2021

Stochastic Motion Planning For Mobile Robots

Ke Sun
University of Pennsylvania

Stochastic Motion Planning For Mobile Robots

Abstract

Stochastic motion planning is of crucial importance in robotic applications not only because of the imperfect models for robot dynamics and sensing but also the potentially unknown environment. Due to efficiency considerations, practical methods often introduce additional assumptions or heuristics, like the use of separation theorem, into the solution. However, there are intrinsic limitations of practical frameworks that prevent further improving reliability and robustness of the system, which cannot be addressed with minor tweaks. Therefore, it is necessary to develop theoretically justified solutions to stochastic motion planning problems. Despite the challenges in developing such solutions, the reward is unparalleled due to their wide impact on a majority of, if not all, robotic applications. The overall goal of this dissertation is to develop solutions for stochastic motion planning problems with theoretical justifications and demonstrate their superior performance in real world applications.

In the first part of this dissertation, we model the stochastic motion planning problem as Partially Observable Markov Decision Processes (POMDP) and propose two solutions featuring different optimization regimes trading off model generality and efficiency. The first is a gradient-based solution based on iterative Linear Quadratic Gaussian (iLQG) assuming explicit model formulations and Gaussian noises. The special structure of the problem allows a time-varying affine policy to be solved offline and leads to efficient online usage. The proposed algorithm addresses limitations of previous works on iLQG in working with nondifferentiable system models and sparse informative measurements. The second solution is a sampled-based general POMDP solver assuming mild conditions on the control space and measurement models. The generality of the problem formulation promises wide applications of the algorithm. The proposed solution addresses the degeneracy issue of Monte Carlo tree search when applied to continuous POMDPs, especially for systems with continuous measurement space. Through theoretical analysis, we show that the proposed algorithm is a valid Monte Carlo control algorithm alternating unbiased policy evaluation and policy improvement.

In the second part of this dissertation, we apply the proposed solutions to different robotic applications where the dominant uncertainty either comes from the robot itself or external environment. We first consider the the application of mobile robot navigation in known environment where the major sources of uncertainties are the robot dynamical and sensing noises. Although the problem is widely studied, few work has applied POMDP solutions to the application. By demonstrating the superior performance of proposed solutions on such a familiar application, the importance of stochastic motion planning may be better appreciated by the robotics community. We also apply the proposed solutions to autonomous driving where the dominant uncertainty comes from the external environment, i.e. the unknown behavior of human drivers.In this work, we propose a data-driven model for the stochastic traffic dynamics where we explicitly model the intention of human drivers. To our best knowledge, this is the first work that applies POMDP solutions to data-driven traffic models. Through simulations, we show the proposed solutions are able to develop high-level intelligent behaviors and outperform other similar methods that also consider uncertainties in the autonomous driving application.

Degree Type

Dissertation

Degree Name

Doctor of Philosophy (PhD)

Graduate Group

Electrical & Systems Engineering

STOCHASTIC MOTION PLANNING FOR MOBILE ROBOTS

Ke Sun

A DISSERTATION

in

Electrical and Systems Engineering

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2021

Supervisor of Dissertation

Vijay Kumar, Professor of Mechanical Engineering and Applied Mechanics

Graduate Group Chairperson

Victor Preciado, Associate Professor of Electrical and Systems Engineering

Dissertation Committee

George Pappas, Professor of Electrical and Systems Engineering

Pratik Chaudhari, Professor of Electrical and Systems Engineering

Stephen Chaves, Staff Engineer at Qualcomm Research

STOCHASTIC MOTION PLANNING FOR MOBILE ROBOTS

To my family.

Acknowledgments

After six years, I finally get to the point of finishing my dissertation. It certainly takes longer than I thought. But the joy I receive is also beyond my imagination. Lots of people have contributed to this enjoyable journey, to whom I own a debt of gratitude.

My deepest thanks go to my advisor, Dr. Vijay Kumar. His call in 2015 pulled me out of endless waiting for PhD application results. Over the years, I was provided abundant and unparalleled project opportunities. The experience of working on projects makes me deeply appreciate his philosophy of looking for research problems from real world applications. I am also grateful for his encouragement on my theoretical research and challenging me to apply them beyond simulations, without which much of this dissertation would not be possible. Meanwhile, I want to thank my other mentors at different points of my PhD career. Dr. C.J. Taylor and Dr. Nikolay Atanasov have patiently guided me through the early years. I also thank my dissertation committee members, Dr. Stephen Chaves, Dr. Pratik Chaudhari, and Dr. George Pappas for being interested in learning, discussing, and reviewing my work.

I feel fortunate to work in such a lab full of talent. Over the years, I have learned so much from each and every one of them. I want to thank my teammates, Kartik, Sikang, Yash, Mike, Bernd, Jeremy, and Shreyas, on the FLA project. The qualities they demonstrated, the persistence to see a problem through, the courage to step outside your comfort zone, the calmness when things go south in a demonstration, the playfulness when the outcome is against your expectation, all have shaped my research attitude in later years.

My research work would not be possible without the help of my coauthors, Kelsey, Brent, Paul, and Gulshan. Our work would not be possible without their valuable contribution. I also thank my peers in MRSL and GRASP for many insightful and inspiring discussions. Thanks to all of you, I was never alone along this journey.

Finally, I want to express my gratitude to my family. I thank my parents for their support, encouragement, and sacrifices since I was young. Their constant inspiration is what drives me to pursuit a career in engineering. I thank my parents-in-law for their understanding the long-term endeavor required for a PhD and our not being able to come home often. Specially, I want to thank my wife who has been with me on this journey since the very first day. She has always been there to provide fresh views during my perplexity, cheer me up during my frustration, and share the joy during my accomplishments. I appreciate all she has done to our family over the years while I was "playing with computers". I should make a note to my son as well in case he stumbles on his dad's dissertation someday and feels left out. Of course, there is more to that. His arrival has brought a new aspect to my life and makes me a better person.

Getting a PhD is definitely not as easy as it looks. Especially for the last few years, the global pandemic, the delicate political environment, and the new role of being a father, all have contributed to this already challenging journey. I know it is because of all of you that I mentioned above and many more I failed to acknowledge, I am still doing what I love. For that, I thank you very much.

