as shown in Fig. 2.5. Despite this, there is not necessarily an improvement in performance from the perspective of estimation and tracking errors in Table 2.2. Furthermore, the geometric complexity in N_V of the model set expansion starts to have an effect when N_V increases to 5 and 7 votes. Overall, we find that the performance of our AMS estimator is similar to the performance of the IMM estimator, in support of *H3*.

Summary

Our experiments indicate that our estimation algorithm exhibits reasonable robustness to mismodeling when compared to the IMM estimator. Furthermore, our experiments provide evidence that voting may provide robustness to mismodeling of sensor noise; however, we also observe that for $N_V \geq 5$, the computational advantages of our estimator compared to IMM estimation begin to diminish.

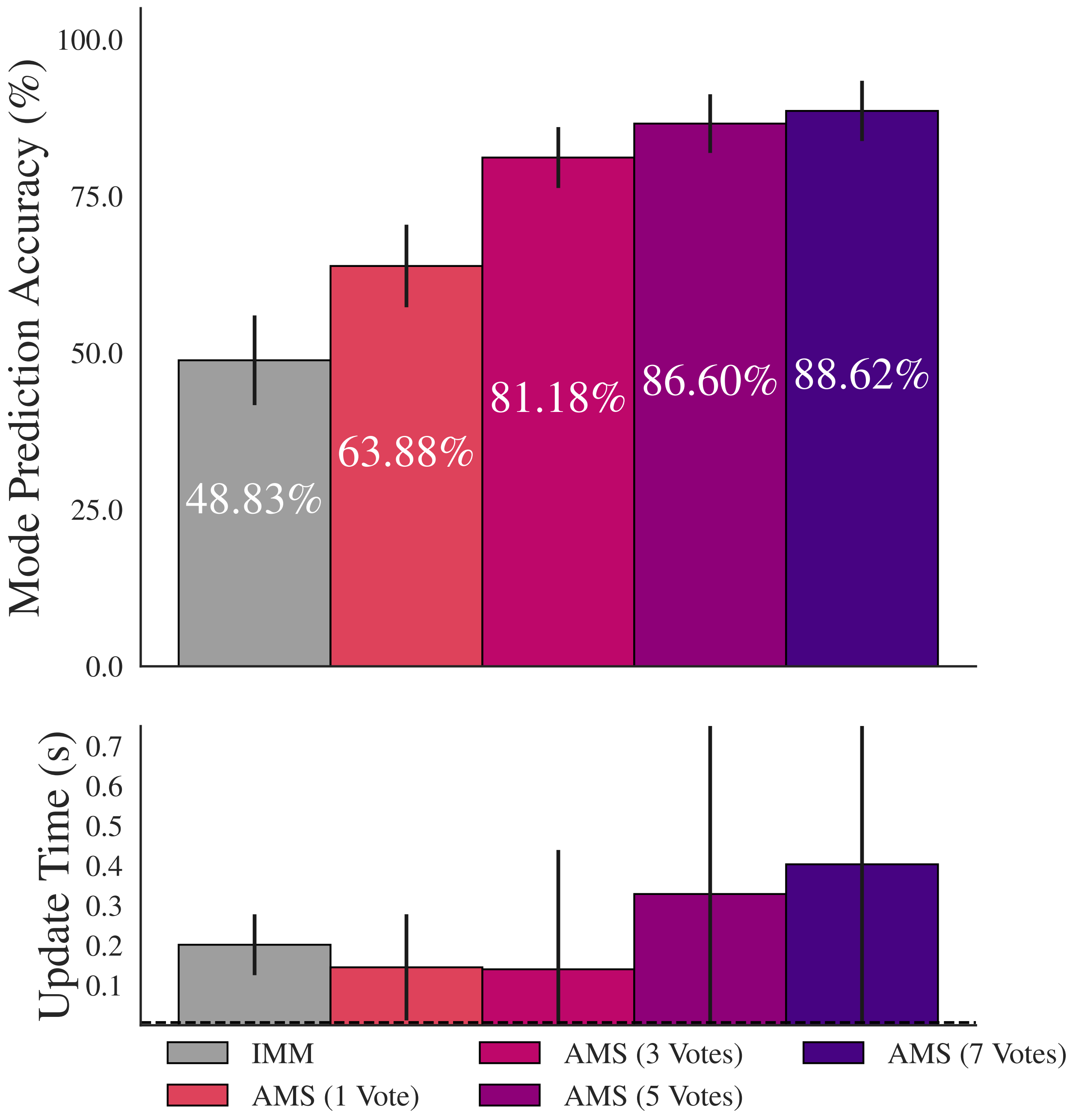

Figure 2.5: Evaluating the robustness of our AMS estimator versus an IMM estimator, when sensor noise is mismodeled. The sensor is modeled as having noise with $\sigma_v^2 = 0.01$, whereas it is simulated with $\sigma_v^2 = 0.03$ (i.e. sensor is noisier than modeled). Error bars indicate standard deviation.

3DOF Manipulator, Mismodelled Sensor Noise (H3 (a))				
	Estimation Error		Tracking Error	
	Joint (rad)	Task (m)	Joint (rad)	Task (m)
IMM	0.090 (1.45e-4)	**0.084 (9.01e-5)**	0.137 (3.19e-4)	0.026 (7.33e-5)
AMS1	0.095 (1.71e-4)	0.095 (1.10e-4)	**0.135 (3.67e-4)**	0.023 (7.24e-5)
AMS3	**0.086 (1.65e-4)**	0.091 (1.07e-4)	0.146 (5.60e-4)	**0.019 (7.65e-5)**
AMS5	0.090 (3.74e-4)	0.090 (1.21e-4)	0.180 (1.21e-3)	**0.019 (8.20e-5)**
AMS7	0.096 (6.15e-4)	0.090 (1.60e-4)	0.199 (1.66e-3)	0.020 (8.61e-5)

Table 2.2: Experiment for H3 (a): 3 DOF manipulator, sensor is noisier than modeled in estimation algorithm (position).

3DOF Manipulator, Incorrect Link Masses (H3 (b))				
	Estimation Error		Tracking Error	
	Joint (rad)	Task (m)	Joint (rad)	Task (m)
IMM	**0.060 (1.02e-4)**	**0.057 (6.39e-5)**	**0.100 (3.01e-4)**	**0.020 (7.16e-5)**
AMS	0.070 (1.01e-4)	0.069 (8.07e-5)	0.104 (3.02e-4)	0.021 (6.56e-5)

Table 2.3: Experiment for H3 (b): 3 DOF manipulator, link masses are slightly different than model used by estimator (position).

3DOF Manipulator, Completely Unknown Mode (H3 (c))				
	Estimation Error		Tracking Error	
	Joint (rad)	Task (m)	Joint (rad)	Task (m)
IMM	**0.113 (3.86e-4)**	**0.075 (2.14e-4)**	0.267 (9.98e-4)	0.035 (1.67e-4)
AMS	0.124 (5.16e-4)	0.084 (2.73e-4)	**0.252 (1.03e-3)**	**0.028 (1.37e-4)**

Table 2.4: Experiment for H3 (c): 3 DOF manipulator, estimator has no access to model for system mode (position).

2.5 Domain Generalization

So far, our experiments have been restricted to manipulator systems with mode switching caused by some change in the robots own *internal* dynamics model. In fact, many interesting switching dynamics arise from changing dynamics *external* to the robot– such as varying environmental conditions or the changing behavior of other agents. For example, a vehicle driving over dirt experiences a change in dynamics that it must compensate for if it suddenly encounters an icy patch on the road, as shown in Fig. 2.1 (center). Similarly, a robot operating around a human must adapt when the person changes their intention, as shown in Fig. 2.1 (right). The following 2 experiments evaluate the performance of our AMS estimator in such situations.

Driving on Variable Terrain

Vehicle Model

We use a kinematic, skid-steering model of the Robotnik Summit XL robot, shown in Fig. 2.1. The state is the position, orientation, and linear and angular velocities of the robot with respect to the world frame. The input is a commanded velocity for each of the 4 wheels.

Modes

For each terrain, we model the effect of friction by setting $v_{k+1} = \gamma v_k + (1-\gamma)v_{k,in}$, where v_{k+1} is the vehicle velocity at the next time step $k+1$, v_k is the vehicle velocity at the current time step k, and $v_{k,in}$ is the commanded velocity, which we convert from the commanded wheel velocities via the kinematics model. Finally, $\gamma \in [0,1]$ is a coefficient represents the amount of friction.

1. *Dirt.* We model friction having negligible effects, so $\gamma = 0$.

2. *Sand.* We model the friction with $\gamma = 0.01$.

3. *Ice.* We model the friction with $\gamma = 0.001$.

4. *Shallow Mud.* Here we set $\gamma = 0$; however, we model the vehicle being stuck in mud if its velocity is below 0.3 m/s.

5. *Deep Mud.* $\gamma = 0$ as in the model of shallow mud, but here the vehicle is stuck if its velocity is below 0.6 m/s.

6. *Single Wheel Stuck.* Here we set $\gamma = 0$; however, to model a single wheel being stuck, we simply zero out that wheels commanded velocity.

Simulated Behavior

We run each experiment for 1000 time steps, sampled every 1/30 s. The vehicle begins on the *dirt* terrain. After 3 m, the terrain switches to *ice*.

Control Objective

We designed a 5 m straight-line reference trajectory in both position and velocity for the vehicle to track. The vehicle starts and ends at zero velocity.

Local Planner

In order to track the reference trajectory, we employ a *local planner*, which performs a Dijkstra search to find a sequence of control inputs (or, local plan) that minimizes tracking error and control efforts, with respect to the model provided to it. We employ a horizon of 5 time steps with a time step of 0.25 s. At each time step, we re-run the local planner and execute the first input in the plan it returns. The local planner uses the model with maximum *a posteriori* probability from the estimator.

Independent Variables

We consider the same independent variables as in Sec. 2.3:

1. *Ground-Truth (GT):* A Kalman filter using the ground-truth model at each time step.
2. *Nominal (N):* A Kalman filter using the *nominal* internal model.
3. *Interacting Multiple Model (IMM):* The IMM algorithm described in Sec. 2.1. Note that the model set includes all possible models at each time step.
4. *Adaptive Model Set (AMS):* Our proposed AMS estimator, described in Alg. 1. For the AMS algorithm, we also manipulate N_V.

Dependent Measures

We measure the mode prediction accuracy, the state estimation error, the position and velocity tracking errors, and the estimator update time, the same as in Sec. 2.3's experiment:

1. *Mode Prediction Accuracy:* The percentage of time steps for which the estimator correctly predicts the system mode. Here, the estimator predicts the mode with maximum *a posteriori* probability.
2. *State Estimation Error:* Defined for each time step to be the norm of the difference between the actual state and the estimated state.

Driving on Variable Terrain				
	Mode Pred. Accuracy (%)	Update Time (ms)	Track. Err. Pos. (m)	Track. Err. Vel. (m/s)
N	34.0 (0.02)	22 (0.1)	0.41 (0.019)	0.11 (0.003)
IMM	79.0 (1.9)	269 (1)	1.4 (0.252)	0.25 (0.023)
AMS	**99.3 (0.11)**	**96 (0)**	**0.03 (0.016)**	**0.03 (0.002)**
GT	*100 (0.0)*	*31 (0)*	*0.03 (0.015)*	*0.02 (0.002)*

Table 2.5: Comparison of performance in the driving domain.

3. *Position/Velocity Tracking Error:* Defined as the norm of the difference between the desired position/velocity and the actual position/velocity. For the jointspace control objective, position/velocity is the joint angles/velocities. For the taskspace control objective, position/velocity is the end-effector position/velocity.

4. *Estimator Update Time:* Defined for each time step as the amount of time (in seconds) needed to update the state estimate. All measures reported in the tables in this chapter are mean values, with standard error reported in parenthesis.

Hypotheses

We assume the same hypotheses as in Sec. 2.3's experiment:

H1 Adapting the model set, via our AMS algorithm, performs better with respect to computation time than a non-adaptive estimator.

H2 Adapting the model set, via our AMS algorithm, performs at least as well as a non-adaptive estimator with respect to state estimation and trajectory tracking error.

Trials

For each algorithm, we conduct 20 trials, each with a different random seed.

Analysis

The results summarized in Table 2.5 support both *H1* and *H2*. We see that our AMS estimator outperforms the IMM and nominal estimators with respect to accuracy (both in mode prediction and state estimation) and performance (in tracking the reference trajectory).

Summary

We find that the AMS estimator works well when facing changes in dynamics arising from factors *external* to the dynamics of the vehicle– namely a change of terrain.

Human Motion Prediction and Robot Navigation

Human Model

Each mode corresponds to a goal location in the environment that the human could walk to. The estimator uses a model of the human in the form of (2.1), with

$$f_i(k, x_k, u_k) = \begin{cases} p_k^x + Tv\cos(u_{i,k}), \\ p_k^y + Tv\sin(u_{i,k}) \end{cases}$$

where the $u_{i,k}$ is the angle along a straight-line path from the persons current position to the g_ith goal position: $u_{i,k} = \tan^{-1}(\frac{p_k^y - g_i^y}{p_k^x - g_i^x})$. T is the time step of the simulation. In the measurement model we have $g_i(x_k) = x_k$. There is no process noise but the measurement noise has covariance $R_i = \mathrm{diag}(0.05^2, 0.05^2)$. The indoor environment occupancy map is from [60] and the map and the possible 20 human goals are shown in Fig. 2.1.

Robot Model

We model the robot as a 3D Dubins car vehicle, with control inputs being the speed and steering angle. The robot has a fixed start and goal state across all trials and uses a spline-based motion planner [82]. At each time step, the robot makes a prediction of the human's goal, taking the maximum *a posteriori* probability goal from the estimator. The robot subsequently predicts the human's trajectory, simply extrapolating forward in time by applying a nominal velocity for a number of time steps into the future. We use this predicted trajectory for robot collision checking.

Simulated Behavior

For each trial, the human starts at the same position, and follows the same mode switching sequence: goal 18 for the first 7 time steps, then goal 14 for the next 8 time steps, then goal 11 for the next 6 time steps, and finally goal 23 for the remainder of the trial (see Fig. 2.1). We run each trial for 38 time steps ($\sim$ 15.2 s).

Independent Variables

We manipulate the estimation algorithm, the same as in Sec. 2.3's experiment.

Dependent Measures

We measure the mode (i.e. goal) prediction accuracy, the state estimation error, and the estimator update time, the same as in Sec. 2.3's experiment. Furthermore, to capture the *safety* of the robot's path, we measure the minimum distance between the robot and the human for the duration of each trial.

Human Goal Prediction & Robot Navigation				
	Mode Pred. Accuracy (%)	Update Time (ms)	Safety (m)	Estimation Err. (m)
N	0 (0)	2 (0)	0.19 (0)	3.50 (0.04)
IMM	59.5 (1.4)	61 (1)	0.30 (0.04)	**0.05 (0)**
AMS	**66.0 (2.8)**	**30 (1)**	**0.32 (0.05)**	0.06 (0.01)
GT	*100 (0)*	*2 (0)*	*0.28 (0.01)*	*0.0 (0)*

Table 2.6: Comparison of performance in the robot navigation domain. Mean values reported with standard errors in parenthesis.

Hypothesis

We assume the same hypothesis as in Sec. 2.3's experiment.

Trials

For each algorithm, we conduct 20 trials, each with a different random seed.

Analysis

We summarize the results in Table 2.6. Our AMS estimator is about twice as fast to update, when compared to IMM, in support of *H1*. It also achieves comparable performance on all other measures when compared to the IMM estimator, in support of *H2*. In fact, we observe that the AMS is actually about 6% more accurate at predicting the human's goal than IMM, consistent with our findings in the other experiments.

Summary

We confirm that the AMS estimator is computationally faster than baseline estimators, and show that it can be employed in settings where the switches arise from the behavior of other agents in the environment.

2.6 Evaluation in Hardware

We further evaluate our estimation algorithm in hardware experiments on a Kinova JACO 7 DOF manipulator, shown in Fig. 2.1. We input desired velocity commands $u_k \in \mathbb{R}^7$ to the manipulator, and have access to measurements of the joint positions $x_k \in \mathbb{R}^7$ via encoders at each of the joints, represented by the output $y_{k+1} = x_{k+1} + v_{k+1}$, where v_{k+1} is Gaussian noise, as in Sec. 2.1.

Modes

Nominal

The nominal dynamics model is $x_{k+1} = x_k + Tu_k + w_k$.

Locked joint

Let L_i be a diagonal matrix where the ith diagonal entry is 0, and all others are 1. Then the dynamics model is $x_{k+1} = x_k + TL_i u_k + w_k$.

End-effector contact.

Let $J(x_k) \in \mathbb{R}^{3\times 7}$ be the Jacobian of the end-effector position forward kinematics mapping; then the linear velocity of the end-effector at time step k is $v_{\mathrm{e},k} = J(x_k)u_k$. Furthermore, let $J^\dagger(x_k) \in \mathbb{R}^{7\times 3}$ be the Moore-Penrose pseudo-inverse of $J(x_k)$. Consider a contact with surface normal $\hat{n} \in \mathbb{R}^3$. Then the dynamics model is $x_{k+1} = x_k + Tu_{0,k} + w_k$ where

$$u_{0,k} = \begin{cases} J^\dagger(x_k)(I - \hat{n}\hat{n}^\top)v_{\mathrm{e},k} & \text{if } \hat{n}^\top v_{\mathrm{e},k} < 0 \\ u_{0,k} = u_k & \text{otherwise.} \end{cases} \tag{2.4}$$

Control Objective

In the first experiment, the JACO follows an end-effector position trajectory. In the second experiment, the JACO follows an end-effector pose (position and orientation) trajectory, sliding a grasped object across a table. In the third experiment, the JACO follows an end-effector position trajectory, specified by a planner whose task is to have the end-effector reach two position goals in the robot's workspace, A and B.

Control Law

Let $X_k, X_k^d \in \mathbb{R}^{4\times 4}$ be the homogeneous transformation matrices representing the actual and desired end-effector poses with respect to a common inertial reference frame, respectively. Let $V_{e,k} \in \mathbb{R}^6$ be the twist with matrix representation given by $X_k^{-1}X_k^d$. Then the tracking control law is

$$u_k = J^\#(x_k)K_p V_{e,k}$$

where K_p is a positive-definite diagonal proportional gain matrix, where $J^\#(x_k) \in \mathbb{R}^{6\times 7}$ is the damped least-squares pseudoinverse of the body manipulator Jacobian.

Procedure

For the first and third experiments, we simulated a locked joint by physically holding that joint in place. For the second experiment, we set the reference trajectory to pass through the table, to ensure contact. In all experiments, we use our AMS algorithm for state estimation.

Analysis

In the first and third experiments, we find that our estimator is able to reliably identify the locked joint in about 1 s. Once the robot stopped at the end of the reference trajectory, we noticed that the estimator assigned about 50% probability to both the nominal and locked modes– this is expected, since it is not possible to distinguish between these modes when no velocity is commanded at the joint. In the second experiment, we found that our estimator took longer to identify the contact– between 3 to 5 s. This is partially due to certain unmodeled factors occurring during contact, including frictional effects such as stiction, as well as motion of the object in the gripper.

Summary

Here, we showed the applicability of AMS to predict mode changes on a hardware manipulator system.

2.7 Discussion

Summary

Robotic systems frequently operate under changing dynamics; to do so effectively requires a good estimator for both the continuous state and dynamics mode. In this work, we propose such an estimator that adapts the set of models at each time step based on a novel algorithm. We provide thorough experimental evaluation in simulation and hardware, to demonstrate the algorithm's effectiveness for state estimation under changing dynamics– including actuation faults, driving over various terrain, and navigating around a human with uncertain and changing intent.

Limitations and Future Work

In future work, we plan to explore "active" algorithms that design control inputs to improve estimates and combine online parameter adaptation with our adaptive model set algorithm. Furthermore, our method is limited to a set of *a priori* specified models; however, adaptive estimators are amenable to learning models online. We discuss these ideas further in Chapter 5.

Part II

Avoiding Unreliable Regions of a Model in Planning

Chapter 3

Bias Planning Away from Model Inaccuracies using Control-Level Information

Robots rely on models to plan intelligent behaviors and complete tasks. Therefore, robots require models that accurately predict the outcome of actions in the real-world. Unfortunately, however, regardless of how we construct a model — from first-principles, through data-driven methods, etc. — it will never be accurate everywhere. An important challenge, therefore, is how to enable robots to complete tasks despite using inaccurate models.

For example, consider a robot navigating to a goal, as shown in Fig. 3.1. Here, the robot neglects to model the fan, which then pushes it off of the planned path into the obstacle. The robot's propulsion is insufficient to overcome the fan, and therefore it must find an alternate path to the goal.

Finding a better path to the goal, however, requires changing how the planner reasons in some way to account for the fan. One possibility is to learn a new model, using observations from the robot's actual execution, through model learning [57], model-based reinforcement learning [63], or learning-based control [21]. With limited data and difficult-to-model phenomena such as aeorodynamics, however, this can be very challenging. Moreover, it is often unnecessary for completing the task– in Fig. 3.1's example, it is possible for the robot to take a longer path around the other side of the obstacle to avoid the fan altogether and still reach the goal. Finding this path does not require a detailed model of the fan's effect on the robot.

Motivated by this, our work focuses on biasing the planner away from regions where the original model has proven to be inaccurate, rather than learning a completely new model. A challenge, however, is *how* to informatively bias the planner, using only the data observed during execution.

To address this, note that any observed performance of a path following controller provides us with valuable information about where the model is likely inaccurate. From the robot's actual path in Fig. 3.1 (Right, solid line) compared to its planned path (Left, dotted

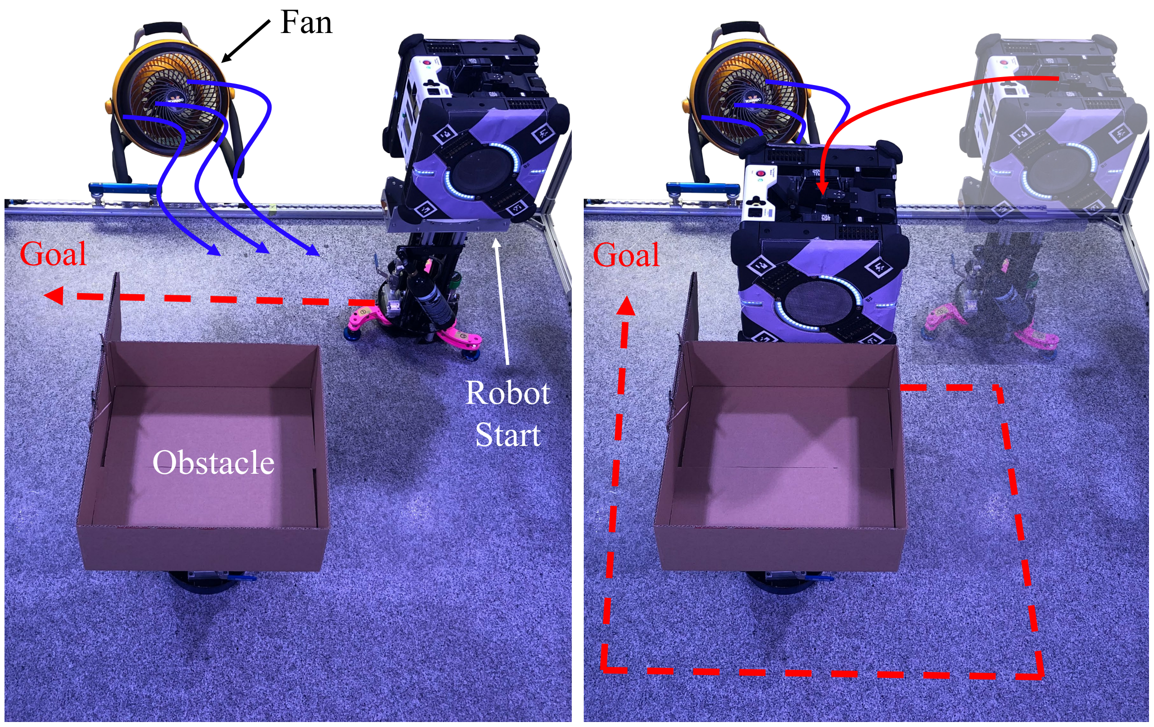

Figure 3.1: (Left) The robot plans a path to the goal through a fan's strong air current not captured by its model. (Right) To avoid the fan, the robot must plan an alternate path.

line), we can infer that the model is not correctly predicting action outcomes in this region; furthermore, the inputs that the controller applies while failing to follow the plan provide even more fine-grained information about *how* the predictions are wrong.

Our key idea is to infer a statistical model over discrepancies between the controller's model and the real-world, using this information. We then translate this discrepancy model to the planning-level, by inflating the cost of states and actions with a *control-level penalty*. To compute this penalty for a state and action, we forward-simulate the behavior of the controller when following this action, taking into account the control-level discrepancy model, and compute a probability over whether the final state differs by more than a small constant from the outcome predicted by the planner's model. The control-level penalty is then proportional to this probability.

In this chapter, we contribute:

1. a principled approach for integrating planning and control to enable robots to complete tasks despite using inaccurate models;

2. an experimental analysis of our approach in simulation, which demonstrates a signifi-

cant improvement in efficiency over baseline approaches;

3. and, a demonstration of our approach on NASA's Astrobee free-flying robot [10] in the lab.

This chapter is based on research in collaboration with Claire Tomlin, and Maxim Likhachev.

3.1 Related Work

We focus on how a robot can complete a task despite using a model that is inaccurate in ways unknown prior to execution. Our approach changes the planner's behavior but keeps the model unchanged, in contrast to adjacent research in, for example, model learning [57], model-based reinforcement learning [63], and learning-based control [21].

Planning with inaccurate models

To bias the planner away from inaccurate parts of the model, Vemula et al. [81] introduce a penalty, or additional cost, at states and actions where the model's predictions were observed to be inaccurate during execution. [80] extends this to incorporate model-free learning to improve performance on repetitive tasks. Our approach differs in two key aspects. First, we maintain a probability over whether the model will poorly predict an action's outcome, and add an additional cost proportional to this probability (rather than a fixed cost, as in [81]). Second, we use control-level information to better guide where to inflate the planner's cost function.

Pan et al. [62] address a similar problem in task planning. They propose to learn probabilities over actions resulting in unexpected outcomes, over repeated executions of the task.

For some applications, we wish to trade-off model accuracy for speed of planning. For example, modelling deformable objects is computationally challenging; [51, 55] propose to use a simplified model in planning by first training a classifier to predict where discrepancies will occur in the simplified model. They then use this classifier to bias the planner away from states and actions where it predicts a discrepancy. While the idea of biasing the planner in this way is similar to our work (and to [81]), we do not assume access to a higher fidelity model of the environment, as is required for training the classifier in [51, 55].

[59] introduces a way to learn a predictor for whether an action taken by a high-level planner will be executable by a lower-level of planning or control. This requires a way to simulate executions of the higher-level actions by the lower-level, which we cannot do before actually executing the robot in the real-world.

Integrating planning and control

Many robots, from legged systems [39, 58] to autonomous vehicles [79, 78], use hierarchical planning and control systems, wherein a high-level planner generates a path for a low-level controller to follow. Typically, the planner uses a simpler model to plan faster over a longer horizon, whereas the control uses a more complex model, closer to the actual robot dynamics, over a shorter horizon. A challenge is to ensure that the plan is executable by the controller, despite differences between the models.

To address this, [14, 38, 72] perform an offline reachability analysis to characterize differences between the planning and control models. Using these precomputations of this during planning, these approaches can guarantee that the planner will avoid paths that the controller would be unable to follow. An alternate approach is to plan with funnels [11], or motion primitives that the system is guaranteed to follow up to bounded disturbances [77, 49], to ensure a feasible path.

A related approach is to increase the model complexity and replan only when the controller cannot follow the plan during execution [75, 74]. This is similar to our approach in that control-level information is used to change the operation of the planner. However, this requires having a set of models to switch to, which we do not assume in our problem setting.

Compared to these approaches, our work involves a greater amount of feedback from the controller to the planner: information about model discrepancies from control is fed back to bias the planner. Additionally, we focus on discrepancies between the robot's model and the real-world, rather than between the two models themselves. From that perspective, many of these approaches are complementary to our work.

3.2 Problem Setup

We consider a hierarchical system consisting of a planning-level and a control-level. Our approach integrates planning and control to handle discrepancies between our environment model and the real-world.

Planning-level

We use a deterministic graph-based representation of the planning problem, consisting of a state space S, action space A, a successor function $succ(s, a)$, which gives the outcome of taking action a at state s, and a cost function $c(s, a)$, which assigns a nonnegative, finite cost to taking a at s. Furthermore, we assume that all discrepancies between our model and the real-world do not change over time. This assumption precludes, for example, other agents unexpectedly moving through the environment, or static obstacles that the robot discovers to be movable during execution.

Given a start state $s_{\text{start}} \in S$, the planner finds a least-cost path to a goal in $G \subset S$; we denote this path π.

Control-level

For control, we represent the environment through a state space X (typically $\mathbb{R}^n$, with n being the state dimension), control space U (typically $\mathbb{R}^m$, with m being the control dimension), and a difference equation $x_{t+1} = f(x_t, u_t, d_t)$, which gives the outcome x_{t+1} of applying the control u_t to the system at state x_t, and t is the integer-valued time index. We assume an additional input $d_t \in D$ (typically $D = \mathbb{R}^l$), which represents the discrepancy between our model of the environment f and the real-world. We do not know d_t *a priori*.

At each time step, the controller produces a u_t so as to reduce the deviation between the robot's actual state x_t and its desired state x_t^* to the greatest extent possible. We compute the desired state trajectory $\{x_t^*\}$ by postprocessing π. We do not impose any additional constraints on the controller's structure, aside from being able to forward-simulate the controller's output for arbitrary state and desired state trajectory inputs. Specifically, given a state x_t and desired trajectory $\{x_t\}$, we must be able to simulate the output of the controller, u_t.

Finally, we require that there exists a function $\phi : S \to X$, representing the relationship between the planning- and control-level states spaces.

Control-level discrepancies

We characterize the discrepancy between the control model f and the real-world through an unknown function of state and control, $d(x, u)$, similar to the disturbance in [2]. We refer to d as the *control-level discrepancy model*.

Specifically, $d(x, u) = \begin{bmatrix} d_1(x, u) & \dots & d_l(x, u) \end{bmatrix}^\top$, where we assume that the components d_i and d_j are independent, for $i \neq j$. We then infer each d_i using Gaussian Process (GP) regression (see, for example [64]). For the remainder of this section, we use the shorthand $z = \begin{bmatrix} x & u \end{bmatrix}^\top$, to represent the stacked vector of state x and control u.

We assume each component i to be a GP with zero mean function and a squared exponential covariance function of the form

$$C(z, z') = v_1^2 \exp\left(-\frac{1}{2} \sum_{i=1}^{n+m} \frac{(z_i - z_i')^2}{w_i^2} \right) + v_0^2 \delta_{zz'} \tag{3.1}$$

where $\{w_i\}, v_0$, and v_1 are hyperparameters, which are typically computed by maximizing the marginal likelihood of the training data (and hence must be recomputed each time more training data is added).

For a new x, u not part of the data set, we denote the predicted mean and variance of d_i as $\mu_i(x, u)$ and $\sigma_i^2(x, u)$, respectively (for $i = 1, 2, \dots, l$). We can express these functions analytically in terms of the covariance function in Eq. 3.1, and the data set (see [64]).

3.3 Approach

Our approach addresses the challenge of inaccurate models by interleaving planning and execution, which we summarize in Alg. 2, and illustrate in Fig. 3.2. We first plan a least-cost path to goal π from the robot's current state s_{start} (Alg. 2, lines 2-4). We then execute the path at the control-level, and monitor the execution for deviations exceeding a threshold of δ (Alg. 2, lines 6-12). If this threshold is exceeded, we use the robot's execution history to update our control-level discrepancy model (Alg. 2, line 13). We provide more detail on each step in the following sections.

Algorithm 2 Interleaving planning and execution.

```
1:  while robot has not reached a goal do
2:      $s_{\text{start}} \leftarrow$ get robot's (planning) state
3:      $\pi \leftarrow \textsc{Plan}(s_{\text{start}})$
4:      $\{x_i^*\} \leftarrow \textsc{Postprocess}(\pi)$
5:      history $\leftarrow \emptyset$
6:      do
7:          $x_t \leftarrow$ get robot's (control) state
8:          $u_t \leftarrow \textsc{GetControl}(x_t, \{x_i^*\})$
9:          $\textsc{Execute}(u_t)$
10:         $x_{t+1} \leftarrow$ get robot's (control) state
11:         history $\leftarrow$ history $\cup \{(x_t, u_t, x_{t+1})\}$
12:     while $\|x_t - x_t^*\| > \delta$
13:     $\textsc{UpdateDiscrepancyModel}$(history)
```

Planning with control-level penalties

Given the problem in Sec. 3.2, we use A* search [27] to plan a least-cost path to goal. However, we use a *penalized* cost function (similar to [81]), of the form

$$c(s, a) + \textsc{ControlLevelPenalty}(s, a) \tag{3.2}$$

for taking action a at state s. This additional *control-level penalty* translates the control-level's discrepancy model, which we learn through observing the robot's executions in the real-world, to a planning-level bias. This bias translates discrepancy information from the control-level to the planning-level. This control-level information is more fine-grained and hence typically very informative to the planner, since the controller operates at a much higher frequency than the planner.

Alg. 3 shows how we compute the control-level penalty for action a at state s. First, FwdSimController returns the sequence of control inputs $u_1, u_2, \ldots, u_{T-1}$ that the controller would produce to follow a from state x, ignoring any model discrepancies. In Sec. 3.2,

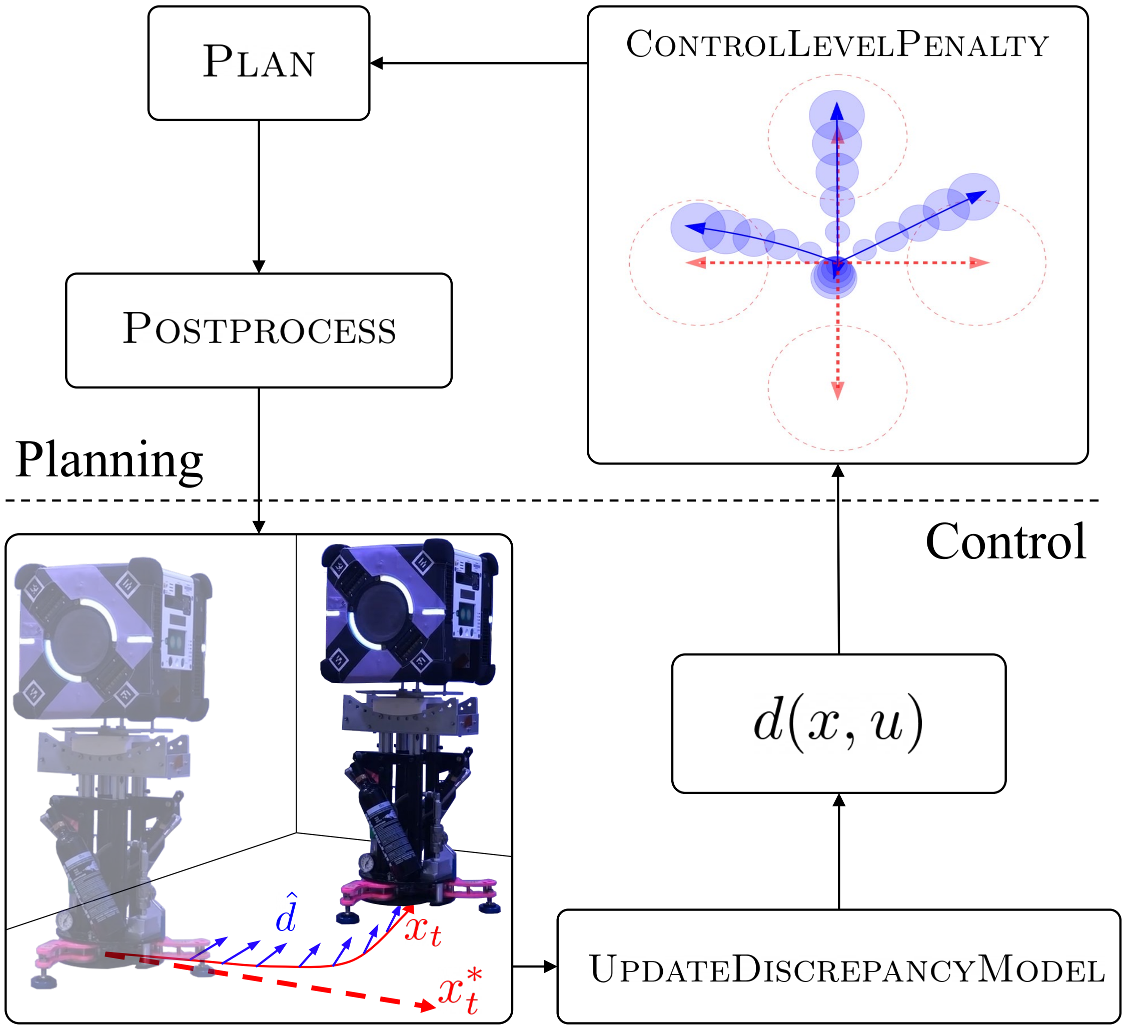

Figure 3.2: An illustration of our approach. The planning-level first generates a desired path (dashed line), which the control-level then executes. If the robot's actual path (solid line) deviates, then we update the control-level discrepancy model with the estimated discrepancies $\hat{d}$. The planner uses the new d to bias away from actions likely to result in a discrepancy, using the control-level penalty.

we remarked that there are very few constraints on the controller design; in fact, the only requirement is that we are able to forward-simulate the controller in this way to generate this sequence of inputs.

Next, we predict the state of the robot after taking the sequence of control inputs $u_1, \ldots, u_{T-1}$ from state x. Specifically, PREDICT predicts the distribution $p(x_T \mid x_1, u_1, \ldots, u_{T-1}) = \mathcal{N}(\mu_{x_T}, \Sigma_{x_T})$ over the final state x_T, given the initial state x, and sequence of controls $u_1, \ldots, u_{T-1}$. To do so, we forward propagate the uncertain discrepancy d through the dynamics, based on our latest estimate. In general, there exist many ways to do this, and we describe our particular implementation in Sec. 3.3.

Given our prediction of x_T, we can evaluate the probability that there will be a discrepancy between what the planning model predicts and the actual outcome of a. This is determined by whether x_T differs from x_T^* by δ or more, as in Alg. 3 line 5. In practice, we estimate this probability, denoted p, empirically via sampling. Finally, we return a penalty proportional to p.