ABSTRACT

STOCHASTIC MOTION PLANNING FOR MOBILE ROBOTS

Ke Sun

Vijay Kumar

Stochastic motion planning is of crucial importance in robotic applications not only because of the imperfect models for robot dynamics and sensing but also the potentially unknown environment. Due to efficiency considerations, practical methods often introduce additional assumptions or heuristics, like the use of separation theorem, into the solution. However, there are intrinsic limitations of practical frameworks that prevent further improving reliability and robustness of the system, which cannot be addressed with minor tweaks. Therefore, it is necessary to develop theoretically justified solutions to stochastic motion planning problems. Despite the challenges in developing such solutions, the reward is unparalleled due to their wide impact on a majority of, if not all, robotic applications. The overall goal of this dissertation is to develop solutions for stochastic motion planning problems with theoretical justifications and demonstrate their superior performance in real world applications.

In the first part of this dissertation, we model the stochastic motion planning problem as Partially Observable Markov Decision Processes (POMDP) and propose two solutions featuring different optimization regimes trading off model generality and efficiency. The first is a gradient-based solution based on iterative Linear Quadratic Gaussian (iLQG) assuming explicit model formulations and Gaussian noises. The special structure of the problem allows a time-varying affine policy to be solved offline and leads to efficient online usage. The proposed algorithm addresses limitations of previous works on iLQG in working with nondifferentiable system models and sparse informative measurements. The second solution is a sampled-based general POMDP solver assuming mild conditions on the control space and measurement models. The generality of the problem formulation promises wide

applications of the algorithm. The proposed solution addresses the degeneracy issue of Monte Carlo tree search when applied to continuous POMDPs, especially for systems with continuous measurement space. Through theoretical analysis, we show that the proposed algorithm is a valid Monte Carlo control algorithm alternating unbiased policy evaluation and policy improvement.

In the second part of this dissertation, we apply the proposed solutions to different robotic applications where the dominant uncertainty either comes from the robot itself or external environment. We first consider the the application of mobile robot navigation in known environment where the major sources of uncertainties are the robot dynamical and sensing noises. Although the problem is widely studied, few work has applied POMDP solutions to the application. By demonstrating the superior performance of proposed solutions on such a familiar application, the importance of stochastic motion planning may be better appreciated by the robotics community. We also apply the proposed solutions to autonomous driving where the dominant uncertainty comes from the external environment, *i.e.* the unknown behavior of human drivers. In this work, we propose a data-driven model for the stochastic traffic dynamics where we explicitly model the intention of human drivers. To our best knowledge, this is the first work that applies POMDP solutions to data-driven traffic models. Through simulations, we show the proposed solutions are able to develop high-level intelligent behaviors and outperform other similar methods that also consider uncertainties in the autonomous driving application.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Robots may be changing our life at a faster pace than we expect. Precision agriculture (Figure 1.1a), personal robotics (Figure 1.1b), search and rescue mission (Figure 1.1c), first response applications (Figure 1.1d), autonomous driving (Figure 1.1e), cellular and chemical delivery (Figure 1.1f), and infrastructure inspection (Figure 1.1g), all these applications and beyond have the potential to change or have already changed the world as we know it. In order to have the robot complete the tasks required by these applications, one has to answer the simply phrased yet profound question: *how to get from A to B.*[1] The key to find answers of the question lies with motion planning algorithms.

Depending on the assumptions imposed on the system, motion planning problems can be, in general, categorized into the following three broad categories:

- *Deterministic motion planning*

  Problems in this category assume a deterministic system model and a known initial state, for which an open-loop control policy suffices. Such problems are well addressed by search-based algorithms such as A* (Hart et al., 1968), or sampling-based algorithms such as RRT, RRT*, and PRM* (Karaman and Frazzoli, 2011).

[1] A and B can be locations as in autonomous driving. More often, A and B are abstract states. As in the first response application, we hope to become fully situation-aware of an initially unknown environment.

- *Motion planning with stochastic dynamics and perfect state information*

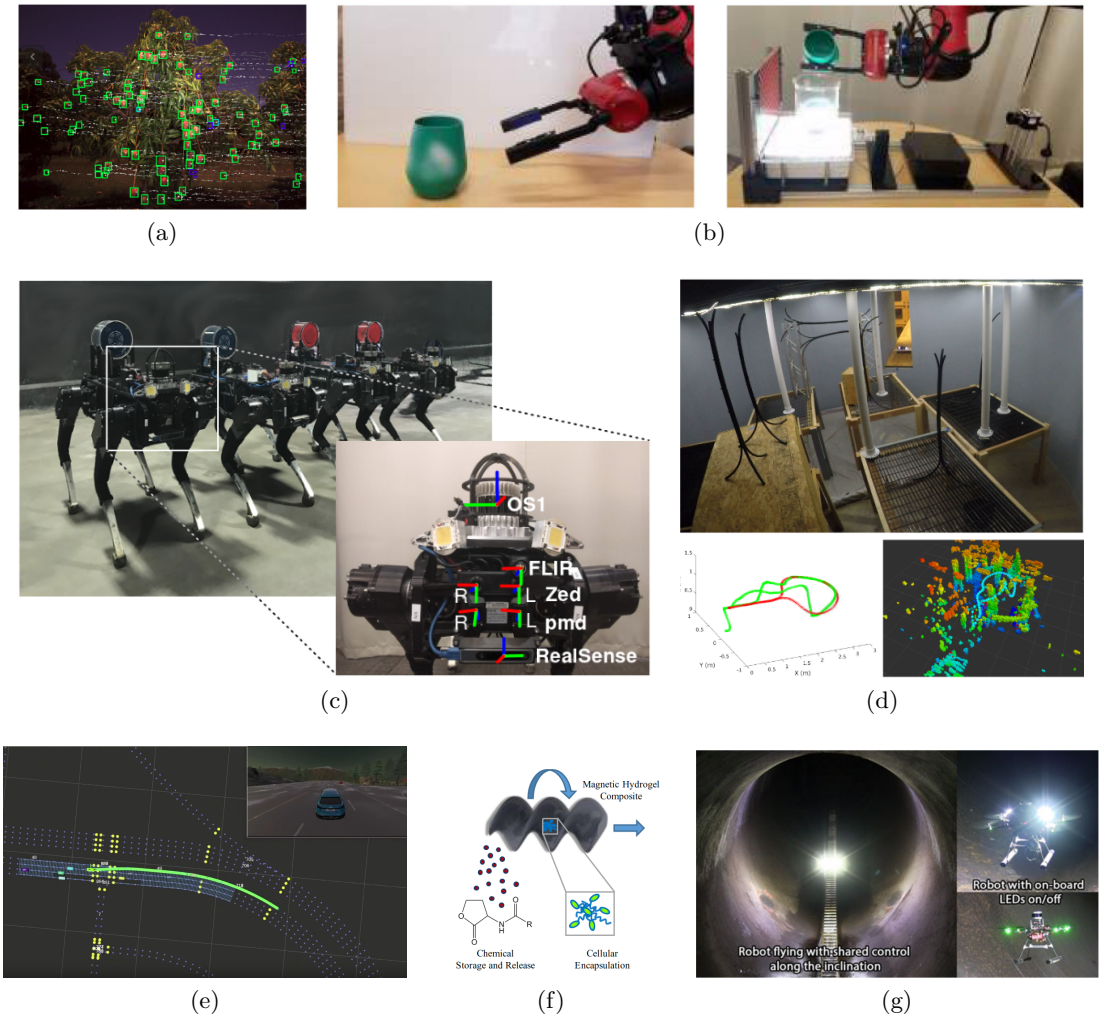
  In this class, uncertainty of system dynamics is taken into account, while perfect state information is assumed to be accessible. Receding Horizon Control (RHC) (Mayne and Michalska, 1990), combined with deterministic motion planning algorithms, directly lends itself to the generation of a feedback control policy for such problems, which implements the idea of certainty equivalent control (Bertsekas et al., 2017, Ch.6.2). However, in order to obtain an optimal feedback policy, it requires motion planning algorithms to consider motion uncertainty explicitly (Melchior and Simmons, 2007; Alterovitz et al., 2007; Tedrake et al., 2010).