Algorithm 3 Evaluating the control-level penalty.

```
1: function CONTROLLEVELPENALTY($(s, a)$)
2:     $x \leftarrow \phi(s)$
3:     $u_1, \ldots, u_{T-1} \leftarrow$ FWDSIMCONTROLLER($x, a$)
4:     $\mu_{x_T}, \Sigma_{x_T} \leftarrow$ PREDICT($x, u_1, \ldots, u_{T-1}$)
5:     $p \leftarrow P(\|x_T - x_T^*\| > \delta \mid x, u_1, \ldots, u_{T-1})$
6:     return $p \cdot c_{\text{penalty}}$
```

Prediction using the discrepancy model

Here, we describe our implementation of PREDICT. Our goal in this section is to derive a principled yet computationally simple way to predict forward in time the distribution over the state under our GP-based discrepancy model. Aside from the approach presented here, however, there exist many other methods for forward simulating using a GP model; see, for example, recent work in [29].

For simplicity, we assume that f is linear. That is:

$$x_{t+1} = Ax_t + Bu_t + B_d d_t \tag{3.3}$$

where $d_t = d(x_t, u_t)$ is the predicted discrepancy at (x_t, u_t) defined in 3.2, and A, B, and B_d are the appropriately-sized matrices. If f is not linear, we can add an additional step to the procedure below, where we linearize f around the current x and u, to find approximate A, B, and B_d.

We assume that the initial state x_1 is known. However, the state at the next time step x_2 is unknown due to the uncertain effect of d. Specifically, $p(x_2 \mid x_1, u_1)$ is Gaussian with the following mean and covariance:

$$\mu_{x_2} = Ax_1 + Bx_1 + B_d \begin{bmatrix} \mu_1(x_1, u_1) \\ \vdots \\ \mu_l(x_1, u_1) \end{bmatrix} \tag{3.4}$$

$$\Sigma_{x_2} = B_d \begin{bmatrix} \sigma_1^2(x_1, u_1) & & 0 \\ & \ddots & \\ 0 & & \sigma_l^2(x_1, u_1) \end{bmatrix} B_d^\top \tag{3.5}$$

where μ_i and σ_i^2 are the mean and variance functions described in Sec. 3.2.

Then, in general at time step t, given the mean and covariance of x_t, μ_{x_t} and Σ_{x_t}, respectively, we approximate the mean and covariance of x_{t+1} as:

$$\mu_{x_{t+1}} = A\mu_{x_t} + Bu_t + B_d\mu(x_t, u_t) \tag{3.6}$$

$$\begin{aligned}\Sigma_{x_{t+1}} &= A\Sigma_{x_{t+1}}A^\top + B_d\mathrm{Cov}(d_t, d_t)B_d^\top + \\ &\quad A\mathrm{Cov}(x_t, d_t)B_d^\top + B_d\mathrm{Cov}(d_t, x_t)A^\top.\end{aligned} \tag{3.7}$$

To arrive at Eq. 3.6 and Eq. 3.7, we use the same 1st- and 2nd-order Taylor series-based approximation approach as described in [24]; the full derivation can be found there.

We implement PREDICT by recursively computing μ_{x_t} and Σ_{x_t} for $t = 1, 2, \ldots, T$ using Eq. 3.4-3.7, and returning μ_{x_T}, Σ_{x_T}. We illustrate an example of these mean and covariance predictions in Fig. 3.2 (upper right).

Updating the discrepancy model

When the robot deviates by δ from the planned path, i.e., $\|x_t - x_t^*\| > \delta$ (Alg. 2, line 12), we update the control-level discrepancy model with the execution history. To do so, we first construct a data set $\{\hat{d}_t\}$, where

$$\hat{d}_t = x_{t+1} - f(x_t, u_t), \tag{3.8}$$

is the observed disturbance, for each (x_t, u_t, x_{t+1}) in the execution history. We illustrate these $\hat{d}_t$ by the blue arrows in Fig. 3.2 (lower left). Finally, we update our GP model $d(x, u)$ defined in Sec. 3.2, with the new data $\{\hat{d}_t\}$.

3.4 Experimental Evaluation

Setup

Planning-level

The planning state is the x- and y-position of the robot. At each state, there are 4 actions: move in the $+x$-, $-x$-, $+y$-, and $-y$-directions by 0.1 m. Each action has cost 0.1. The planner uses an A* search with a standard Manhattan distance heuristic. To postprocess the path, we fit a piecewise polynomial to it, which we then pass to the controller.

Control-level

The control state is the x- and y-position, and yaw angle θ (i.e., angle about the z-axis) of the robot. The robot is holonomic, so the control inputs are simply linear velocities in x and y, and an angular velocity about z.

Simulation

We conducted a set of simulation experiments primarily to evaluate the efficiency of our approach at completing tasks, compared to state-of-the-art approaches. For the simulation experiments, we used the Gazebo-based Astrobee simulator [22].

Methods

We compared our approach to two baselines. First, we compared with replanning, which simply replans a path to goal whenever the robot deviates too far from the plan. This is a simple, yet widely used method in hierarchical planning and control systems, described in Sec. 3.1. Second, we compared with CMAX [81], which inflates the costs of all states and actions in a small neighborhood around previously observed model discrepancies. In our experiments, we fixed this neighborhood size to 0.075 m.

Measures

To measure efficiency, we considered the total time to complete the task (this includes both the planning and execution times), and the total distance traveled. On the planning side, we measured the planning times, and the number of replans.

Scenarios

We ran 50 randomized instances of the following 3 scenarios. The environment in S1 and S2 is 2 $m \times 2$ m, and 3.6 $m \times 2.4$ m in S3.

S1 (Wire) The robot navigates to the goal, but an unmodelled wire obstructs its path, shown in Fig. 3.3. The wire is always at the origin, and we randomize its size in x ($[0.05\ m, 0.15\ m]$) and y ($[0.25\ m, 0.75\ m]$). For simplicity, we model the wire as a rigid-body.

S2 (Fan) The robot navigates to the goal, but there is an unmodelled fan in the environment, as shown in Fig. 3.4. We simulate the fan as exerting an external force on the robot when it is within a specified region near the fan. There is also an obstacle at the center of the environment. We randomize over the angle ($[10°, 45°]$) and magnitude

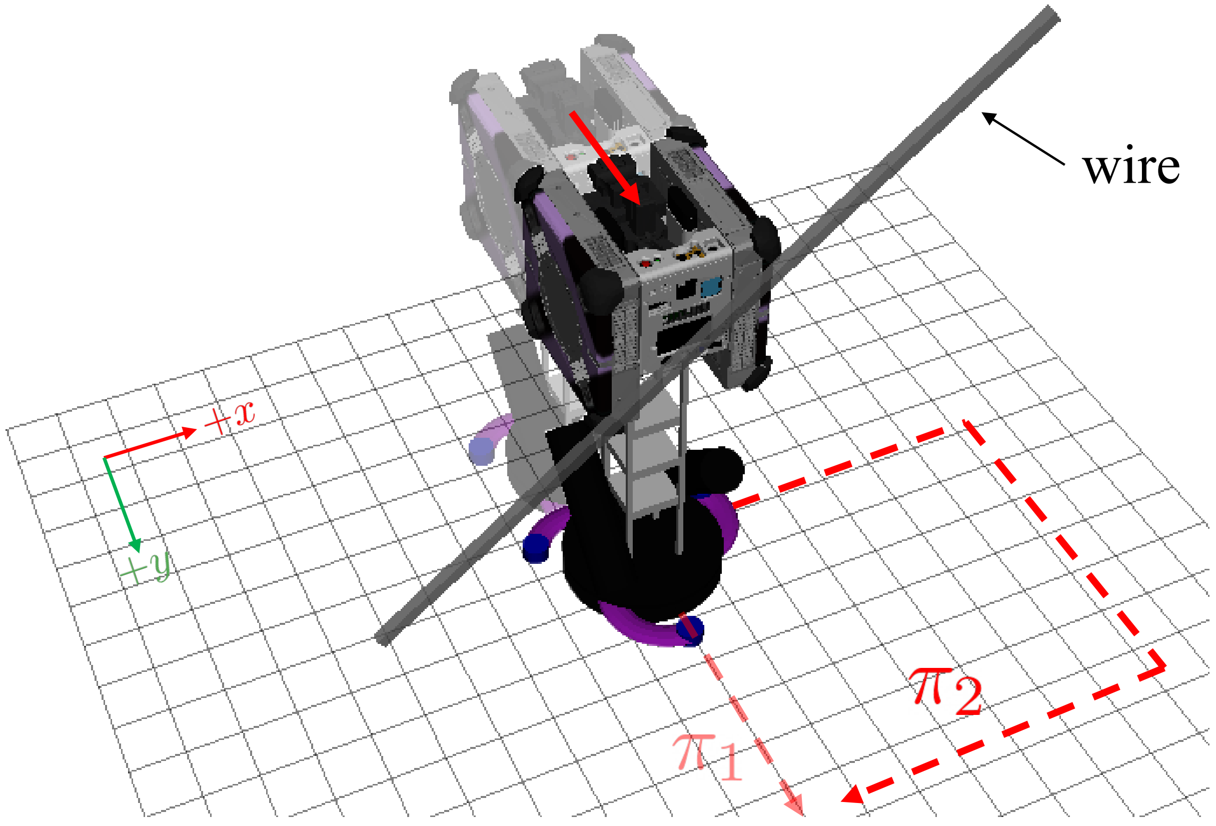

Figure 3.3: A wire obstructs the robot's path to goal π_1. An alternate path, such as π_2, is required to reach the goal.

([$0.25\ N, 0.35\ N$]) of the fan, and the obstacle's size ([$0.23\ m, 0.53\ m$] in both x and y).

S3 (3 Fans + Wire) The robot navigates the environment shown in Fig. 3.5, which contains 3 fans and 1 wire. We randomize the magnitude and angle of each fan the same as in S2. We randomly place the wire in the freespace in the lower half of the environment shown in Fig. 3.5, and randomize its length ([$0.02\ m, 0.20\ m$]) and angle ([$-20°, 20°$]), and use the same rigid-body model as in S1.

Analysis

Across all scenarios, the robot completes the task significantly faster than CMAX using our approach, as shown in Fig. 3.6. Furthermore, the margin by which our approach is more efficient, with respect to both time (Fig. 3.6) and distance traveled (Table 3.1) increases as

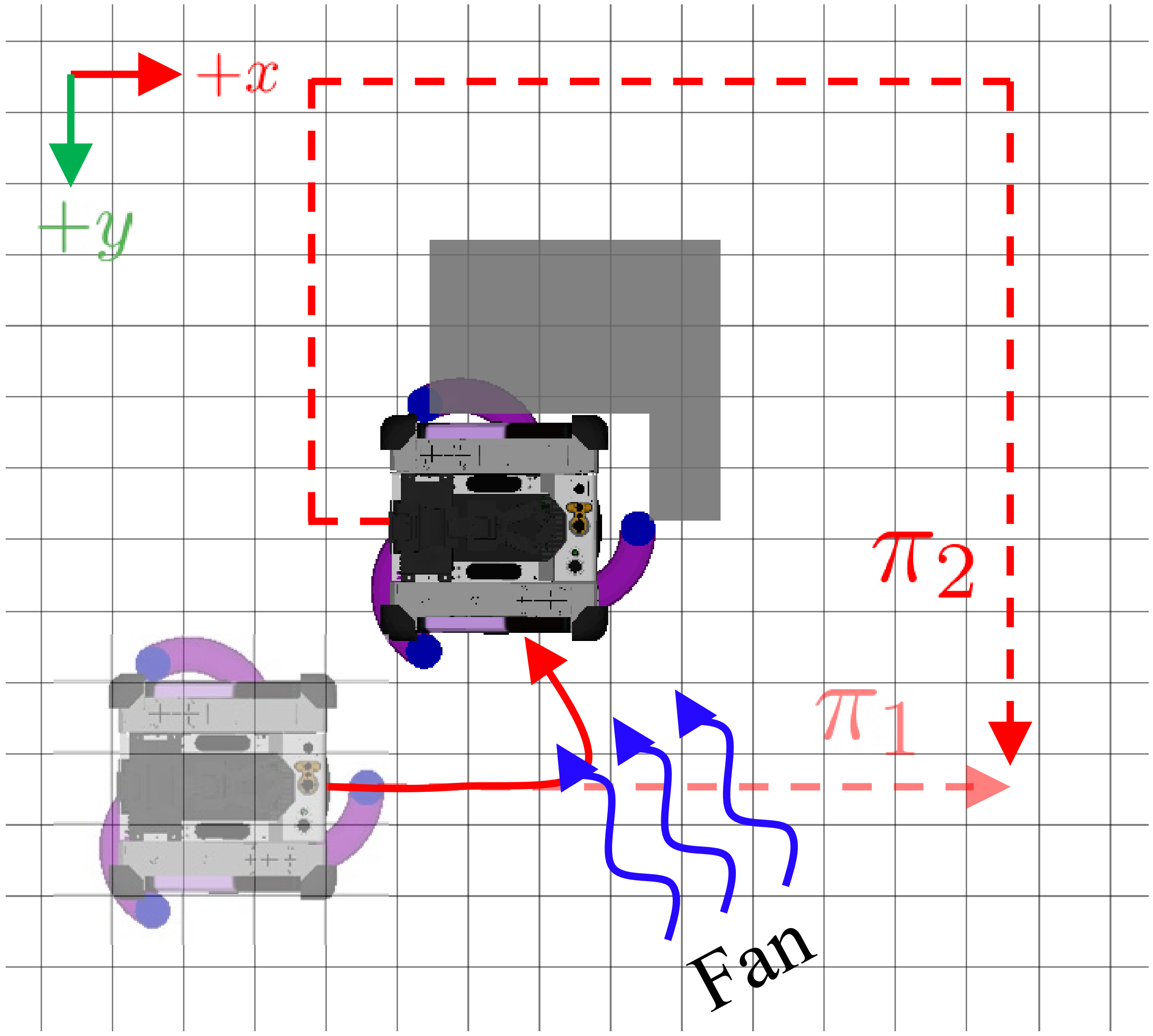

Figure 3.4: A simulated fan exerts a force on the robot, indicated by the blue arrows. Unaware of the fan, the robot initially plans path π_1, but is pushed into the obstacle. An alternate path, such as π_2, is required to complete the task.

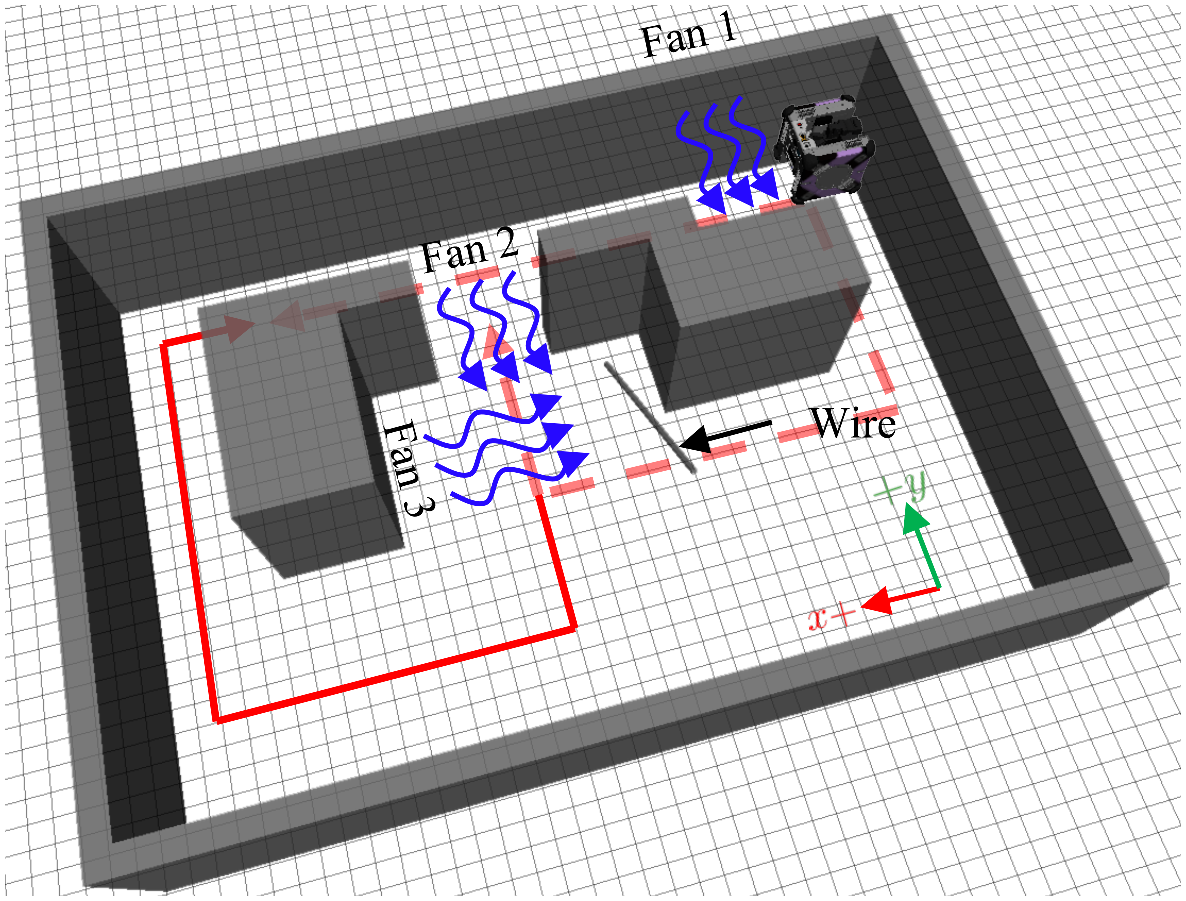

Figure 3.5: 3 fans and 1 wire obstruct the robot's path to goal. We show several alternate paths in red.

	S1	S2	S3
Replan	N/A	N/A	N/A
CMAX	2.46 (0.08)	7.95 (0.35)	13.79 (0.69)
Ours	2.53 (0.07)	**5.41 (0.12)**	**8.62 (1.64)**

Table 3.1: Mean distance traveled (m) by the robot in each scenario, with standard error in parentheses.

	S1	S2	S3
CMAX	**1.96e-6 (1.00e-7)**	**8.65e-5 (1.06e-5)**	**7.10e-4 (2.35e-5)**
Ours	8.35e-2 (4.56e-3)	9.93e-2 (5.12e-3)	9.42e-2 (3.34e-3)

Table 3.2: Mean planning time in seconds, with standard error in parentheses.

the complexity of the scenario increases, from S1 being the simplest to S3 being the most complex.

We found that replanning was unable to complete any task. In all experiments, the planner continues to produce a path that passes through the discrepancy region, and the robot gets stuck. This validates that we chose scenarios where changing the model or planner is necessary to complete the task.

In S1, the control-level information provides little additional insight into the size of the region where the model is inaccurate– the robot is stopped in place by the wire obstructing its path. Consequently, our approach shows the smallest gain in efficiency with respect to time, and no significant difference compared to CMAX with respect to distance. However, as the complexity of the scenarios increase in S2 and S3, this control-level information becomes more valuable– our approach is able to more quickly determine that the model is wrong in a larger region by observing the performance of the path following controller as the robot attempts to move through the fan's air current. That is why we see a larger advantage for our approach over CMAX in these scenarios, with respect to both time and distance.

One drawback of our approach, however, is increased planning time, as shown in Table 3.2. While our approach leads to lower overall time to complete the task (including fewer number of replans, as shown in Table 3.3), each planning call is more expensive. The additional computation time is largely due to the Predict routine, which performs a series of expensive GP-related computations, as described in Sec. 3.3 (in some cases, Predict took up to 90% of the planning time).

Real-robot

To demonstrate our approach on a real-robot, we ran our planner on-board NASA's Astrobee free-flyer; for more details about Astrobee's sensing, localization, propulsion, etc., see [10,

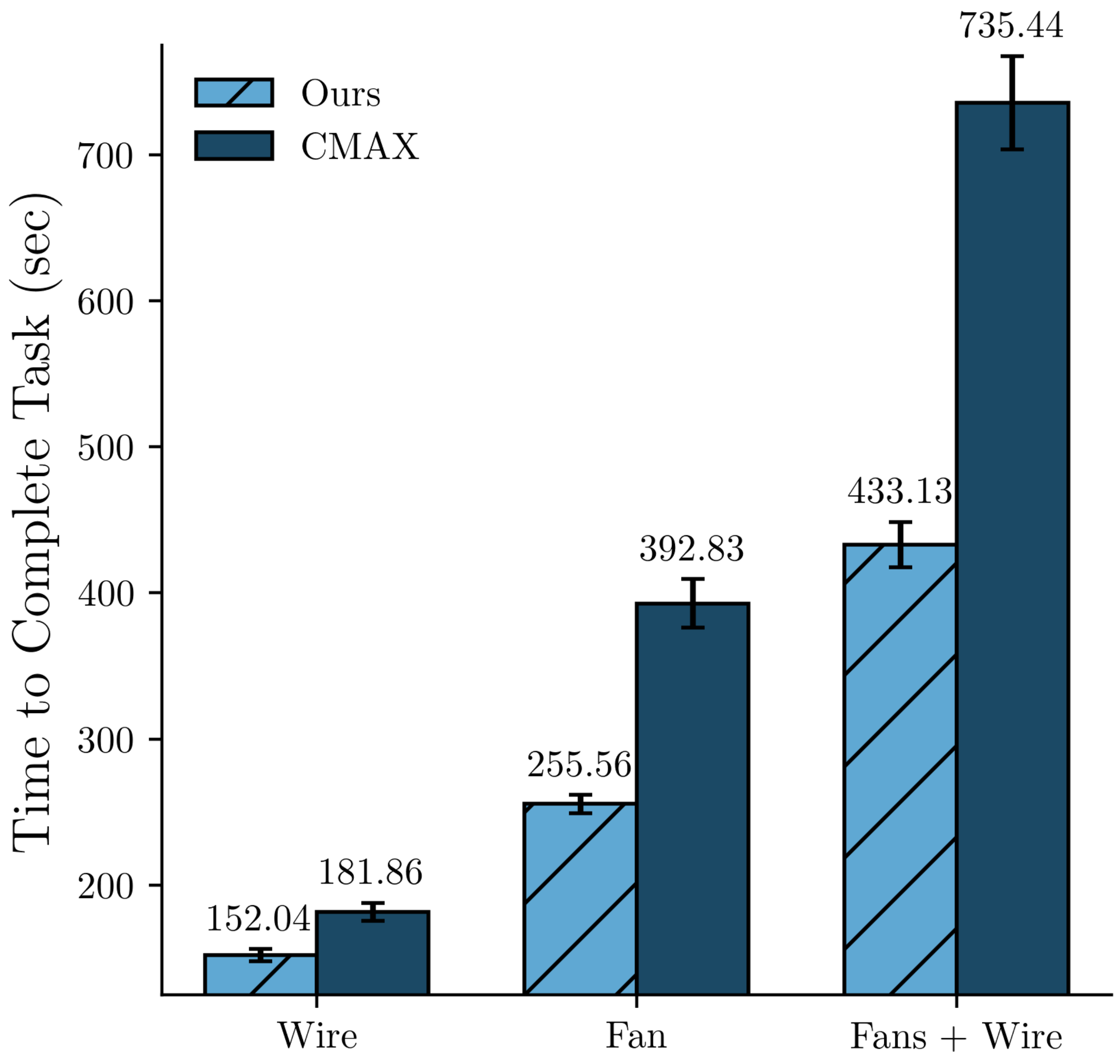

Figure 3.6: The mean and standard errors of time to complete the task, for the wire (S1), fan (S2), and 3 fans + 1 wire (S3) scenarios.

	S1	S2	S3
CMAX	10.2 (0.4)	16.2 (0.3)	29.8 (2.2)
Ours	**4.6 (0.2)**	**5.0 (0.2)**	**8.6 (1.6)**

Table 3.3: Mean number of replans (standard error in parentheses).

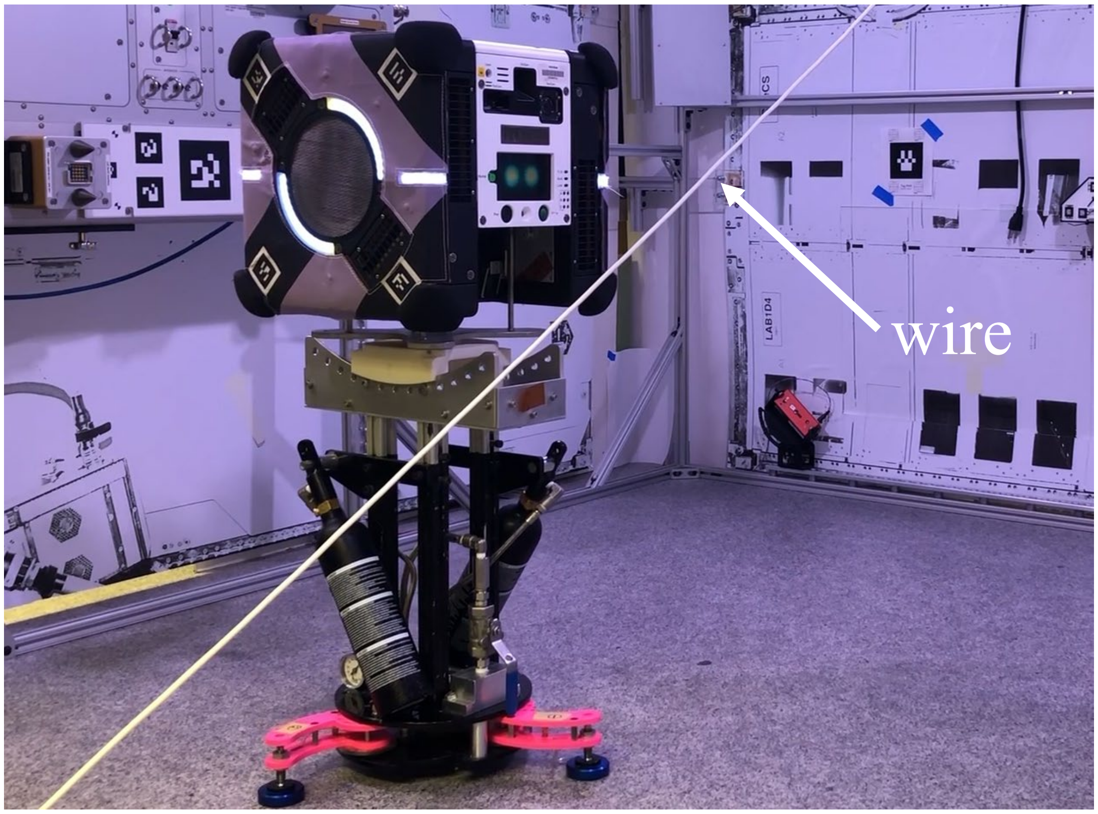

Figure 3.7: Astrobee plans a path to the goal, but the path is obstructed by an unseen wire.

22]. Astrobee is designed for zero-gravity, but we utilized a $2\ m \times 2\ m$ low-friction surface in a lab at NASA. We show our full results, comparing our approach with replanning and CMAX, in the accompanying video.

Fan

Similar to S1, as shown in Fig. 3.1. The planner finds a path around the other side of the obstacle (Fig. 3.1, solid line), to avoid the fan and reach the goal.

Wire

Similar to S2, as shown in Fig. 3.7. Unlike the simulation, the robot can actually get stuck in the wire, and there is also some flexibility and elasticity in the wire, adding additional challenge. The planner is eventually able to find a path to the goal around the wire.

3.5 Conclusion and Future Work

In this chapter, we have presented an approach to enable robots to complete tasks despite planning with an inaccurate model. We utilize control-level information to bias replanning away from where we infer the model to be inaccurate. We found that our approach led to significantly more efficient robot behavior compared to baselines in simulation. Finally, we demonstrated our approach running successfully in lab experiments on NASA's Astrobee free-flyer.

Despite the advantages, our approach introduces additional overhead, leading to increased planning times. We plan to investigate ways to address this through algorithmic improvements and optimizations. Furthermore, we believe that finding additional, more complex robotics applications – such as environments that include other agents or movable objects – is an interesting direction for future work. Finally, we discuss additional extensions, such as to stochastic environments, in Chapter 5.

Chapter 4

Planning Conservatively with Unreliable Human Models

Robots such as autonomous vehicles and assistive manipulators are increasingly operating in dynamic environments and close physical proximity to people. In such scenarios, it is important that robots not only account for the current state of the humans nearby, but also predict their future state to plan safe and efficient trajectories.

To maximally preserve safety, a robust optimal control perspective models the human as taking any action (with equal likelihood) from a set of controls. The predictor combines this control set with a conservative human dynamics model to compute a *full forward reachable set*, or the set of all states that the human could reach from their current state [53, 19]. This approach allows the robot to produce safe predictions when very little is understood about human decision-making.

A complementary perspective is that there is structure to human decision-making: humans have intentions, and make decisions in pursuit of these intentions. For example, consider an indoor home environment where people often move towards chairs, tables, or doorways. Predictors synthesized from this perspective, called *intent-driven predictors*, build data-driven models of human actions given intent [6, 65, 86, 34, 12], and have been successful in a variety of domains including manipulation [3, 18, 37], autonomous driving [71], and navigation [47, 68] (see [69] for a survey). Since human behavior varies between people and over time, these decision-making models are often parameterized and predictors maintain a belief distribution over the model parameters [21, 40]. This provides a direct and succinct way for the robot to use online data to update its human model [86, 8, 35, 6].

However, a key challenge remains with such intent-driven predictors. To update the belief over model parameters and to generate predictions, intent-driven predictors rely on priors and on likelihood models which describe the probability of observing a data point as a function of the model parameters. Although these two components enable data and prior knowledge to improve the human model online, likelihood models are difficult to specify and the priors may be incorrect.

In this work we seek an approach which bridges robust control and intent-driven pre-

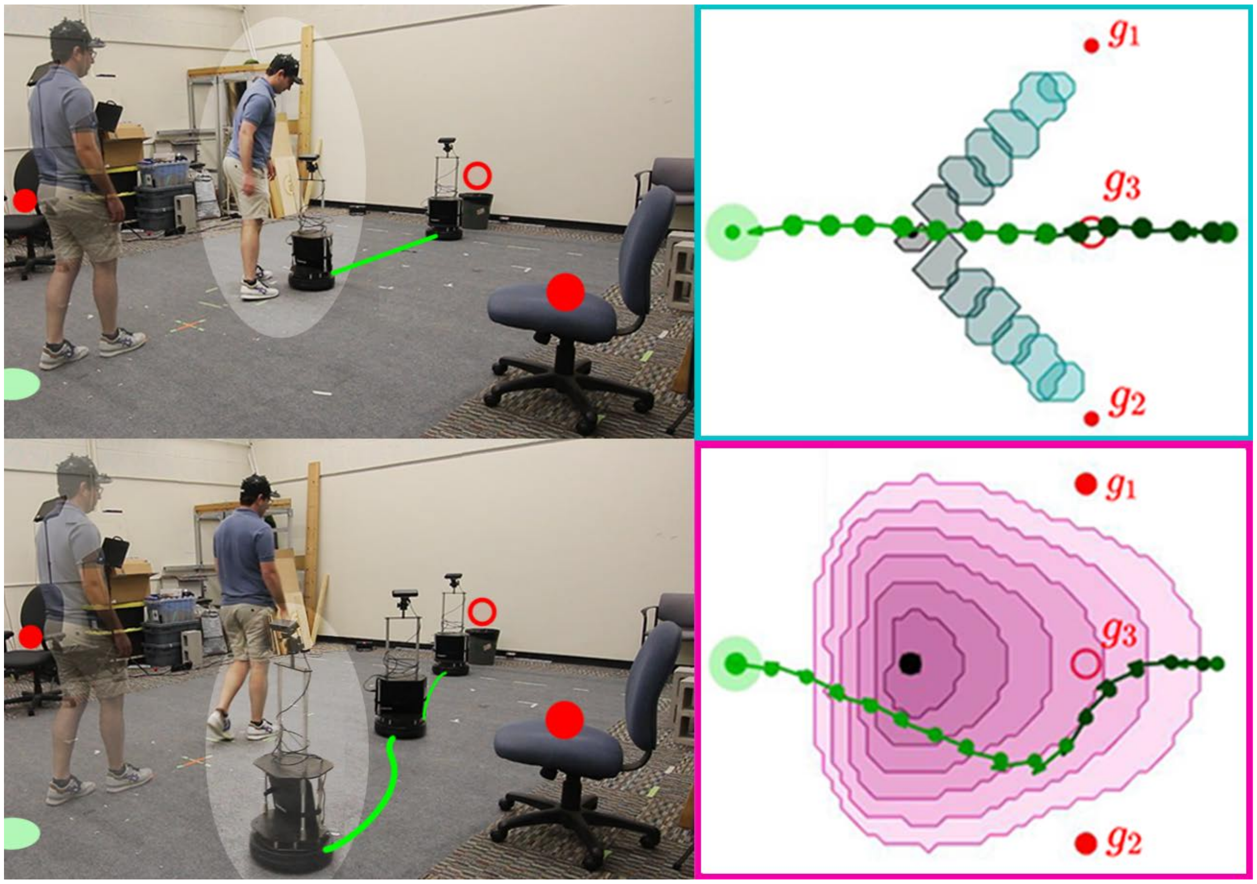

Figure 4.1: When intent-driven human models are misspecified in Bayesian predictors, robots confidently plan unsafe motions (top). Our approach (bottom) trusts the intent-driven model only to remove completely *unlikely* human actions, resulting in safer robot plans despite a misspecified model. (not depicted here) When planning using the worst-case predictor, the robot has to leave the environment entirely to avoid the predicted human state.

dictors: a predictor which is more robust to misspecified models and priors, but still able to leverage human data online to safely reduce conservatism. Our key idea is to compute a *restricted* forward reachable set by *trusting the intent-driven model to tell us only what is completely unlikely*. However, unlike intent-driven predictors, we will not rely on the exact probability of each action under our model during prediction. Rather, we use the decision-making model and the belief to divide the set of human actions in two disjoint sets of likely and unlikely actions. We then predict human motion by treating all sufficiently likely actions as equally probable, much like in the full forward reachable set. Using this restricted control set results in a prediction problem which can be readily formalized and solved through existing robust control methods and tools [52, 15]. We utilize Hamilton-Jacobi (HJ) reachability analysis [53, 46] which is a method for guaranteeing safety for continuous-time, nonlinear dynamical systems. Finally, to properly restrict the set of human controls based on the intent-driven model and belief over model parameters, we augment the state space with the belief. Since the belief encodes the likelihood of human actions given the history of human actions, this explicit belief tracking allows us to compute the likely actions at any future state.

To summarize, our key contributions in this chapter are:

- a robust control framework for human motion prediction which provides robustness against misspecified models and model parameter priors;
- a comparison of our approach to forward reachable set and stochastic predictors for static and time-varying human intent models and in three pedestrian scenarios where the belief over the human intent changes online;
- a demonstration of our prediction approach in hardware.

This chapter is based on research previously published in [7], in collaboration with Andrea Bajcsy, Somil Bansal, Claire Tomlin, and Anca Dragan.

4.1 Problem Setup

We consider a robot operating in a shared workspace with a human. The robot needs to predict the human's motion (we assume that the robot and human states can be accurately sensed) and plan a collision-free path around the human to reach the goal as efficiently and safely as possible. To describe the motion of the robot and the human, we model both as dynamical systems. Let the state of the human and the robot be $x_H \in \mathbb{R}^{n_H}$ and $x_R \in \mathbb{R}^{n_R}$ respectively. The time-evolution of these states can be described by

$$\dot{x_H} = f_H(x_H, u_H)$$

$$\dot{x_R} = f_R(x_R, u_R)$$

where the human and robot's controls are $u_H \in \mathbb{R}^{m_H}$ and $u_R \in \mathbb{R}^{m_R}$ respectively.

The robot's goal is to plan a control trajectory $u_R(t)$ with $t \in [0, \bar{T}]$ such that it does not collide with the human or any (known) static obstacles, and reaches its goal g_R by $\bar{T}$. In this work, we will solve this planning problem in a receding horizon fashion. However, the future states of the human are not known *a priori*, and thus the robot must predict future human motion in order to plan collision-free trajectories.

Throughout this chapter, we will focus on contrasting the intent-driven and full forward reachable set predictor with our novel predictor. However, all prediction schemes ultimately produce a set of sufficiently likely states (forward in time until the prediction horizon, N) that the robot uses for collision checking. We define the set of likely human states at some future time, $t + \tau$ as: $\mathcal{K}^t(\tau), \ \forall \tau \in [0, N]$.

Running Example

We now introduce a running example for illustration purposes throughout the chapter. Consider a mobile robot that needs to navigate to a goal position $g_R \in \mathbb{R}^2$ in a room where a person is walking. Since the human is a pedestrian in this scenario, we use a planar human model with state $x_H = [h_x, h_y]$ and dynamics $\dot{x}_H = [v_H \cos(u_H), v_H \sin(u_H)]$. We model the human as moving at a fixed speed $v_H \approx 0.6 m/s$ and controlling their heading angle $u_H \in [-\pi, \pi]$. Our mobile robot is modeled as a 3D system with state given by its position and heading $x_R = [s_x, s_y, \phi]$, and speed and angular speed as the control $u_R = [v_R, \omega]$, and dynamics $\dot{x}_R = [v_R \cos\phi, v_R \sin\phi, \omega]$. The robot control inputs are constrained between $[0, 0.6] m/s$ and $[-1.1, 1.1] rad/s$ respectively.

4.2 Background: Robust versus Intent Prediction

In this work, we aim to unify ideas from the robust control with the intent-driven prediction so we start with a brief background on both. In each section, we refer interested readers to more comprehensive resources on each approach.

Robust Control Prediction

The most conservative prediction of human motion is the set of *all* states the human could reach in a time horizon. Let t be the current real time and $\tau \in [0, N]$ be a future time used by the predictor. Also, let $\xi(x_H^0, \tau, u_H(\cdot)) := x_H^\tau$ denote the human state starting from the current state $x_H^0 := x_H^t$ at time 0 and applying control u_H for a duration of τ. The *full Forward Reachable Set (FRS)* is defined as:

$$\mathcal{K}^t_{FRS}(\tau) := \{x_H^\tau : \exists u_H(\cdot), x_H^\tau = \xi(x_H^0, \tau, u_H(\cdot))\} \tag{4.1}$$

In other words, if the human is in state x_H^0, then they are predicted to reach any state x_H^τ in τ time if that state is reachable through some control signal $u_H(\cdot)$.