- *Motion planning with stochastic systems and imperfect state information*

  Compared with the last two categories, problems in this category provide the most general modeling of a robotic system, which not only respect uncertainty originated from system dynamics and sensor data but an unknown initial state. Practitioners often extend the idea of the separation theorem, and rely on the state estimation algorithms to provide an accurate estimate. Then, methods from the second category could be applied directly assuming perfect state information.

In this dissertation, we consider the problems in the third category, and specifically refer to such problems as *stochastic motion planning*. Problems in this category try to compensate imperfect system models with stochastic processes, *i.e.* noise, solving which could potentially improve safety, robustness, and reliability of systems in completing tasks.

In the following, we provide an overview of the practical approach to stochastic motion planning problems through the lens of an autonomous flight project with an Micro Aerial Vehicle (MAV). Despite the specificity of the application, the hardware configuration and the software framework transfer to a wide range of applications and various robotic platforms (Montemerlo et al., 2008; Salavasidis et al., 2019). By exposing the limitations of the practical approach, we motivate the necessity of seeking new solutions to stochastic motion planning problems, especially the ones with theoretical justifications.



(a) (b) (c) (d) (e) (f) (g)

Figure 1.1: Applications of robots relevant to various aspects of our life. (a) Fruit counting with a monocular camera (Liu et al., 2019). (b) Precise pouring with a robotic arm from an unknown container (Kennedy et al., 2019). (c) Searching for targets in a tunnel environment with a quadrupedal robot team (Miller et al., 2020). (d) Mapping a disastrous environment with a MAV (Thakur et al., 2020). (e) Autonomous driving on highways (Sun et al., 2020a). (f) Cellular and chemical delivery with micro robots (Hunter et al., 2018). (g) Penstock inspection with multirotor aerial vehicles (Özaslan et al., 2017).

1.1 A Motivating Case Study

Motivated by the search and rescue mission, the goal of the FLA project[2] is to develop a complete solution, including both hardware and software, that could navigate a MAV from point A to B and back in an unknown environment. A lightweight platform is preferred so that the robot not only moves fast but is agile to maneuver. In addition, the robot should be able to complete the task autonomously using onboard sensing only, while external intervention from human operators is disabled. In the following, we provide a brief overview of the hardware platforms and software framework that we developed in this project followed by a summary of the outcome. Then we show a few representative failure cases to study the limitations of the current solution. More details about our system can be found in Mohta et al. (2018b) and Mohta et al. (2018a).

1.1.1 Hardware and Software Solutions

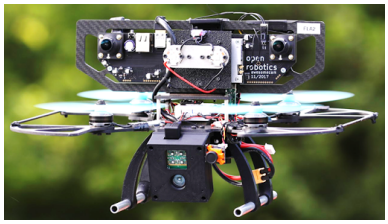
We have developed quadrotor platforms of different sizes, shown in Figure 1.2, adapting to different target environments. The sensor configurations across the platforms are mostly the same, which include monochrome stereo cameras, IMUs, and a downward single-beam Lidar. An additional color camera is installed at the front to detect the landmark (a red barrel) at the goal location. The major difference of the sensors lies with the mapping device. Based on the payload of different platforms, we use a 3-D Lidar[3], a gimbled 2-D Lidar[4], or a time-of-flight depth sensor[5] to create maps of the environment. For computation, the large platforms support Intel NUC[6]. Nvidia Jetson TX2[7] is slightly slower but lighter, which is installed on the small platform. As shown in Figure 1.2d, we integrate the computer and sensors on a single PCB board (Quigley et al., 2019) to further reduce the weight of the small platform.

[2] https://www.darpa.mil/program/fast-lightweight-autonomy
[3] https://velodynelidar.com/products/puck/
[4] https://hokuyo-usa.com/products/lidar-obstacle-detection/utm-30lx
[5] https://pmdtec.com/en/technology/time-of-flight/
[6] https://www.intel.com/content/www/us/en/products/details/nuc.html
[7] https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/

(a)

(b)

(c)



(d)

Figure 1.2: Different quadrotor platforms developed in the FLA project. (a) shows the relative size of the platforms. From left to right, the diameters (motor-to-motor) of the quadrotors are 450mm, 350mm and 250mm, with weight ranges from 3.4kg to 1.3kg. (b, c, d) show the sensor configurations. All platforms are equipped with stereo cameras, IMUs and downward single-beam Lidars. Based on the payload of the platforms, we use a 3-D Lidar, a gimbled 2-D Lidar, or a time-of-flight depth sensor for mapping respectively.