In general there are many techniques for computing these sets [15, 19, 52], but in this work we use Hamilton-Jacobi (HJ) reachability analysis [46, 54]. In HJ reachability, the computation of the FRS is formulated as a dynamic programming problem which ultimately requires solving for the value function $V(\tau, x_H)$ in the following initial value Hamilton Jacobi-Bellman PDE (HJB-PDE):

$$\begin{aligned}\frac{\partial V(\tau, x_H)}{\partial \tau} + \max_{u_H \in \mathcal{U}} \nabla_{x_H} V(\tau, x_H) \cdot f(x_H, u_H) = 0 \\ V(0, x_H) = l(x_H),\end{aligned} \tag{4.2}$$

where $\tau \geq 0$. The function $l(x_H)$ is the implicit surface function representing the initial set of states that the human occupies $\mathcal{L} = \{x_H : l(x_H) \leq 0\}$. Note that this equation is the continuous-time analogue of the discrete-time Bellman equation. The maximization over the human's control, $u_H \in \mathcal{U}$, encodes the effect of the human dynamics and control on the value, which lies in the set of all possible controls. Note that since this optimization considers all controls, the predictions will include all possible states the human could reach, thereby resulting in the safe but oftentimes overly conservative predictions. Once the value function $V(\tau, x_H)$ is computed, the FRS predictions are given by the sub-zero level set $\mathcal{K}^t_{FRS}(\tau) = \{x_H : V(\tau, x_H) \leq 0\}$. For more details on the HJB-PDE, please refer to [54].

Intent-driven Bayesian Prediction

Unlike the robust predictor, the intent-driven Bayesian predictor couples a structured model of how the human chooses their actions with the dynamics model. In general, constructing a human decision-making model for a robotic application is a particularly difficult modeling challenge and many approaches exist in the literature (see [69]). In this work, we consider stochastic control policies that are parameterized by a discrete random variable λ^t where Λ is the set of all values that λ^t can take. The human's control policy can be described by the probability density function $u^t_H \sim p(u^t_H \mid x^t_H; \lambda^t)$. Here, λ^t can represent many different aspects of human decision-making, including what goal locations they are moving towards [86] or even the kind of visual cues they pay attention to in a scene [34]. We refer to these aspects of human decision-making as the human's *intent*. Furthermore, we use the superscript t on the parameter to denote that the value of the human parameter can be time-varying. This allows the human model to encode how the human's intent changes over time; for example, if a person changes the goal they are moving towards in a room.

In general, the specific choice of parameterization is often highly problem specific and can be hand-designed or learned from prior data [86, 20]. Regardless of the specific parameterization, in practice, the true value of λ^t is frequently unknown beforehand and instead can be estimated from the measurements of the true human behavior. Thus, at any time t, the robot additionally maintains a belief distribution $b^t(\lambda^t)$ over the model parameters, which allows it to estimate the human's intent online via a Bayesian update:

$$b^t_+(\lambda^t \mid u^t_H, x^t_H) = \frac{P(u^t_H \mid x^t_H; \lambda^t) b^t(\lambda^t)}{\sum_{\bar{\lambda} \in \Lambda} P(u^t_H \mid x^t_H; \bar{\lambda}) b^t(\bar{\lambda})} \tag{4.3}$$

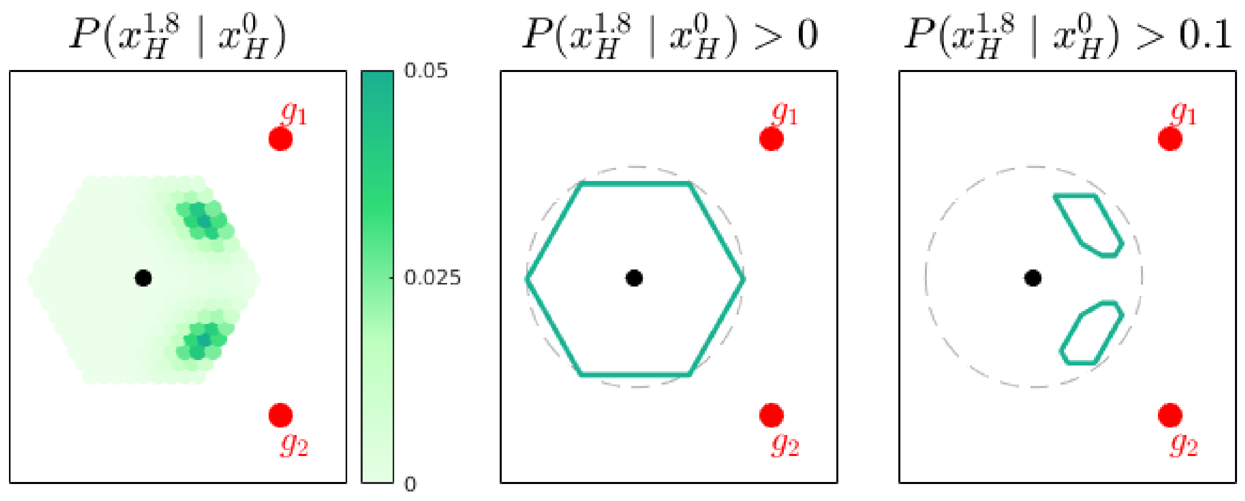

Figure 4.2: Intent-driven predictions 1.8 seconds into future. Human is a black dot, grey dashed circle is the full FRS. (left) Full state distribution shown in green gradient with dark color indicating higher probability, (center) $\epsilon = 0$ is the full FRS, (right) $\epsilon > 0$ drastically reduces set of states to avoid.

Returning to our running example, the robot has uncertainty about the human's goal location. Let the human parameter $\lambda^t \in \Lambda = \{g_1, g_2\}$ take two values which indicates which goal location the human moving towards. The human decision-making model at any state and for a particular goal is given by a Gaussian distribution over the heading angle with mean pointing in the goal direction and a variance representing uncertainty in the human action:

$$p(u_H^t \mid x_H^t; \lambda^t) = \begin{cases} \mathcal{N}(\mu_1, \sigma_1^2), & \text{if } \lambda^t = g_1 \\ \mathcal{N}(\mu_2, \sigma_2^2), & \text{if } \lambda^t = g_2 \end{cases}, \tag{4.4}$$

where $\mu_i = \tan^{-1}\left(\frac{g_i(y) - h_y^t}{g_i(x) - h_x^t}\right)$ and $\sigma_i = \pi/4$ for $i \in \{1, 2\}$. Here, $(g_i(x), g_i(y))$ represents the position of goal g_i.

At prediction time, the stochastic nature of the human decision-making model and the belief over the parameters is naturally converted into state *distributions* (instead of deterministic sets) forward in time. Note that typically, these predictors use a temporally and spatially discretized form of the dynamics by integrating f_H over a fixed time interval δt. Controls are often discretized too and assumed to be held fixed during δt. This results in the predictor maintaining and updating discrete distributions over the human state space. Given the current real time t, we will denote a future time discrete timestep by $k \in \{0, 1, \dots, \frac{H}{\delta t}\}$.

Suppose the current state of the human at the start of the prediction horizon is $x_H^0 := x_H^t$ and the current belief is $b^0(\lambda^0) := b^t(\lambda^t)$. Assume the human is at x_H^k at some future time k. Combining the dynamics and human policy, the human's state distribution at the next timestep $k+1$ is

$$P(x_H^{k+1} \mid x_H^k; \lambda^k) = \sum_{u_H^k} P(x_H^{k+1} \mid x_H^k, u_H^k) P(u_H^k \mid x_H^k; \lambda^k).$$

This equation can be applied recursively to compute $P(x_H^{k+1} \mid x_H^0; \lambda^{0:k})$ starting from $k = 0$. Marginalizing over all sequences of values that the human parameter λ could take, $\mathcal{S}_k$, where $|\mathcal{S}_k| = |\lambda|^k$, we get the overall distribution over the human state at future time step $k+1$: $P(x_H^{k+1} \mid x_H^0)$.

Here, the probability of the parameter sequence has to be set in the model and is generally defined by

$$P(\lambda^{0:k} \mid x_H^0) = \Big(\prod_{m=1}^{k} P(\lambda^m \mid \lambda^{m-1}) \Big) b^0(\lambda^0).$$

Note that in the case of static latent parameters, the summation simplifies to

$$P(x_H^{k+1} \mid x_H^0) = \sum_{\lambda \in \Lambda} P(x_H^{k+1} \mid x_H^0; \lambda) b^0(\lambda).$$

Importantly, at planning time, the robot must decide which predicted states are sufficiently likely to warrant avoiding. A strict notion of safety requires the robot to avoid all states whose probability is > 0. While safe (and equivalent to the full FRS), this choice of states does not leverage the data encoded through the belief or the human decision-making model. To reduce the volume of this set in a way commensurate with human decision-making, choosing a nonzero probability threshold is desirable and reveals a significantly smaller set of states that aligns with the model. Thus, the ultimate predicted set of human states that the robot must avoid at planning time is:

$$\mathcal{K}_\epsilon^t(k) = \left\{ x_H^k : P(x_H^k \mid x_H^0) > \epsilon \right\}, \forall k \in \left\{ 0, \dots, \frac{N}{\delta t} \right\} \tag{4.5}$$

where $\epsilon \geq 0$ is a safety threshold and a design parameter.

4.3 A Robust-Control Framework for Intent-Driven Human Prediction

Our key idea in this approach is to compute a *restricted* forward reachable set by trusting the intent-driven model to infer only what is *completely unlikely*. After using the intent-driven model to prune away sufficiently unlikely actions, our robust predictor will safeguard against all sufficiently likely actions equally, much like in the full forward reachable set. The main question becomes how to perform this control-set pruning in a principled way over the prediction horizon.

One simple way of choosing this set is as follows. At the beginning of the prediction horizon, let the human state be $x_H^0 := x_H^t$ and the current belief be $b^0 := b^t$. We can form a new distribution over the human's controls at the first time step by marginalizing out the latent model parameters, given the initial belief we have over those parameters: $p(u_H \mid x_H) = \sum_{\lambda \in \Lambda} p(u_H \mid x_H; \lambda) b^0(\lambda)$. Then, we can choose the set of human actions to be those for which this marginalized initial likelihood is above a threshold: $\mathcal{U}(x_H) = \{u_H : p(u_H \mid x_H) \geq \delta\}$. This leads to a set of reachable states for $t = 1$. To obtain the set of states at $t = 2$, it is tempting to follow the same process, restricting the set of future actions based on b^0. Unfortunately, this would (accidentally) model that the human is "resampling" their intent from this initial distribution independently at every step. It would disregard that a human's second action will be *consistent* with their first, with the intent only changing according to the dynamics of λ. Thus, we must enforce that the human control from a state x_H at $t = 2$ is not only consistent with the initial belief, but also with the control that took them to state x_H.

To properly restrict the set of feasible controls over the prediction horizon, we need to take into account how the likelihood of any future control depends on the past sequence of human controls. The belief precisely encodes this likelihood given the past sequence of human controls through the Bayesian update from Eq. 4.3. Thus, our predictor explicitly tracks the updated belief as it makes predictions, rather than just the updated state, and restricts future actions based on future beliefs (see left of Fig. 4.4 for intuitive depiction). Let this joint state space be denoted by

$$z^t := \begin{bmatrix} x_H^t & b^t(\lambda^t = \lambda_1) & \dots & b^t(\lambda^t = \lambda_{|\Lambda|}) \end{bmatrix}. \tag{4.6}$$

When predicting using this state space, to simultaneously predict the possible future beliefs over λ^t and corresponding likely human states, we consider the joint dynamics:

$$\dot{z}^t = \begin{bmatrix} \dot{x}_H^t & \dot{b}^t(\lambda^t = \lambda_1) & \dots & \dot{b}^t(\lambda^t = \lambda_{|\Lambda|}) \end{bmatrix}, \tag{4.7}$$

where $\dot{z}^t := f(z^t, u_H^t)$. The continuous evolution of the belief $b^t(\lambda^t)$ can be described by:

$$\dot{b}^t(\lambda^t) = \gamma \left(\left(\lambda^t \mid u_H^t, x_H^t \right) - b^t(\lambda^t) \right) + k \left(b^t(\lambda^t) \right) \tag{4.8}$$

for any specific value of λ^t. Here, the function $k(\cdot)$ represents the intrinsic changes in the human intent, whereas the other component captures the Bayesian change in $b^t(\lambda^t)$ due to the observation u_H^t. Note that the time derivative in (4.8) is pointwise in the space of all λ's. Typically, the Bayesian update is performed in discrete time when the new observations are received. However, to unify this with continuous-time robust controls tools, in this work, we reason about continuous changes in $b^t(\lambda^t)$. Intuitively, to relate the continuous-time Bayesian update to the discrete-time version, γ in (4.8) can be thought of as the observation frequency. Indeed, as $\gamma \uparrow \infty$, i.e., observations are received continuously, $b^t(\lambda^t)$ instantaneously changes to $b_+^t(\lambda^t \mid u_H^t, x_H^t)$. On the other hand, as $\gamma \downarrow 0$, i.e., no observation are received, the Bayesian update does not play a role in the dynamics of $b^t(\lambda^t)$.

Now that we are able to track the evolution of the robot's belief and the human's physical states, we can prune unlikely human actions by combining the intent-driven model and the *predicted* belief over the human model parameters. For some future time $\tau \in [0, N]$, the marginalized human action distribution at joint state z^τ is given by

$$p(u_H^\tau \mid z^\tau) = \sum_{\lambda \in \Lambda} p(u_H^\tau \mid x_H^\tau; \lambda) b^\tau(\lambda). \tag{4.9}$$

Very importantly, note that this set is joint state dependent, and therefore *belief*-dependent. This allows us to prune away the sufficiently unlikely actions by removing actions which are not assigned sufficient probability under the future predicted belief (and not just the initial belief):

$$u_H^\tau \in \mathcal{U}(z^\tau), \quad \mathcal{U}(z^\tau) = \{u_H^\tau : p(u_H^\tau \mid z^\tau) \geq \delta\} \tag{4.10}$$

where $p(u_H \mid z)$ is computed as in Eq. (4.9) and δ is a threshold that partitions the actions into likely and unlikely.

Returning to our running example, consider the case when the intrinsic behavior of the human does not change over time, i.e., $k(b^t(\lambda^t)) = 0$, meaning the human has a fixed goal they are moving to. Since λ takes only two possible values, the joint state space is three dimensional. In particular, $z = \begin{bmatrix} h_x & h_y & p_1 \end{bmatrix}$, where $p_1 := b^t(\lambda^t = g_1)$ and $b^t(\lambda^t = g_2)$ is given by $(1 - b^t(\lambda^t = g_1))$ so we do not need to explicitly maintain it as a state. The state-dependent control distribution is $p(u_H \mid z) = p_1 \mathcal{N}(\mu_1, \sigma_1^2) + (1 - p_1)\mathcal{N}(\mu_2, \sigma_2^2)$ and can be used to compute the set of allowable controls $\mathcal{U}$ for different values of δ via Eq. (4.10). Note Fig. 4.3 where the top-left inset figures show the allowable controls for $x = (0, 0)$ and two different belief states $b^0(\lambda = g_1) = 0.5$ and $b^0(\lambda = g_1) = 0.9$ for three different δ thresholds.

Using HJ-Reachability for Prediction

Using a control set rather than a distribution results in a prediction problem which can be readily solved using the HJB-PDE formulation in Section 4.2. At any real time t, given the current state of the human and the current belief over the model parameters, we can construct the joint state at the beginning of the prediction horizon $z^0 := z^t$. Using this initial state and the thresholded control policy from (4.9), we are interested in computing the following set:

$$\mathcal{V}(\tau) := \{z^\tau : \exists u_H(\cdot) \in \mathcal{U}(z), z^\tau = \xi(z^0, \tau, u_H(\cdot))\}, \tag{4.11}$$

where $\tau \in [0, N]$. Intuitively, $\mathcal{V}(\tau)$ represents all possible states of the joint system, i.e., all possible human states and beliefs over λ, that are reachable under the dynamics in (4.7) for some sequence of human actions. We refer to this set as the *Belief Augmented Forward Reachable Set (BA-FRS)* from here on. Much like the computation of the full forward reachable set from Sec. 4.2, we can leverage the same tools from HJ-Reachability to compute the BA-FRS where x_H is replaced with z and instead of optimizing over all controls $\mathcal{U}$, we use the restricted set of controls $\mathcal{U}(z)$ instead.

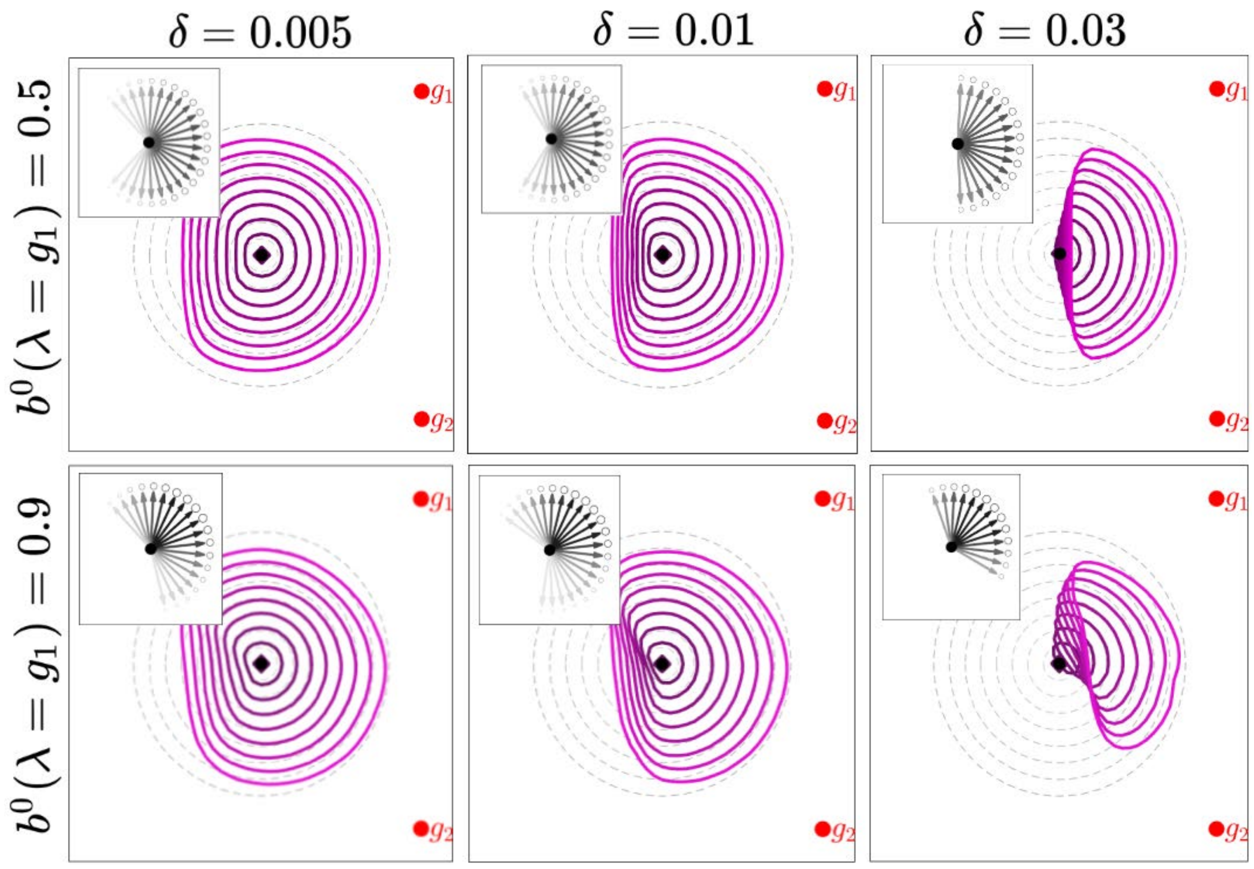

Figure 4.3: Effect of the belief and the δ-threshold on the admissible set of controls (shown in upper-left inset) and the overall predictions (shown in pink) for 3 seconds into the future.