As shown in Figure 1.3, the software stack consists of three main modules for estimation, planning, and control respectively. In the estimation module, we apply a stereo visual inertial odometry (Sun et al., 2018) to estimate the current state of the robot, which is then fused with other sensors to bump up the estimation frequency (Mohta et al., 2018a). Point clouds from Lidars or depth sensors are directly registered in the map using the estimated pose for efficiency. Based on the estimated robot state and map, components in the planning module generate reference trajectories. For open environments, we developed algorithms based on either optimization (Liu et al., 2017b) or graph search (Liu et al., 2017a) to plan feasible trajectories considering the dynamics of the robot. In contrast, the speed is significantly lower indoors where we could switch to a simpler line tracking algorithm only considering kinematics. Finally, the control algorithm in the last module

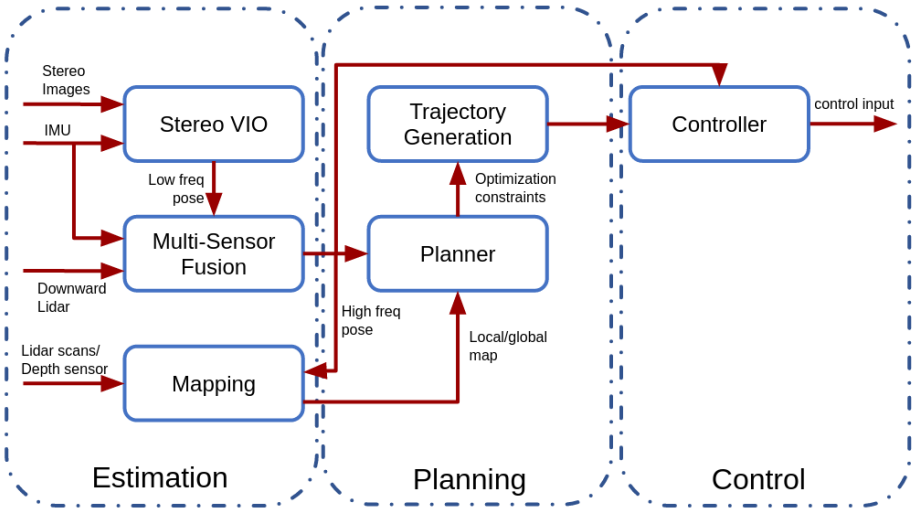


Figure 1.3: The software architecture designed for autonomous flight of a MAV.

generates commands to ensure the robot could closely follow the reference trajectory.

1.1.2 Results and Limitations

We have successfully tested the quadrotor platforms in a variety of environments including indoors and outdoors as shown in Figure 1.6. Figure 1.6e shows a representative complete mission. To complete the task in this case, the robot has to first traverse through a wooded area, then find the only opening of the warehouse located on the opposite side, get to the target in the warehouse, and finally go back to the starting position. The total distance of the round trip is about 700m. Many elements of the mission are considered challenging for the state-of-the-art autonomous flight solutions, such as long-distance traveling, cluttered environments, indoor-outdoor transitions, *etc.*

We also observe failure cases with two examples shown in Figure 1.4[8]. In Figure 1.4a, the robot fails to detecting branches because of the limited angle resolution of the range sensor, and gets tangled with the tree on the right. In Figure 1.4b, the robot turns while facing a blank wall. Due to the lack of features, the estimation uncertainty of visual odometry grows significantly. The robot collides into the wall within the next few seconds due to

[8]See `failure_case_tree_tangle.mp4` and `failure_case_warehouse_collision.mp4` for supplementary videos for the failure cases.

(a) (b)

Figure 1.4: Failure examples in the field tests. (a) the robot get tangled with the tree since it does not see the thin branches due to the limited angle resolution of the range sensor. (b) during turning, the robot faces a blank wall which is notoriously challenging for visual odometries. As a result, the uncertainty of estimation increases significantly. The robot collide into the wall within the next few seconds.

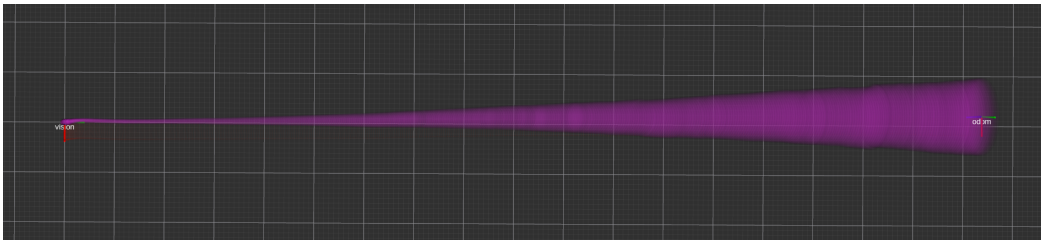


Figure 1.5: The growth of position uncertainty (one standard deviation) estimated by our stereo VIO (Sun et al., 2018) as the quadrotor travels along a straight line with top speed close to 18m/s. For scale reference, the solid grid lines are of 10m resolution.

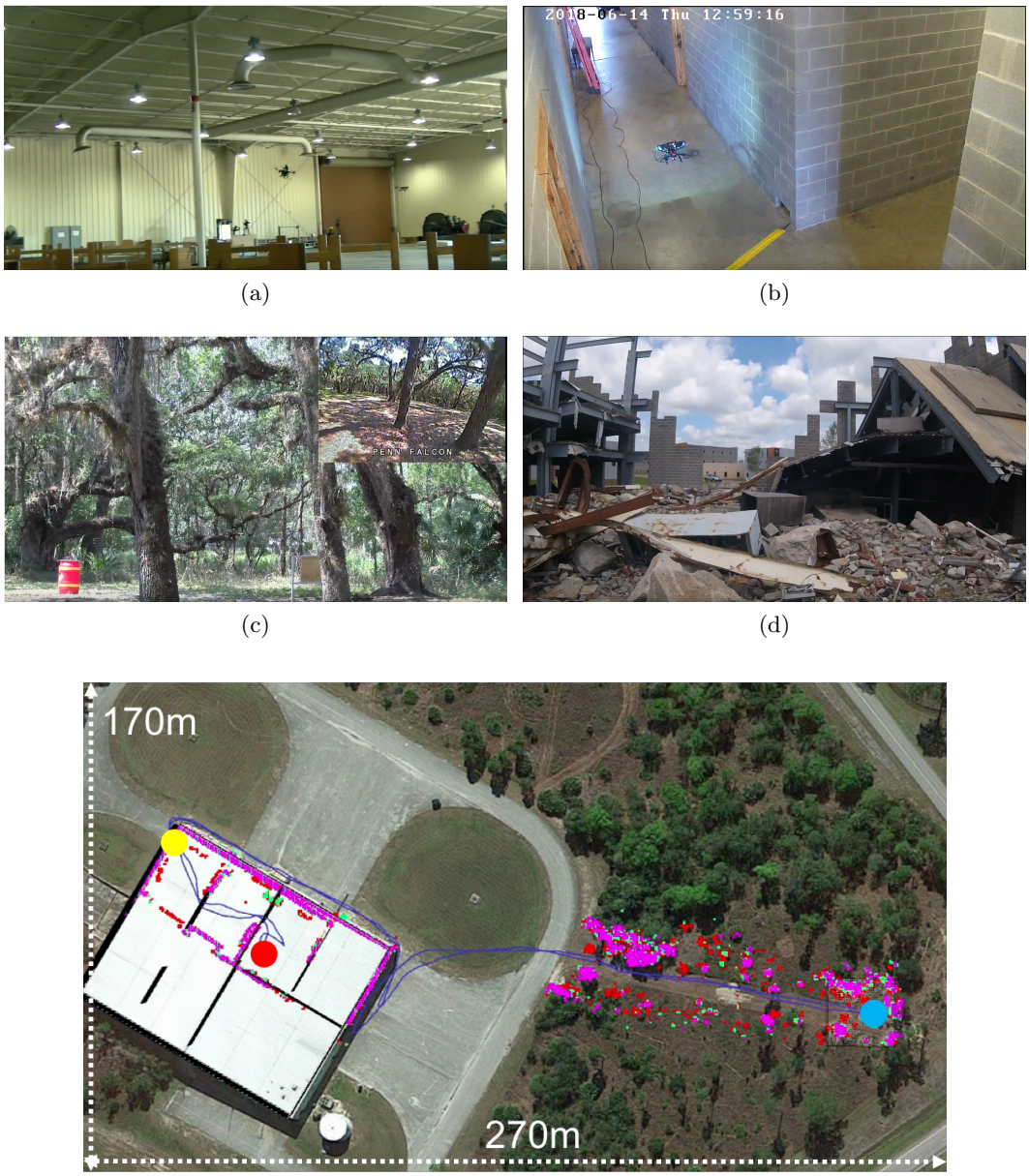
the large estimation error. To summarize, both failure cases are caused by the fact that the planner or controller does not consider estimation uncertainty, either uncertainty of maps or of state estimates. This limitation is embedded into the software structure since the adoption of separation theorem. The separation theorem allows planners and controllers to assume perfect state knowledge. Therefore, the whole software stack can be decoupled into separate modules, which eases the development process. Unfortunately, it also rules out the possibility of a proper treatment of estimation uncertainty during decision making.

Despite the huge amount of effort in improving estimation accuracy over the past decades, estimation algorithms are still subject to theoretical limitations, such as observability issues with Lidar or visual odometries. Non-degeneracy of such algorithms not only depends on the external environment (structure geometry or feature density) (Zhang et al.,

2016), but also the motion of the robot (Martinelli, 2012; Wu and Roumeliotis, 2016). As shown in Figure 1.5, we test our stereo VIO with a straight line constant velocity flight. The top speed is close to 18m/s. As the quadrotor travels, the standard deviation along y-axis of the robot can grow up to 8m in a few seconds. The large uncertainty can easily cause divergence of the estimation, or collision if there is any obstacle. Addressing the issue would require active intervention of motion planning algorithms.

There are works, often known as perception-aware planning (Falanga et al., 2018; Spasojevic et al., 2020; Watterson et al., 2018), trying to avoid estimation degeneracy by tweaking the existing framework shown in Figure 1.3. The common feature of related works is to introduce perception related costs into the objective function in order to motivate intelligent behaviors like active localization. However, the ideology may not be sufficient for two major reasons. First, the desire of reducing perception uncertainty can conflict with the ultimate goal of completing a task. Consider the case where the robot starts with a known initial state, the optimal way of staying well-localized is to remain static and never move to complete the task. Expected behaviors of the robot can only be obtained by carefully calibrating the weights of different cost terms. In theory, the task, when modeled properly, should have already motivated the robot to actively reduce estimation uncertainty whenever necessary. Second, introducing sensible additional cost terms relies on human heuristics of the desired intelligent behaviors. For relatively simple tasks, like motion planning in a known environment, proposing such heuristics might be straightforward. However, the method does not transfer to more complicated tasks, like exploration, where human fails to have reasonable heuristics of intelligent behaviors, or tasks, like autonomous driving, where intelligent behaviors are hard to encode mathematically.

In the following section, we provide a precise problem formulation for stochastic motion planning. Inspired by the literature of stochastic control (Maybeck, 1982), we formulate the problem considering system stochasticity from various sources, which then provides room for formally addressing the uncertainty in motion planning.



(a) (b) (c) (d) (e)

Figure 1.6: A variety of field test environments. (a) warehouse. (b) hallways. (c) cluttered forest. (d) a debris environment simulating collapsed structures. (e) shows a representative complete mission. In this case, the robot has to navigate from the start (blue) to the goal (red) located in the warehouse and then return. To complete the task, the robot has to first travel through the wooded area, find the only opening of the warehouse (yellow), get to the goal, and finally return to the start. The round trip is about 700m. The colored points overlaid on the satellite image are the global map registered with the estimated robot poses.

1.2 The General Problem Formulation of Stochastic Motion Planning

We assume the system[9] under consideration can be modeled by its transition probability and measurement likelihood, *i.e.*,

$$\begin{aligned} &P(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t), \\ &P(\boldsymbol{z}_t|\boldsymbol{x}_t), \end{aligned} \tag{1.1}$$

where $\boldsymbol{x} \in \mathcal{X}$, $\boldsymbol{u} \in \mathcal{U}$, and $\boldsymbol{z} \in \mathcal{Z}$ are the continuous state, control and measurement of the system. We note (1.1) imposes minimal assumptions on the system modeling where explicit formulations for motion and measurement models are not required. Meanwhile, we do not assume to know the explicit *p.d.f.* of the transition probability or the measurement likelihood. Instead, generative models are sufficient at this moment, *i.e.* given a state, we could sample possible measurements, and with an addition control input, we could further sample the next states.