After solving the dynamic programming problem to obtain the BA-FRS from (4.11), our predictions include not only the physical locations of the human but also the corresponding future beliefs. However, for motion planning, the robot needs to collision-check against a set of physical states the human could occupy. We obtain this set by projecting $\mathcal{V}(\tau)$ on the human state space via

$$\mathcal{K}_\delta^t(\tau) = \bigcup_{z^\tau \in \mathcal{V}(\tau)} \Pi(z^\tau), \quad \forall \tau \in [0, N]$$

where $\Pi(z)$ is the physical state component of z.

Returning again to our running example, our starting set of states, $\mathcal{L}$, is a small ball at the joint starting state $z^0 = [0, 0, 0.5]$, shown in grey in Fig. 4.4. Consider how the state and belief can change in a small ($\delta t = 0.4668$) timestep after observing the person moving towards goal 1 via $u_H = \pi/4$. Since this action is highly likely under the model where $\lambda = g_1$, then the next joint state will have the person not only moved physically in that direction, but

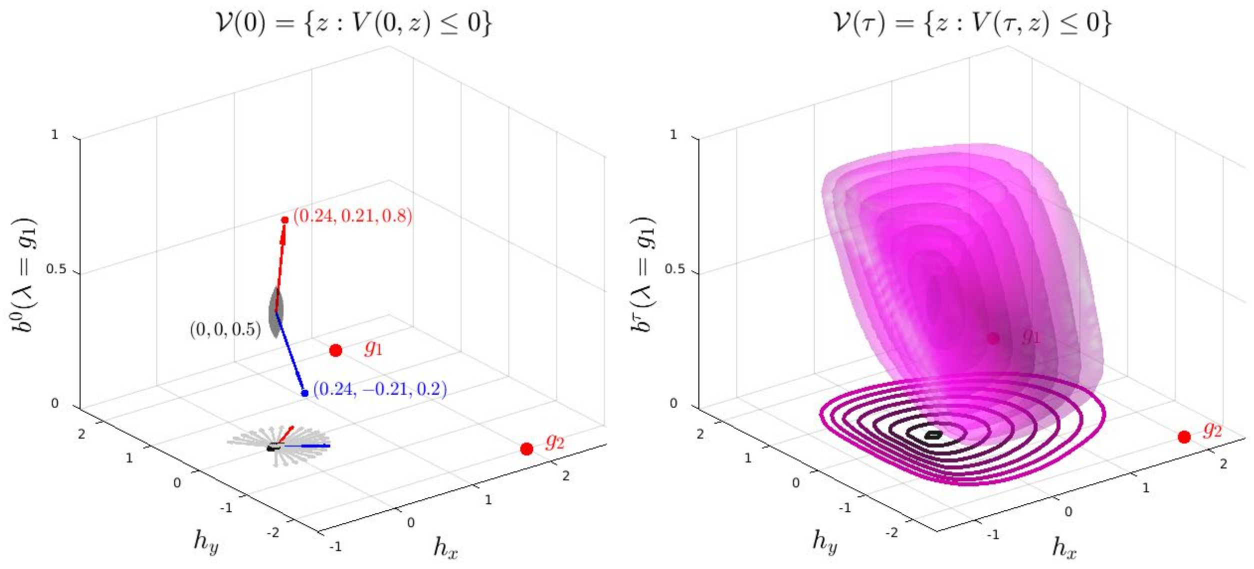

Figure 4.4: (left) Initial set in z-space (in grey). Likely control distribution for $\delta = 0.01$ shown projected in $h_x - h_y$ plane. Comparisons of the resulting joint state if the human moves directly towards g_1 (in red) versus towards g_2 (in blue). (right) 4 seconds BA-FRS and its projection into x_H-space.

the posterior will have increased probability mass on $b(\lambda = g_1)$. Similarly, this probability decreases if the human moves to g_2. After solving for $V(\tau, z)$ via (4.2), we take the sub-zero level set to retrieve the joint state predictions (Fig. 4.4, right), and the predictions $\mathcal{K}^t_\delta$ after projecting onto the human's state space.

In summary, to predict the human's motion, our predictor optimizes the initial value HJB-PDE from (4.2) but instead of optimizing over *all* controls, our formulation modifies Eq. (4.2) to maximize over the restricted set $u_H \in \mathcal{U}(z)$ which changes based on human state, time, and belief. Ultimately, the proposed prediction framework is a *less conservative* FRS, but a *more conservative* intent-driven predictor. This has two advantages: (1) when the intent-driven model is correct, it computes an under-approximation of the full FRS to reduce conservatism in a principled way, and (2) when the model is incorrect, we can be more robust to such inaccuracies since the predictions no longer rely on the exact action probabilities. We discuss this further in Sec. 4.4.

4.4 Prediction Comparisons

We now compare our predictor with the intent-driven Bayesian predictor and the full FRS when (1) the human intent is static, (2) the human intent is time-varying, and (3) the human moves in unmodelled ways over time.

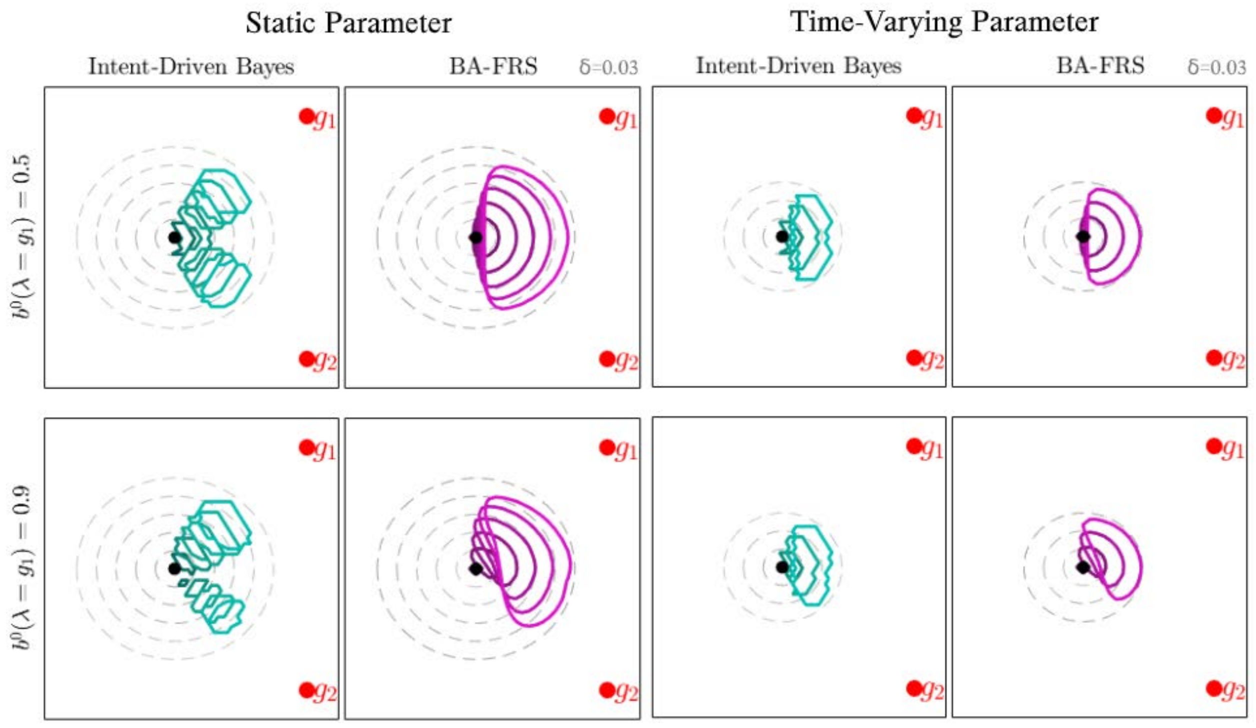

Figure 4.5: Comparisons of Bayesian and BA-FRS predictions for static vs. time-varying human intent. Dashed lines are the full FRS. Predictions are for 2 seconds for the static parameter and 1.8 seconds for time-varying. For the Bayesian predictor, we choose ϵ to capture the $(1-\delta)$ most likely states.

Static parameter

One simple but common predictive model of human behavior assumes that the human's intent (and thus model parameter λ) is static. In our running example, this means the person has a fixed goal location they are moving towards and they will not change their mind. Mathematically, in the intent-driven Bayesian predictor, this is represented via the λ transition distribution $P(\lambda^{k+1} \mid \lambda^k) = \mathbb{1}_{\{\lambda^{k+1}=\lambda^k\}}$ where $\mathbb{1}$ represents the indicator function. In the BA-FRS predictor it means $k(b^t(\lambda^t)) = 0$ in the distribution dynamics.

In the left block images in Fig. 4.5, we see a snapshot of predictions generated by the intent-driven predictor and the BA-FRS forward in time for 2 seconds ($N = 18$). The full forward reachable set is visualized as a series of concentric dashed grey circles. The top row represents a uniform belief over the two goals, while the bottom row represents a high belief on goal 1 (g_1). As expected, both the intent-driven predictor and the BA-FRS are far less conservative than the full FRS. Furthermore, the set of sufficiently likely states predicted by the intent-driven Bayesian predictor is always contained within the BA-FRS. Consequently, when the belief over λ is confident that the human is moving towards g_1 (see bottom row of Fig. 4.5), then the BA-FRS allows us to compute an approximation of the stochastic predictor.

Time-varying parameter

A more complex model of human intent allows it to vary over time. To encode this time-varying intent in the predictor, we need a model of how the human chooses the next value of λ^t. In our running example, let a simple model for how the person changes their behavior to be:

$$P(\lambda^{k+1} \mid \lambda^k) = \begin{cases} \alpha + (1-\alpha) \cdot \Delta(\lambda^k) & \text{if } \lambda^{k+1} = \lambda^k \\ (1-\alpha) \cdot \Delta(\lambda^{k+1}) & \text{if } \lambda^{k+1} \neq \lambda^k \end{cases} \tag{4.12}$$

where α is a known model parameterwhich governs how likely the person is to change their intent and Δ is a known discrete distribution over the model parameters. This model encodes that if the person was moving towards $\lambda^k = g_1$ at the previous timestep k, they are likely to continue to walk to g_1 at the next timestep with probability $\alpha + (1-\alpha) \cdot \Delta(\lambda^k = g_1)$, or they can switch to $\lambda^{k+1} = g_2$ at the next timestep with probability $(1-\alpha) \cdot \Delta(\lambda^{k+1} = g_2)$. In the BA-FRS predictor, this time-varying intent model is encoded via the distribution dynamics: $k(b^t(\lambda^t)) = \alpha b^t(\lambda^t) + (1-\alpha) \cdot \Delta(\lambda^t) - b^t(\lambda^t)$.

In the right block of images in Fig. 4.5, we see a snapshot of predictions when the latent parameter is time-varying forward in time for 1.8 seconds ($H = 11$). Note that when the parameter is time-varying, the computational complexity of the intent-driven Bayes predictor exponentially increases in the size of the prediction horizon, $|\Lambda|^N$, due to the necessity of tracking all sequences of values that λ can take over time. In practice, prediction was computationally prohibitive for horizons greater than 1.8 seconds. In contrast, the BA-FRS computation grows linearly in the length of the prediction horizon, but exponentially in the number of parameter values due to the addition of the belief in the state. Thus, for time-varying parameters which take a few values and for longer prediction horizons, our prediction method can be particularly suitable for getting an approximation of Bayes predictor at a lower computational complexity.

When λ is static, then the intent-driven predictor with a high belief over g_1 deems moving directly towards g_2 to be highly unlikely. However, when λ is time-varying, the human can "switch" which goal they are moving towards, thereby making states in the direction of g_2 somewhat likely as well. For the BA-FRS, even though directly moving towards g_2 is unlikely under the intent-driven model and belief, the BA-FRS realizes that moving *away* from g_1 is *still likely enough*. Consequently, the predicted BA-FRS mass moves in the direction of g_2 over time, in the case of both static and time-varying λ, allowing us to be robust to suboptimal human trajectories as discussed in the next section.

Online updates & robustness to misspecified models

Ultimately, both the intent-driven Bayesian predictor and the BA-FRS will update the belief over the human parameters online based on how the person moves. Here we simulate three

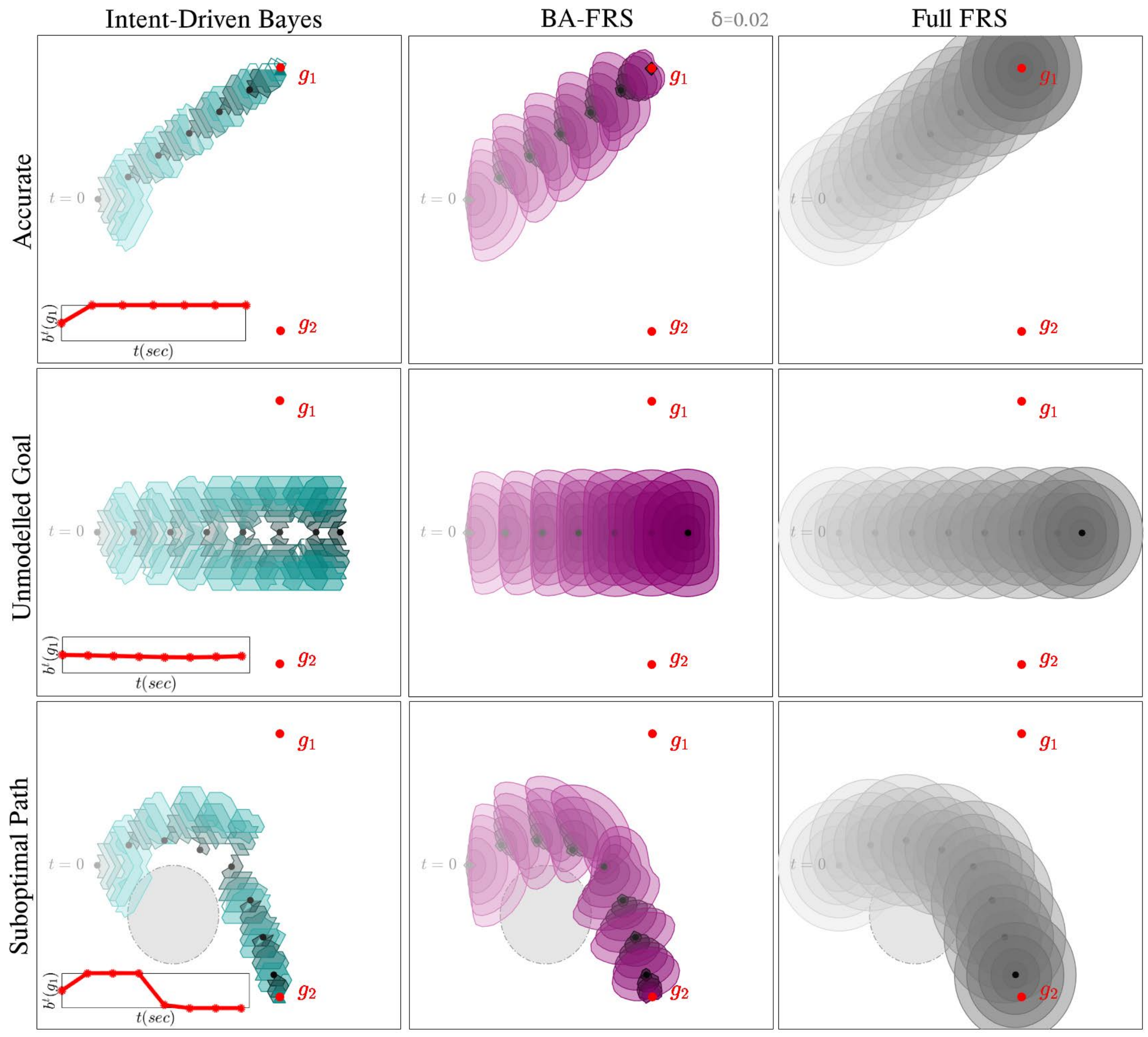

Figure 4.6: Comparison of the intent-driven Bayesian, our BA-FRS, and the full FRS predictions for three scenarios. In the first row the human moves towards one of the modelled goals. In the middle the human moves towards an *unmodelled* goal. In the last row the human is moving towards a modelled goal (g_2) but they take a suboptimal path under our model because they are avoiding an *unmodelled* obstacle on the ground (shown in grey circle). The belief over g_1 is visualized over time in the lower-left inset plot.

scenarios–one where the person takes a path well-modelled by the intent-driven model and two where the person behaves in an unmodelled way–and discuss how our framework ensures robustness in situations like these.

In all examples, the predictors begin with a uniform prior over the two goals, use a static model of human intent, and the BA-FRS uses a $\delta = 0.02$. In the top row of Fig. 4.6 the human has a fixed intent to move towards the upper goal 1 (g_1). In this scenario, the intent-driven model is correctly specified and as the person moves towards g_1, the belief over g_1 increases and the Bayesian predictions focus towards this goal. Our BA-FRS performs similarly since it too performs the belief update over time. However, since the BA-FRS explicitly tracks the evolution of the belief in the future during prediction, the sets include more states even in the direction of g_2. This is because the predictions are safeguarding against slightly suboptimal actions which are still likely enough under the model and would lead to the belief over g_1 *decreasing* in the future. Nevertheless, the BA-FRS takes up significantly less volume than the full FRS, thereby reducing overall conservativism.

The second and third rows demonstrate two human behaviors that are unmodelled – a scenario where the human is actually walking towards a third unmodelled goal in between g_1 and g_2 and a scenario where the human takes a seemingly suboptimal path to g_2 due to an unmodelled obstacle. In the later scenario, the belief over g_1 sharply increases as the person moves around the unknown obstacle. This results in the Bayesian predictor being overly optimistic, and it places most probability mass on states that are in the direction of g_1. In contrast, our predictor remains cautiously conservative because (1) it is safeguarding against the slightly suboptimal but still sufficiently likely actions and (2) it is evolving the belief during the prediction horizon. In fact, the true sequence of human states and actions lies within the predictions of the BA-FRS, ensuring that a robot which relies on these predictions will in fact avoid the states that the human eventually occupies. We discuss the middle scenario in greater detail in Sec. 4.5.

We conducted a series of additional experiments with simulated human trajectories to systematically analyze the three misspecification types: (1) accurate goal but inaccurate optimality level, (2) unmodelled goal, (3) accurate goal but unmodelled obstacle. We compare different predictors for prediction accuracy and conservatism. A predictor is considered accurate at a particular time step if the true future human states lies within the predictions for the entire prediction horizon. Conservatism is measured by computing the percent volume of the full FRS that the predicted states occupy. Both the accuracy and conservatism metrics are computed at each time step and averaged over the horizon $[0, \bar{T}]$. Note that the full FRS always achieves 100% accuracy but also 100% conservatism. These metrics provide us a proxy for the prediction's effect on the safety and efficiency of the robot's plan; ideally, a predictor should have high accuracy and low conservatism over the entire human trajectory. In all experiments, the Bayesian and BA-FRS predictors modelled two goals and used a fixed $\sigma = \pi/4$ in the action model described in the running example from 4.2 and $\delta = 0.02$.