We assume the task can be modeled by a summation of stage rewards[10] in the form of,

$$V_0(\boldsymbol{b}_0) = \mathop{\mathbb{E}}_{\substack{\boldsymbol{x}_0, \boldsymbol{z}_t, \\ t=0,1,\ldots,N-1}} \left\{ \sum_{t=0}^{N-1} \gamma^t r_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{x}_{t+1}) + \gamma^N r_N(\boldsymbol{x}_N) \right\}, \tag{1.2}$$

where $\boldsymbol{b} \in \mathcal{B}$ is the belief state, *i.e.* the distribution of the system state. N is the planning horizon. $r_t(\cdot) : \mathcal{X}^2 \times \mathcal{U} \mapsto \mathcal{R}$ is the stage reward function[11], while $r_N(\cdot) : \mathcal{X} \mapsto \mathcal{R}$ is the terminal reward function. To ensure (1.2) is well defined, we assume both stage and terminal reward functions are bounded, *i.e.* $|r_t(\cdot)| < \infty$ and $|r_N(\cdot)| < \infty$. The task model in (1.2) could also adapt to infinite horizon cases ($N \to \infty$) where $r_N(\cdot)$ can be trivially set to 0. $\gamma \in (0, 1)$ is the discount factor, the major purpose of which is to ensure that (1.2) remains well-defined even for infinite horizon cases.

[9] The "system" may not only refer to the robot itself. It could also be both the robot and the operating environment when the environment cannot be modeled as a constant known *a prior*.

[10] Equivalently, we could also model the task with cost terms instead of rewards.

[11] It is also common to set the stage reward in the form of $r_t(\cdot) : \mathcal{X} \times \mathcal{U} \mapsto \mathcal{R}$, which only depends on the current state instead of both current and future states.

The goal of planning algorithms is to find a control policy, $\pi = \{\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{N-1}\}$, that drive the robot to complete the task. Mathematically, the optimal control policy, π^*, can be found by optimizing the objective function in (1.2), *i.e.*

$$\pi^* = \arg\max_{\pi} V_0(\boldsymbol{b}_0) = \arg\max_{\pi} \mathop{\mathbb{E}}_{\substack{\boldsymbol{x}_0, \boldsymbol{z}_t \\ t=0,1,\ldots N-1}} \left\{ \sum_{t=0}^{N-1} \gamma^t r_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{x}_{t+1}) + \gamma^N r_N(\boldsymbol{x}_N) \right\}. \quad (1.3)$$

Combining the models for the system and the task in (1.1) and (1.2), we have modeled the stochastic motion planning problem as a Partially Observable Markov Decision Process (POMDP) (Kaelbling et al., 1998). There are some known results of POMDPs. We, here, state a few that are relevant to this dissertation without proofs. More on POMDPs can be found in (Kaelbling et al., 1998; Bertsekas et al., 2017; Thrun et al., 2005).

In POMDPs, one possible sufficient statistic is belief, which summarizes all information in the history that is necessary to make the optimal decision at the current moment. Therefore, POMDPs are also known as belief state MDPs. Leveraging the result, if a feedback control policy is to be found, the optimization in (1.3) can be rewritten as,

$$\pi^* = \arg\max_{\pi} V_0(\boldsymbol{b}_0) = \arg\max_{\pi} \mathop{\mathbb{E}}_{\substack{\boldsymbol{x}_0, \boldsymbol{z}_t \\ t=0,1,\ldots N-1}} \left\{ \sum_{t=0}^{N-1} \gamma^t r_t(\boldsymbol{x}_t, \mu_t(\boldsymbol{b}_t), \boldsymbol{x}_{t+1}) + \gamma^N r_N(\boldsymbol{x}_N) \right\}. \quad (1.4)$$

The control policy in this case is in the form of $\pi = \{\mu_0, \mu_1, \ldots, \mu_{N-1}\}$ with $\mu_t(\cdot) : \mathcal{B} \mapsto \mathcal{U}$. Specially, for infinite horizon cases, there exists a stationary optimal policy, *i.e.* $\pi^* = \left\{\mu_0^*, \mu_1^*, \ldots, \mu_{N-1}^*\right\}$ with $\mu_t^*(\cdot) = \mu^*(\cdot)$ for $t = 0, 1, \ldots, N-1$ (Kaelbling et al., 1998; Porta et al., 2006).

There are cases where POMDP problems are easy to solve. If the underlying system is linear and the objective function is quadratic, the POMDP problem can be decoupled into an estimator and a controller. The estimator is the optimal solution of the problem of estimating the current state assuming no control takes place. The controller, on the other hand, is the optimal solution of the control problem assuming perfect state information.

Together, both solutions constitute the optimal solution of the POMDP problem. Above is the celebrated *separation theorem* that we have mentioned at the beginning of this chapter. Specially, when the noises of the system dynamics and measurement are of Gaussian distributions, Kalman filters and LQRs lend themselves to be the optimal estimator and controller. However, for nonlinear system or more complicated objective functions, the separation theorem no longer applies.

In general, POMDP problems are indeed very hard. The challenges are twofold. First, as discussed before, policies for POMDP problems are functions of beliefs instead of states. Even for a system with finite state space of size n, the belief space is continuous and resides in $\mathbb{R}^{n-1}$. Second, in order to find an optimal policy, one also has to consider all future outcomes alternating controls and measurements, the number of which grows exponentially with the search depth. Pineau et al. (2003) summarize the two challenges vividly as *curse of dimensionality* and *curse of history*.

1.3 Outline and Contributions

This dissertation formulates the stochastic motion planning problem as a general continuous POMDP respecting system uncertainty. We propose solutions of different optimization regimes to address the problem with theoretical justifications. Without introducing additional assumptions or heuristics into the solutions[12], it is expected that the proposed stochastic motion planning algorithms could "discover" high-level intelligent behaviors by itself and drive the robot to complete the task reliably and safely. The effectiveness of the proposed solutions are demonstrated with various applications where the dominant uncertainty comes from either the robotic system or external environment. In the following, we briefly summarize the content and contributions of each chapter.

[12]We assume additional structures on the system and task models to make the problem more accessible. As a result, the proposed solutions work for subcategories of the problem formulated in Sec. 1.2.

Chapter 2

In this chapter, we review the existing literature that are relevant to the proposed solutions or considered applications.

Chapter 3

We propose a gradient-based optimization solution to the problem in Sec. 1.2 based on iterative Linear Quadratic Gaussian (iLQG) (van den Berg et al., 2012b). Compared to the limitations of the previous works, the proposed solution is not only able to work with nondifferentiable system models, but sensors with sparse informative measurements due to limited sensing ranges. Such features enable practical applications of the proposed solution with commonly used sensors, like Lidars or cameras.

Chapter 4

We propose a sampled-based general POMDP solver for continuous systems to solve the problem in Sec. 1.2. A few limitations of existing online general POMDP solvers are addressed. First, the proposed solution is able to work with continuous measurement spaces. Second, with more accurate belief approximation at traversing the tree, we fix the issue of over optimistic reward-to-go estimation from a rollout policy. Through theoretical analysis, we show the proposed continuous POMDP solver could produce unbiased policy value estimation with sufficient samples and simulation episodes, which makes the solution a valid Monte Carlo Tree Search (MCTS) algorithm, or a valid Monte Carlo control algorithm in general.

Chapter 5

In this chapter, we consider an application where the dominant uncertainties come from the robot itself, while assuming a static and known environment. More specifically, we apply the proposed algorithms in previous chapters to navigate a car-like robot with range sensors in a known environment. We compare the performance of the proposed solutions with

other applicable methods. The methods are compared not only in terms of the objective functions but also other metrics relevant to robotic applications. By demonstrating the superior performance of the proposed methods with such a familiar application to the robotics community, we believe the importance of stochastic motion planning could be better appreciated.

Chapter 6

In this chapter, we consider a more challenging autonomous driving application. Compared to Ch. 5, the dominant uncertainty in autonomous driving comes from external environment, *i.e.* the unknown behavior of other players in the traffic. In this work, we first propose a data-driven model for stochastic traffic dynamics sharing the motion model structure in (1.1). Then, we apply the proposed solution in Ch. 4 to plan motions for the ego vehicle within the stochastic traffic. Through comparisons in both handcrafted and real traffic scenarios, we show that the proposed solution achieves better performance compared to other approaches that also consider traffic stochasticity. To our best knowledge, this is the first work that applies a full POMDP solution to autonomous driving in general scenarios. Meanwhile, this might also be the first work that tightly couples planning and predictions with a data-driven traffic model for the autonomous driving problem.

Chapter 7

Ch. 7 concludes the dissertation and provides prospects for future research directions.

Chapter 2

Background Literature

In this chapter, we review background literature that is relevant to the stochastic motion planning problem considered in this dissertation. The problem is often known as belief space planning in the robotics community, of which we review the existing works in Sec. 2.1. There is also a strong body of literature in the AI community addressing the general POMDP problems. Related works are reviewed in Sec. 2.2. In Ch. 6 of this dissertation, we consider the stochastic motion planning problem in the context of autonomous driving. The application requires a stochastic model for the traffic dynamics, and, meanwhile, a motion planning algorithm considering uncertainties originated from not only traffic but sensor data. Related works on both topics are reviewed in Sec. 2.3 and 2.4 respectively.

2.1 Belief Space Planning

Algorithms for belief space planning are often related to Linear systems with Quadratic cost and Gaussian noise (LQG) for practical considerations. Most works on LQG systems (Prentice and Roy, 2009; Platt et al., 2010; Kopitkov and Indelman, 2019) assume maximum likelihood measurements, which help in identifying a unique posterior distribution, but introduce additional information. Du Toit and Burdick (2012) later prove that maximum likelihood measurement introduces the least amount of information comparing with other future measurement sequences. van den Berg et al. (2011) first figure out the

actual dynamics of the belief state of an LQG system. He shows that the covariance of the posterior Gaussian distribution is fixed for arbitrary future measurements, while the mean follows a Gaussian distribution. His finding motivates later works that remove the maximum likelihood assumption (Vitus and Tomlin, 2011b; van den Berg et al., 2012b; Sun et al., 2016; Rafieisakhaei et al., 2017).

The LQG-related approaches can also be categorized based on its origin of the deterministic counterpart. Censi et al. (2008) solve the problem with graph search. Bry and Roy (2011) propose RRBT by extending RRT*. More works are based on PRM since the belief roadmap proposed by Prentice and Roy (2009). Agha-mohammadi et al. (2014, 2018) introduces "belief stablizers", which assume the existence of a local controller at each node in PRM that can bring the belief at a node to its neighbor nodes representing other belief. A known problem of graph-based solutions is the lack of strict ordering of covariance matrices. Censi et al. (2008) and Bry and Roy (2011) address this problem by imposing partial ordering for beliefs. In contrast, Shan and Englot (2017) avoid the necessity of partial ordering by simply considering the largest eigenvalue of a covariance matrix.

In addition to originating from graph search methods, there are algorithms trying to find a control policy through optimization inheriting the idea of Linear Quadratic Regulator (LQR). van den Berg et al. (2012b) and Sun et al. (2016) directly solve for the feedback control policy through dynamic programming. Vitus and Tomlin (2011b), Indelman et al. (2015), and Rafieisakhaei et al. (2017), on the other hand, find the next best action and form a feedback policy through RHC.

2.2 General Solutions for Partially Observable Markov Decision Process (POMDP)

In the AI community, there is also a strong body of literature on POMDP solvers for general systems, where only generative models, instead of explicit model formulations, are required.

For finite discrete systems, Pineau et al. (2003) first propose offline point-based method, which forms the basis for many later extensions (Shani et al., 2013). There are two main ideas embedded in the point-based methods which break the *curse of dimensionality* and the *curse of history* respectively. In the case that an initial belief is known, it is believed that only a small subset of the belief space is reachable. Therefore, one can only consider the reachable belief space, which can be approximated by a set of belief samples. The second idea is to exploit the fact that the optimal value function for a finite discrete system is a convex piecewise linear function of the belief, first proved by Sondik (1971). Although the number of piecewise linear regions grows with search depth, one can only keep a fixed number of "α-vectors" in order to bound the computation complexity. Extensions of point-based methods (Smith and Simmons, 2004; Hanna Kurniawati, 2008) are usually different in how to choose the belief samples and update the "α-vectors".