For (1), the human was simulated as moving towards modelled goal g_1 by sampling an action $u(x) \sim \mathcal{N}(\mu_1, \sigma^2)$. To capture a range of human behavior from completely optimal to completely random, we simulated five levels of σ (depicted in Fig. 4.7). We sampled 7

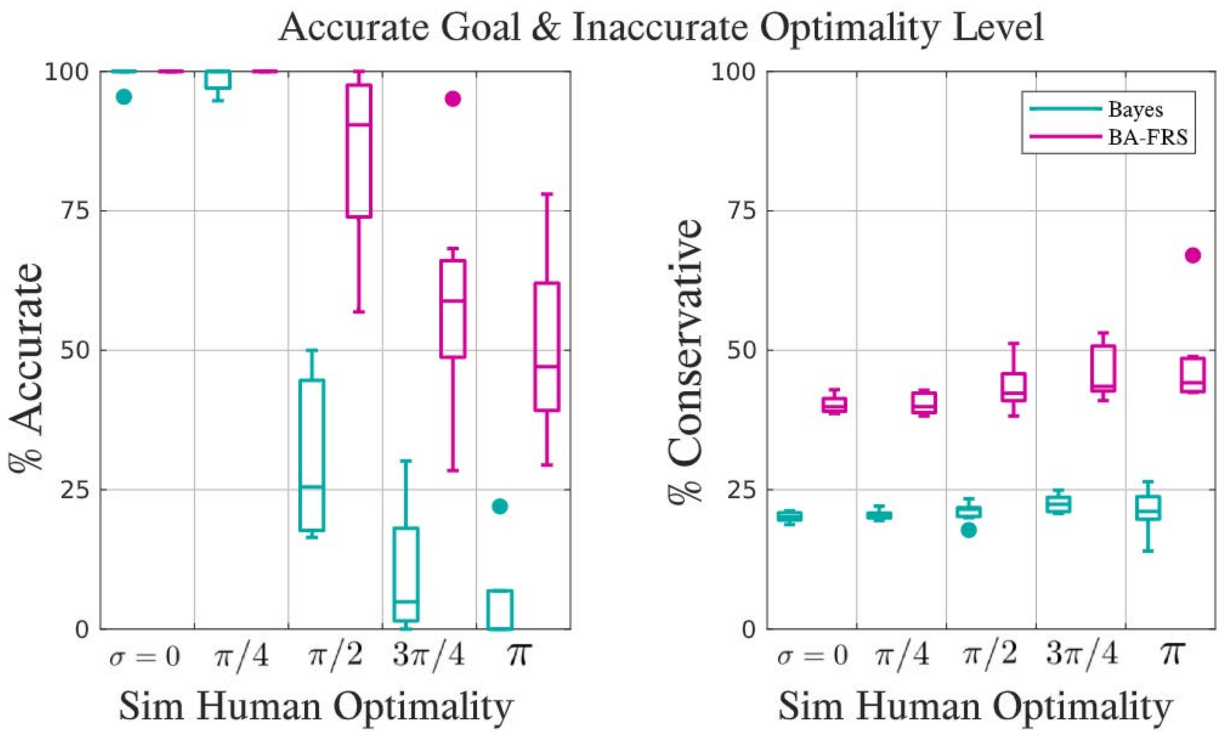

Figure 4.7: Simulated human moves to modelled g_1 but with varied optimality from $\sigma = 0$ (optimal) to π (random). Both predictors use a fixed $\sigma = \pi/4$.

random initial human states for each σ and averaged results over these trials. Fig. 4.7 shows box plots of our metrics for Bayesian and BA-FRS. Although the BA-FRS is about twice as conservative as Bayesian, it maintains a high prediction accuracy across all optimality levels, while still being far less conservative than the full FRS (BA-FRS is $\approx 45\%$ of the full FRS).

For (2) and (3) we fixed the human's initial condition but varied the unmodelled goal or unmodelled obstacle. For unmodelled goals, we randomly sampled 7 unmodelled goals which were diversely spread in the (x,y)-plane. The true (unknown) human trajectory is a straight line to the unmodelled goal starting from the initial position. Fig. 4.8 (top) shows plots of the accuracy and conservatism for each of the unmodelled goals, sorted from the "most" modelled (e.g. an unmodelled goal which is nearby a modelled goal) to "least" modelled. For unmodelled obstacles, the simulated human always moved towards g_2, but their straight-line path was always obstructed by an *unmodelled* obstacle, forcing them to take a trajectory around the obstacle. We simulated 7 of these trajectories around various circular and rectangular-shaped obstacles. Fig. 4.8 (bottom) shows the results sorted from least deviation from straight-line trajectory to the goal to most deviation. The more irrational the human "appears" (either due to an unmodelled goal or taking a suboptimal path to the goal), the more the drop in accuracy of the Bayesian predictor, as it overrelies on the intent

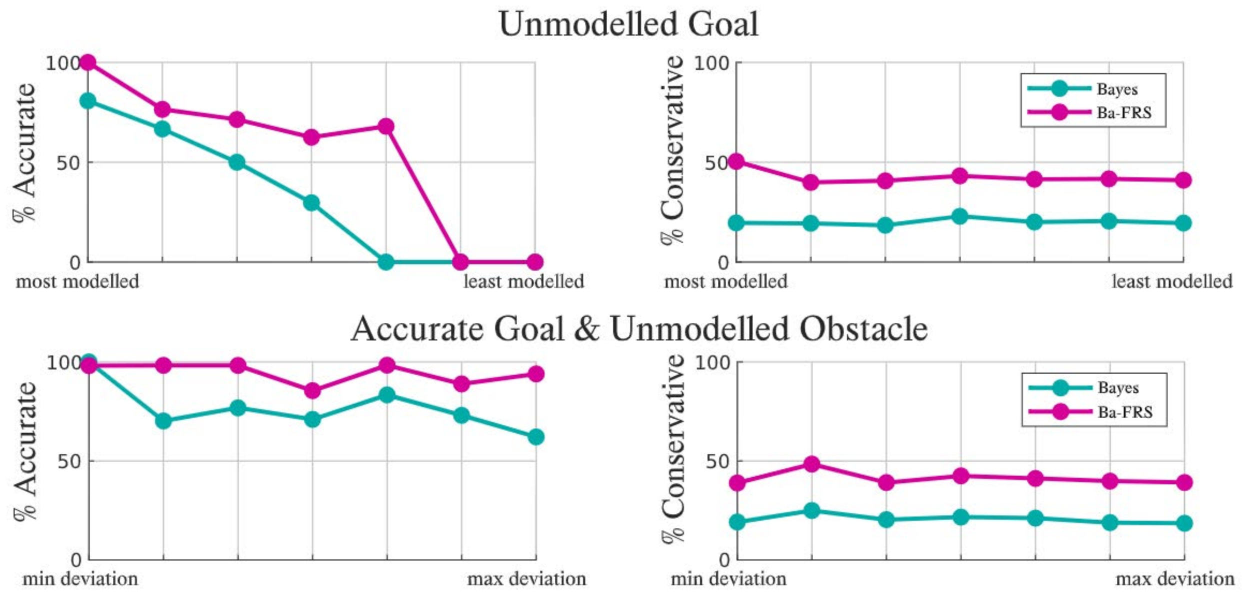

Figure 4.8: (top) Simulated human moves to one of 7 *unmodelled goals* in an optimal trajectory. Results are in increasing misspecification of the goal. (bottom) Simulated human moves to modelled goal g_2 but has their straight-line path obstructed by an *unmodelled obstacle*. Results from 7 unmodelled obstacles are shown in increasing deviation from the straight trajectory.

model to explain the human's behavior. In contrast, since BA-FRS only uses the human model to filter likely and unlikely actions, it maintains a relatively higher accuracy.

4.5 Implications for Safe Motion Planning

Consider the scenario where the actual human goal is midway between the modelled goals g_1 and g_2 (see g_3 label in Fig. 4.1 and middle row of Fig. 4.6), but the true human goal is not explicitly modelled in the intent-driven model. We will use this example in simulation and in hardware to demonstrate the challenges with over-relying on a misspecified intent-driven model. Our hardware experiments are performed on a TurtleBot 2 navigating around a human pedestrian. We measured human positions at 200Hz using a VICON motion capture system and used on-board odometry sensors for the robot state measurement. The robot is modelled via the dynamics in Sec. 4.1, its goal g_R is behind the initial state of the human (green circle in Fig. 4.1) and it uses a spline-based planner [82] to plan six-second trajectories in a receding-horizon fashion.

When the robot uses the full FRS for human motion prediction (see Fig. 4.6 for visualizations of the predictions over time), the robot plans a trajectory which deviates significantly from the ideal straight line path towards its goal and in fact forces the robot to leave the testbed[1]. In contrast, the Bayesian predictor consistently predicts that that pedestrian will walk towards one of the goals and fails to assign sufficient probability to the true human states because of its over reliance on the model. Ultimately, this leads to a collision between the human and the robot (top row Fig. 4.1). Our proposed approach does not rely heavily on the exact action probabilities, and infers that the straight line trajectory is likely enough under the pedestrian model. As a result, the robot makes a course correction early on to reach its goal without colliding with the pedestrian (bottom row Fig. 4.1).

4.6 Conclusion

When robots operate around humans, they often employ intent-driven models to reason about human behavior. Even though powerful, such predictors can make poor predictions when the intent-driven model is misspecified. This in turn will likely cause unsafe robot behavior. In this work, we formulated human motion prediction as a robust control problem over the set of only sufficiently likely actions, offering a bridge between conservative full forward reachable set predictors and intent-driven predictors. We demonstrated that the proposed framework provides more robust predictions when the prior is incorrect or human behavior model is misspecified, and can perform these predictions in continuous time and state using the tools developed for reachability analysis. In the future, we will scale it to higher dimensions with multiple humans, perform a user study to gauge the impact of our predictor on a humans' comfort in close navigation scenarios, and integrate it with online model confidence estimation approaches.

[1]Hardware demonstration videos: https://youtu.be/uZi-zIi1S6A

Part III

Future Work and Conclusion

Chapter 5

Future Research Directions

In this chapter, we present directions for future research, building from the approaches for model-based planning and execution that we have presented in the previous chapters. Specifically, we focus on two broad categories: first, how to unify the three approaches presented in this dissertation into a single framework; and second, how to extend that framework to handle a broader set of unreliable models.

5.1 Towards a Unified Framework for Planning and Execution with Unreliable Models

Thus far, the work in this dissertation has focused on improving two aspects of model-based planning and execution — namely, how to *choose a model*, and subsequently how to *plan with that model* — to be more robust to unreliable models. We have presented three specific approaches to do so, and investigated how they enable various types of robots to successfully complete their tasks despite using an unreliable model in planning and execution. So far, we have investigated these approaches largely in isolation, despite the fact that they address a complementary set of challenges around using unreliable models. Therefore, a natural next question is: can we unify these three approaches into a single, more robust model-based planning and execution framework? We explore possible directions to answer that question in this section.

From a high-level design perspective, we can combine the three approaches presented so far into a single framework, as illustrated in Fig. 5.1. Instead of having a single model of the world that the robot uses for its entire lifetime, we propose to choose the best model of the environment *online*, using the latest information available to the robot to select from a larger model set. As we have emphasized throughout this dissertation, however, no model is perfect– so, we need to better prepare the planner in case this model that we chose proves to be unreliable in practice. To do so, we can leverage the planning approaches that we introduced in Chapter 3 and Chapter 4; broadly, these planners avoid regions of the state and action space where we believe our model to be unreliable. Since each planner is designed

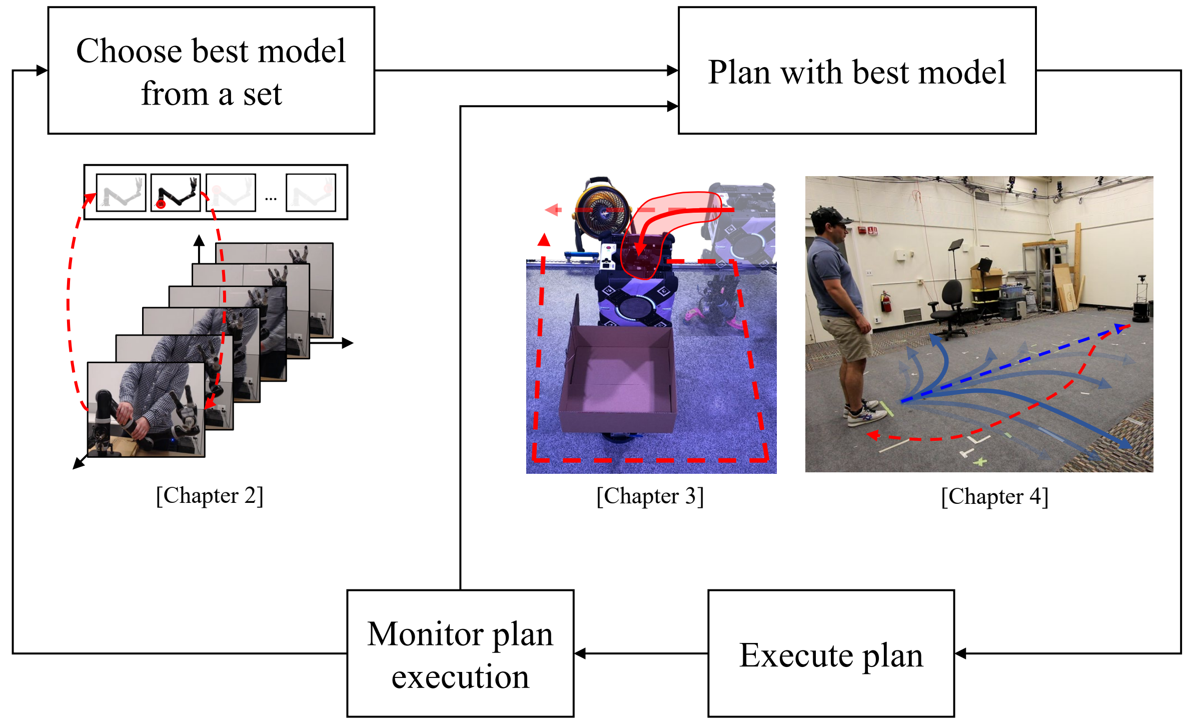

Figure 5.1: We illustrate one way to unify our approaches in Chapter 2, Chapter 3, and Chapter 4, into a single model-based planning and execution framework.

to operate on a different class of models— the former on deterministic models, and the latter on stochastic models— using them together within a single planning framework promises to broaden the type of models we can handle within this framework.

This unified framework presents a natural composition of the approaches we have presented in this dissertation; however, there are several practical challenges that we must address before implementing it, which we enumerate in the remainder of this section. In the following section, we propose some approaches to extend this framework to a broader set of unreliable models.

What set of models is sufficient for the task?

One key component of the unified framework in Fig. 5.1 is a mechanism to choose a best model to use in planning. In Chapter 2, we presented an approach to efficiently choose this best model from a set, and demonstrated the advantages of this over assuming a fixed model for several applications. However, throughout Chapter 2, we assumed that we were already given a *sufficient* set of models for the task— but, what exactly defines a *sufficient* set of models?

Broadly, a sufficient set of models would ideally reliably cover all possible scenarios the robot may encounter during its lifetime and for its desired tasks. Developing tools to analyze and possibly provide guarantees on the coverage of a particular model would be valuable from the perspective of system design— and hence is an important direction for future research. Since computational requirements scale with the number of models in the robot's set, such tools could provide us with ways to choose the minimal number of models that guarantees the required level of coverage. For example, if there are two different models that, when used by the planner, result in the same plan, then perhaps the robot only needs to keep one of them.

Should we replan with the same model or choose a new model?

In the unified framework in Fig. 5.1, the robot first plans a path to the goal, using the best model available. Then, the robot executes that plan; during execution, the robot monitors for any unexpected deviations from the plan. If that occurs, then the robot must find an alternate path to the goal. As discussed, however, simply replanning may not necessarily be sufficient— instead, we can either change to a new model, if a better one is available, or keep the same model, and simply bias replanning away from where we believe that model to be inaccurate.

This choice is sometimes simple. For example, if there is no better model available according to the robot's current information, then we have no choice but to keep the same model; however, if there *is* a better model available, then we must decide whether to change to that new model, or keep the existing one, and bias the planner away from where we believe that model to be unreliable, e.g., using the approach in Chapter 3. Always changing the model when a better one is available is likely a good first solution to this. Nonetheless, how the model selection and planning are integrated can significantly impact the overall performance of this framework. For example, changing the model and keeping the same model may result in two very different plans, one of which may be more efficient than the other; it is not clear that one strategy is always better than the other. Therefore, it is important to address this question more thoroughly in future research.

Should uncertainty over the best model influence the robot's behavior?

In Fig. 5.1, we show choosing the model and planning with that model as being decoupled aspects of the system. However, there are situations where there is a lot of ambiguity over which model best describes the observed behavior of the system, which in turn leads to high uncertainty over what "best" model we should choose for use in planning. For example, consider again the robot arm in Fig. 1.2. If the robot is unable to rotate one of its joints, it may not be clear whether this is because an actuator failed, the robot made unexpected contact with the environment, a human is interacting with the robot, and so on, based on the available sensor information (encoders, force torque sensors, etc.) For each of these situations, we may desire the robot to exhibit a different behavior to complete its task— but the robot will be unable to plan the correct one if it does not know which model to choose. Therefore, the robot will either plan with a wrongly chosen model— and hence come up with an unsuitable behavior for the task— or, it may quickly switch between a set of likely models and replan; in either case, the performance of the robot's planning and execution will suffer.

One solution would be to plan a motion that *actively* attempts to disambiguate between several equally likely models. This could help overall task performance, since the robot may be able to identify the most accurate model earlier on, and therefore will be able to proactively plan accordingly. Of course, there are some important trade-offs that we would need to consider, such as how much time to spend on exploratory or information gathering actions versus exploiting the current best model to complete the task quickly or efficiently. We believe that extending the current framework in this way is an interesting direction for future research; in fact, it may be possible to utilize existing ideas around exploration in Reinforcement Learning (RL) [76], or concepts in Active Learning [16], to actually implement such a system design. Furthermore, a similar problem has been studied in the context of dynamical systems, i.e., System Identification [5], where the concept of persistent excitation provides guidance on how to choose informative actions for the purposes of parameter identification.

Rather than choosing a single best model to plan with, it may also be possible to plan over a *set* of possible models. One benefit of this is that we no longer need to choose a single best model to plan with — instead, we defer reasoning about this to the planner. In fact, such a formulation may naturally encode the exploratory or information gathering behavior that we discussed earlier, if we encoded the planning problem, e.g., as a Partially Observable Markov Decision Process (POMDP). A key challenge, however, is computational — planning under uncertainty, e.g. with POMDPs, is known to have high computational time and space requirements to find a solution. Nonetheless, we believe this is also a promising direction for future research.

5.2 Handling a Broader Set of Unreliable Models

Bias replanning away from unreliable regions in probabilistic models

Both planning approaches in this dissertation, presented in Chapter 3 and Chapter 4, share the common idea of avoiding regions where we believe the model to be inaccurate, and otherwise relying on the model's predictions as usual in planning. Each, however, has its limitations: the approach in Chapter 3 is restricted to *deterministic* models, i.e., where the model predicts a single outcome for each state and action; and, the approach in Chapter 4 focuses on *stochastic* models, i.e., where the model predicts a probability distribution over multiple possible outcomes, and more specifically on a class of models tailored for human motion prediction. Therefore, extending and possibly combining these approaches would broaden the types of unreliable models that a combined framework could deal with.

In Chapter 3, we use information gathered during execution to understand where the model is inaccurate; we then use this understanding to bias replanning away from these inaccurate regions. In this way, we are able to leverage the model where it is accurate, and avoid relying on it all together where it is inaccurate. So far, we have only shown how this approach works for settings where the underlying model is deterministic. One challenge of extending this idea to stochastic models is that it is more difficult to define whether the model has accurately predicted the outcome of a state and action (i.e., transition), if we have only one observation of the robot executing that transition in the real-world. For example, if we observe an outcome that has low likelihood under the model, it is unclear whether that happened by chance, or because our model's prediction was wrong. On the other hand, if the robot is performing a repetitive task, we may be able observe the same transition executed multiple times, and hence be able to better determine whether or not the model's predicted distribution over the outcome is accurate. How exactly to do this, however, is not trivial, and therefore is an important direction for future research.

Our approach in Chapter 4 instead focuses specifically on stochastic models, introducing the idea of only relying on the model to predict what is very unlikely, and planning conservatively (i.e., assuming the worst-case outcome) everywhere else. While stochastic models are more general than deterministic models, one drawback of this approach is that our decision of where to rely on the model is fixed *offline*, in contrast our approach in Chapter 3. Therefore, it would be interesting to see if we could extend this approach to combine information gathered online to update our understanding about the reliability of the model. This, in turn, could be used to adapt where we rely on the model, and hence better trade-off between conservatism and efficiency in the planning.

Learning new models during execution

In all the work presented so far in this dissertation, there is always an assumption that some model, or part of some model, which we have constructed prior to execution, can reliably capture the relevant aspects of the robot's environment. However, in some cases, this assumption may not hold.

Consider, for example, a deep space exploration mission where little is known about the location where the robot is sent prior to arrival. It may not be possible to construct a reliable model of the environment *a priori*; yet at the same time, the robot may be required to act quickly, as time is limited — e.g., since the robot will have a short lifetime due to extreme operating conditions — and with little interaction with remote operators due to high communication latency.

One possible solution is to add the ability to *learn a new model* using observations collected at execution time to the framework in Fig. 5.1. This fits naturally into the approach we introduced in Chapter 2, since there is no requirement that the full model set needs to be fixed beforehand. Indeed, it is possible to have that set grow over time as the robot learns new models. While there has already been extensive research around learning a model from this kind of data, e.g., in model-based RL [63], model learning [57], and learning-based control [21], several challenging questions remain. For example, it may not always be necessary or desirable to learn a new model from scratch— so, when is it better to update an existing model compared to learning a completely new model? Uncertainty over the current model adds to this challenge, because what model should be updated with the observed data if there are several possible models? Related to this, the data may be very sparse, so does this constrain the type of model that such a system can learn? Regardless, we believe that this model learning is a key direction of future research towards broadening the types of unreliable models that our framework can handle.

Finally, closely related to learning models is the idea of generating new models by combining existing ones. For example, is there a meaningful way to combine several models, e.g., via some weighted combination, to produce new and useful predictions from an existing set?

Chapter 6

Conclusion

Models play an important role in the planning and execution systems of many modern robotic systems. Specifically, the model enables a robot to forward-simulate the outcomes of various actions it may potentially take in the real-world, which is an important component of finding an efficient plan for completing a desired task. Unfortunately, however, it is often very difficult to accurately model the real-world. As a result, it is common for models to make inaccurate predictions about the outcomes of actions. When a robot relies on such models in planning, its task performance suffers. Motivated by this, we have investigated how to increase the robustness of planning and execution frameworks that must use unreliable models of the real-world.

In this dissertation, we have introduced methods to improve two key aspects of model-based planning and execution systems: choosing a model, and planning with that model. To that end, in the first part of this dissertation, we focused on improving the first aspect—namely, choosing the best model that we can given all the available information. Specifically, in Chapter 2, we presented an approach for choosing the best model from a set, which alleviates the need to come up with a single reliable model prior to deploying the robot. Unfortunately, however, no model is perfect— even the best model we can choose may be unreliable in certain regions of the state and action space. Therefore, in the second part of this dissertation, we develop approaches that enable a planner to avoid utilizing parts of the model we believe to be unreliable. Specifically, Chapter 3 focused on deterministic models, and introduced a method for biasing the planner away regions in the state and action spaces where we believe the model will make inaccurate predictions. Chapter 4 focused instead on stochastic models in settings where the robot operates in an environment shared with a human. We introduced a method for finding a balance between conservatism and efficiency in planning, when we believe the predictive model of the human's motion is unreliable.

Finally, in Chapter 5, we provided some avenues for future work concentrated around: unifying the approaches in this dissertation into a single model-based planning and execution framework; and, extending the set of unreliable models that such a framework can handle.

Bibliography

[1] G. Ackerson and K. Fu. "On state estimation in switching environments". In: *IEEE Transactions on Automatic Control* 15.1 (Feb. 1970), pp. 10–17. ISSN: 2334-3303. DOI: 10.1109/TAC.1970.1099359.

[2] Anayo K Akametalu et al. "Reachability-based safe learning with Gaussian processes." In: *53rd Conference on Decision and Control (CDC)*. Citeseer. 2014, pp. 1424–1431.

[3] Heni Ben Amor et al. "Interaction primitives for human-robot cooperation tasks". In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 2831–2837.

[4] Veronica E Arriola-Rios et al. "Modeling of deformable objects for robotic manipulation: A tutorial and review". In: *Frontiers in Robotics and AI* (2020), p. 82.

[5] Karl Johan Åström and Peter Eykhoff. "System identification—a survey". In: *Automatica* 7.2 (1971), pp. 123–162.

[6] Haoyu Bai et al. "Intention-aware online POMDP planning for autonomous driving in a crowd". In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 454–460.

[7] Andrea Bajcsy et al. "A robust control framework for human motion prediction". In: *IEEE Robotics and Automation Letters* 6.1 (2020), pp. 24–31.

[8] Tirthankar Bandyopadhyay et al. "Intention-aware motion planning". In: *Algorithmic Foundations of Robotics X*. Springer, 2013, pp. 475–491.

[9] H. A. P. Blom and Y. Bar-Shalom. "The interacting multiple model algorithm for systems with Markovian switching coefficients". In: *IEEE Transactions on Automatic Control* (1988). ISSN: 2334-3303. DOI: 10.1109/9.1299.

[10] Maria Bualat et al. "Astrobee: Developing a free-flying robot for the international space station". In: *AIAA SPACE 2015 Conference and Exposition*. 2015, p. 4643.

[11] Robert R Burridge, Alfred A Rizzi, and Daniel E Koditschek. "Sequential composition of dynamically dexterous robot behaviors". In: *The International Journal of Robotics Research* 18.6 (1999), pp. 534–555.

[12] Sergio Casas, Wenjie Luo, and Raquel Urtasun. "Intentnet: Learning to predict intention from raw sensor data". In: *CoRL*. 2018, pp. 947–956.

[13] C. B. Chang and M. Athans. "State Estimation for Discrete Systems with Switching Parameters". In: *IEEE Transactions on Aerospace and Electronic Systems* AES-14.3 (May 1978), pp. 418–425. ISSN: 2371-9877. DOI: 10.1109/TAES.1978.308603.

[14] Mo Chen et al. "Fastrack: a modular framework for real-time motion planning and guaranteed safe tracking". In: *IEEE Transactions on Automatic Control* 66.12 (2021), pp. 5861–5876.

[15] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. "Flow*: An analyzer for non-linear hybrid systems". In: *CAV*. 2013, pp. 258–263.

[16] David A Cohn, Zoubin Ghahramani, and Michael I Jordan. "Active learning with statistical models". In: *Journal of artificial intelligence research* 4 (1996), pp. 129–145.

[17] Antoine Cully et al. "Robots that can adapt like animals". In: *Nature* 521.7553 (2015), pp. 503–507.

[18] Hao Ding et al. "Human arm motion modeling and long-term prediction for safe and efficient human-robot-interaction". In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2011, pp. 5875–5880.

[19] Katherine Driggs-Campbell, Roy Dong, and Ruzena Bajcsy. "Robust, Informative Human-in-the-Loop Predictions via Empirical Reachable Sets". In: *IEEE Trans. on Intelligent Vehicles*. 2018.

[20] Chelsea Finn, Sergey Levine, and Pieter Abbeel. "Guided cost learning: Deep inverse optimal control via policy optimization". In: *ICML*. 2016, pp. 49–58.

[21] Jaime F Fisac et al. "A general safety framework for learning-based control in uncertain robotic systems". In: *Transactions on Automatic Control* (2018).

[22] Lorenzo Fluckiger et al. "Astrobee robot software: A modern software system for space". In: *iSAIRAS (International Symposium on Artificial Intelligence, Robotics and Automation in Space)*. ARC-E-DAA-TN55483. 2018.

[23] Gianni Gilardi and Inna Sharf. "Literature survey of contact dynamics modelling". In: *Mechanism and machine theory* 37.10 (2002), pp. 1213–1239.

[24] A Girard, CE Rasmussen, and R Murray-Smith. "Gaussian process priors with uncertain inputs: multiple-step-ahead prediction, delovno poro-£ ilo DCS TR-2002-119". In: *University of Glasgow, Glasgow* (2002).

[25] KP Hand, AE Murray, JB Garvin, et al. "Europa Lander Mission. Europa Lander Study 2016 Report". In: *NASA, Tech. Rep. JPL D-97667* (2016).

[26] Thomas J Harris, F Boudreau, and John F Macgregor. "Performance assessment of multivariable feedback controllers". In: *Automatica* 32.11 (1996), pp. 1505–1518.

[27] Peter E Hart, Nils J Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths". In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[28] Masahiko Haruno, Daniel M Wolpert, and Mitsuo Kawato. “Mosaic model for sensorimotor learning and control”. In: *Neural computation* 13.10 (2001), pp. 2201–2220.

[29] Lukas Hewing et al. “On simulation and trajectory prediction with gaussian process dynamics”. In: *Learning for Dynamics and Control*. PMLR. 2020, pp. 424–434.

[30] Michael Hofbaur. *Hybrid Estimation of Complex Systems*. English. Vol. 319. Lecture Notes in Control and Information Sciences. Germany: Springer Verlag, 2005. ISBN: 3-540-25727-6.

[31] Yew Cheong Hou, Khairul Salleh Mohamed Sahari, and Dickson Neoh Tze How. “A review on modeling of flexible deformable object for dexterous robotic manipulation”. In: *International Journal of Advanced Robotic Systems* 16.3 (2019), p. 1729881419848894.

[32] Inseok Hwang, H. Balakrishnan, and C. Tomlin. “Performance analysis of hybrid estimation algorithms”. In: *42nd IEEE International Conference on Decision and Control (IEEE Cat. No.03CH37475)*. Vol. 5. Dec. 2003, 5353–5359 Vol.5. DOI: `10.1109/CDC.2003.1272488`.

[33] Amin G Jaffer and Someshwar C Gupta. “On estimation of discrete processes under multiplicative and additive noise conditions”. In: *Information Sciences* 3.3 (1971), pp. 267–276. ISSN: 0020-0255. DOI: `https://doi.org/10.1016/S0020-0255(71)80010-5`.

[34] Kris M Kitani et al. “Activity forecasting”. In: *European Conference on Computer Vision*. Springer. 2012, pp. 201–214.

[35] Mykel J Kochenderfer et al. “Airspace encounter models for estimating collision risk”. In: *Journal of Guidance, Control, and Dynamics*. 2010, pp. 487–499.

[36] Dorothea Koert et al. “Online learning of an open-ended skill library for collaborative tasks”. In: *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*. IEEE. 2018, pp. 1–9.

[37] Hema Swetha Koppula and Ashutosh Saxena. “Anticipating human activities for reactive robotic response.” In: *International Conference on Intelligent Robots and Systems (IROS)*. 2013, p. 2071.

[38] Shreyas Kousik et al. “Safe trajectory synthesis for autonomous driving in unforeseen environments”. In: *Dynamic Systems and Control Conference*. Vol. 58271. American Society of Mechanical Engineers. 2017, V001T44A005.

[39] Scott Kuindersma et al. “Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot”. In: *Autonomous robots* 40.3 (2016), pp. 429–455.

[40] Przemyslaw A Lasota and Julie A Shah. “A multiple-predictor approach to human motion prediction”. In: *ICRA*. 2017, pp. 2300–2307.

[41] X Rong Li and Vesselin P Jilkov. “Survey of maneuvering target tracking. Part V. Multiple-model methods”. In: *IEEE Transactions on Aerospace and Electronic Systems* 41.4 (2005), pp. 1255–1321.

[42] X Rong Li and CT Leondes. “Hybrid estimation techniques”. In: *Control and Dynamic Systems: Advances in Theory and Applications* 76 (1996), pp. 213–287.

[43] X Rong Li, Youmin Zhang, and Xiaorong Zhi. “Multiple-model estimation with variable structure. IV. Design and evaluation of model-group switching algorithm”. In: *IEEE Transactions on Aerospace and Electronic Systems* 35.1 (1999), pp. 242–254.

[44] X Rong Li, Xiaorong Zwi, and Youmin Zwang. “Multiple-model estimation with variable structure. III. Model-group switching algorithm”. In: *IEEE Transactions on Aerospace and Electronic Systems* 35.1 (1999), pp. 225–241.

[45] Xiao-Rong Li, Yaakov Bar-Shalom, and William Dale Blair. “Engineer’s guide to variable-structure multiple-model estimation for tracking”. In: *Multitarget-multisensor tracking: Applications and advances.* 3 (2000), pp. 499–567.

[46] John Lygeros. “On reachability and minimum cost optimal control”. In: *Automatica*. Vol. 40. 6. Elsevier, 2004, pp. 917–927.

[47] Wei-Chiu Ma et al. “Forecasting Interactive Dynamics of Pedestrians With Fictitious Play”. In: *CVPR*. 2017.

[48] Guilherme Maeda et al. “Active incremental learning of robot movement primitives”. In: *Conference on Robot Learning*. PMLR. 2017, pp. 37–46.

[49] A. Majumdar and R. Tedrake. “Funnel libraries for real-time robust feedback motion planning”. In: *The International Journal of Robotics Research* 36.8 (2017), pp. 947–982.

[50] P.S. Maybeck. *Stochastic Models, Estimation, and Control*. ISSN. Elsevier Science, 1982. ISBN: 9780080960036.

[51] Dale McConachie et al. “Learning when to trust a dynamics model for planning in reduced state spaces”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 3540–3547.

[52] I. Mitchell. “A toolbox of level set methods”. In: *http://www. cs. ubc. ca/mitchell/ToolboxLS/toolboxL* 2004.

[53] I. Mitchell, A. Bayen, and C. J. Tomlin. “A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games”. In: *IEEE TAC*. Vol. 50. 7. 2005, pp. 947–957.

[54] Ian M. Mitchell, A. M. Bayen, and C. J. Tomlin. “A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games”. In: *IEEE Transactions on Automatic Control* 50.7 (2005), pp. 947–957. DOI: `10.1109/TAC.2005.851439`. URL: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1463302`.

[55] Peter Mitrano, Dale McConachie, and Dmitry Berenson. "Learning where to trust unreliable models in an unstructured world for deformable object manipulation". In: *Science Robotics* 6.54 (2021), eabd8170.

[56] Richard M Murray et al. *A mathematical introduction to robotic manipulation*. CRC press, 1994.

[57] Duy Nguyen-Tuong and Jan Peters. "Model learning for robot control: a survey". In: *Cognitive processing* 12.4 (2011), pp. 319–340.

[58] Joseph Norby and Aaron M Johnson. "Fast global motion planning for dynamic legged robots". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 3829–3836.

[59] Michael Noseworthy et al. "Active learning of abstract plan feasibility". In: *Robotics: Science and Systems (RSS)*. 2021.

[60] Dominik van Opdenbosch et al. "Camera-based Indoor Positioning using Scalable Streaming of Compressed Binary Image Signatures". In: *IEEE International Conference on Image Processing (ICIP 2014)*. 2014.

[61] H. A. P. Blom. "An efficient filter for abruptly changing systems". In: *The 23rd IEEE Conference on Decision and Control*. Dec. 1984, pp. 656–658. DOI: 10.1109/CDC.1984.272089.

[62] Tianyang Pan et al. "Failure is an option: Task and Motion Planning with Failing Executions". In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 1947–1953.

[63] Athanasios S Polydoros and Lazaros Nalpantidis. "Survey of model-based reinforcement learning: Applications on robotics". In: *Journal of Intelligent & Robotic Systems* 86.2 (2017), pp. 153–173.

[64] Carl Edward Rasmussen. "Gaussian processes in machine learning". In: *Summer school on machine learning*. Springer. 2003, pp. 63–71.

[65] Amir Rasouli et al. "PIE: A large-scale dataset and models for pedestrian intention estimation and trajectory prediction". In: *ICCV*. 2019, pp. 6262–6271.

[66] Ellis Ratner et al. "Efficient Dynamics Estimation With Adaptive Model Sets". In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 2373–2380.

[67] X. Rong Li. "Multiple-model estimation with variable structure. II. Model-set adaptation". In: *IEEE Transactions on Automatic Control* 45.11 (Nov. 2000), pp. 2047–2060. ISSN: 2334-3303. DOI: 10.1109/9.887626.

[68] Christoph Rösmann et al. "Online trajectory prediction and planning for social robot navigation". In: *AIM*. 2017.

[69] Andrey Rudenko et al. "Human Motion Trajectory Prediction: A Survey". In: *IJRR*. 2019.

[70] Andrey Rudenko et al. "Human motion trajectory prediction: A survey". In: *The International Journal of Robotics Research* 39.8 (2020), pp. 895–935.

[71] Friederike Schneemann and Patrick Heinemann. "Context-based detection of pedestrian crossing intention for autonomous driving in urban environments". In: *IROS*. 2016.

[72] Sumeet Singh et al. "Robust Tracking with Model Mismatch for Fast and Safe Planning: an SOS Optimization Approach". In: *arXiv preprint arXiv:1808.00649* (2018).

[73] Friedrich Solowjow and Sebastian Trimpe. "Event-triggered Learning". In: *Automatica* 117 (July 2020).

[74] Breelyn Styler and Reid Simmons. "Plan-time multi-model switching for motion planning". In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 27. 2017, pp. 558–566.

[75] Breelyn Kane Styler and Reid Simmons. "Robust Efficient Robot Planning through Varying Model Fidelity". In: *Third International Workshop on Planning and Robotics (PlanRob)*. 2015.

[76] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[77] Russ Tedrake et al. "LQR-trees: Feedback motion planning via sums-of-squares verification". In: *The International Journal of Robotics Research* 29.8 (2010), pp. 1038–1052.

[78] Sebastian Thrun et al. "Stanley: The robot that won the DARPA Grand Challenge". In: *Journal of field Robotics* 23.9 (2006), pp. 661–692.

[79] Chris Urmson et al. "Autonomous driving in urban environments: Boss and the urban challenge". In: *Journal of field Robotics* 25.8 (2008), pp. 425–466.

[80] Anirudh Vemula, J Andrew Bagnell, and Maxim Likhachev. "CMAX++: Leveraging experience in planning and execution using inaccurate models". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 7. 2021, pp. 6147–6155.

[81] Anirudh Vemula et al. "Planning and execution using inaccurate models with provable guarantees". In: *Robotics: Science and Systems (RSS)*. 2020.

[82] R. Walambe et al. "Optimal trajectory generation for car-type mobile robot using spline interpolation". In: *IFAC*. 1. 2016, pp. 601–606.

[83] Xiao-Rong Li and Y. Bar-Shalom. "Multiple-model estimation with variable structure". In: *IEEE Transactions on Automatic Control* 41.4 (Apr. 1996), pp. 478–493. ISSN: 2334-3303. DOI: 10.1109/9.489270.

[84] Hang Yin, Anastasia Varava, and Danica Kragic. "Modeling, learning, perception, and control methods for deformable object manipulation". In: *Science Robotics* 6.54 (2021), eabd8803.

[85] Jihong Zhu et al. "Challenges and outlook in robotic manipulation of deformable objects". In: *IEEE Robotics & Automation Magazine* 29.3 (2022), pp. 67–77.

[86] Brian D. Ziebart et al. "Planning-based prediction for pedestrians". In: *IROS*. 2009, pp. 3931–3936.