Offline POMDP solvers often estimate the value function over the entire belief space. In contrast, online POMDP solvers focus on estimating the value for the current belief and generating the next best action. Most of the online POMDP solvers are based on tree search (Ross et al., 2008), alternating three steps: find the most promising leaf node; expand the node; update the parents of the leaf node through dynamic programming. Variations of online POMDP solvers (Paquet et al., 2005; Ross and Chaib-Draa, 2007) are usually different in what heuristics is applied in finding the next best leaf node to expand. Developments in sampling based online POMDP solvers, such as POMCP (Silver and Veness, 2010) and DESPOT (Ye et al., 2017), have demonstrated their effectiveness and efficiency in solving large scale problems (PacMan with 10^{56} states).

It is worth noting the subtle difference between POMCP and tree search methods. POMCP is based on Monte Carlo Tree Search, where the tree and the predefined rollout policy, together, construct a single policy. As the number of simulations increases, the policy represented by the tree may hopefully converge to the optimal policy. In contrast, tree search methods, such as DESPOT, embed all policies in the tree. The job of the

algorithm is to identify the optimal policy by accurately estimating the values at the nodes in the tree.

Most of the general POMDP solvers, either offline or online, are limited to finite discrete measurement space. A few works attempt to relax this limitation (Porta et al., 2006). Hoey and Poupart (2005) propose the insight that observations should only be differentiated when they result in different posterior distributions, which provides a metric to partition large, or even continuous, observation space. van den Berg et al. (2012a), Chaudhari et al. (2013), and Sunberg and Kochenderfer (2018) try to solve the continuous POMDPs in general. van den Berg et al. (2012a) generalize iLQG (van den Berg et al., 2012b), focusing on continuous POMDPs with Gaussian noise. Chaudhari et al. (2013) propose a sequential approach to discretize a continuous POMDP through incremental sampling. Sunberg and Kochenderfer (2018) extend POMCP and utilize progressive widening (Couëtoux et al., 2011) to handle both continuous control and observation spaces.

A few works try to apply general POMDP solvers for stochastic motion planning problems. Kurniawati et al. (2011) propose milestone guided sampling, which is, at its root, a point-based method and is still limited to discrete problems. The specialty of milestone guided sampling is that it employs a similar idea with PRM to guide the sampling of belief space. Lauri and Ritala (2016) address the navigation of a ground vehicle with range sensors using POMCP. The continuous observation space is discretized by converting range sensor measurements to tuples consisting of cell indices and corresponding occupancy status. However, the number of possible discrete measurements is still huge, which could results in an overly shallow tree and, therefore, barely useful for long term planning.

2.3 Traffic Dynamics Modeling

A critical component of autonomous driving, before considering motion planning, is modeling the traffic dynamics. Simple parametric models, such as IDM (Treiber and Kesting, 2013) and MOBIL (Kesting et al., 2007), may serve this purpose, which has been used

in our previous work (Sun et al., 2020a) and more (Sunberg et al., 2017; Evestedt et al., 2016; Zhang et al., 2020). However, such models suffer from two major limitations. First, the models are not able to describe the detailed behavior of agent vehicles, like a trajectory of lane change. Second, the models are designed for study the macro traffic behavior. Predictions of the models may not agree with the local behavior of agent vehicles. Such limitations of hardcrafted parametric models motivate data-driven approaches in predicting agent behavior, harnessing the power of deep neural networks. Thanks to the recently available datasets, such as ArgoVerse (Chang et al., 2019), nuScenes (Caesar et al., 2020), Lyft5 (Houston et al., 2020), and Waymo Open (Sun et al., 2020b), data-driven approaches to modeling the stochastic traffic dynamics become possible.

Traffic models in related works are often non-Markovian (Chai et al., 2019; Cui et al., 2019; Phan-Minh et al., 2020; Rhinehart et al., 2019), which predict the future in the next few seconds based on the past agent trajectories of a short duration. The non-Markovian structure allows more flexibility for the network to exploit data over multiple steps and could therefore potentially improve prediction accuracy. However, it may be hard to apply efficient planning algorithms implementing concepts like dynamic programming with such models because of the lack of Markovian structure. Recurrent models (Sanchez-Gonzalez et al., 2018) are suitable for this purpose, which take the network output as the input for the next time step. However, recurrent models are known to be subject to compounding errors (Ross et al., 2011). Although not fully resolved, the issue can be alleviated with scheduled sampling as shown by Bengio et al. (2015).

Another important aspect of the traffic model is to reflect the interactions between the ego and agents, *i.e.* make predictions of the traffic conditioned on the motion of the ego vehicle. The property is investigated by Rhinehart et al. (2019), Khandelwal et al. (2020), Salzmann et al. (2020), and Tolstaya et al. (2021). Rhinehart et al. (2019) and Khandelwal et al. (2020) both infer agent motion conditioned on a local goal of the ego. Simply conditioning on the goal might be insufficient, since various ego motions could lead to the same

goal, while each of them could interact with the agents differently. Instead, Salzmann et al. (2020) and Tolstaya et al. (2021) perform inference conditioned on the future trajectory of the ego. It is straightforward to use both models for query, *i.e.* predicting agent trajectories with a complete ego trajectory. However, constructing an ego trajectory incrementally would be hard to realize with such models.

A large body of prior works (Chai et al., 2019; Cui et al., 2019; Phan-Minh et al., 2020; Messaoud et al., 2020; Gao et al., 2020; Zhao et al., 2020) focus on predicting the future of a single agent. However, a traffic scene may consist of various numbers of players, where predictions for all are required. Although the previously mentioned algorithms can be applied to all agents sequentially, they are limited in capturing the complex interactions between agents. As a result, the predictions may not be consistent, *i.e.* predictions of different agents could lead to collisions. To address the issue, more recent works (Salzmann et al., 2020; Casas et al., 2020; Casas et al., 2020; Liang et al., 2020; Zeng et al., 2021) are able to make joint predictions for all agents of interest, which either explicitly or implicitly apply the idea of a GNN (Battaglia et al., 2018).

Casas et al. (2018); Rhinehart et al. (2019); Salzmann et al. (2020) try to model the partial observability of traffic dynamics by introducing latent variables into the network. Rhinehart et al. (2019) applies latent variables per agent per step in order to randomize the predictions. Casas et al. (2018) and Salzmann et al. (2020) introduce discrete latent variables intended to affect the high-level behavior of agent predictions. In the first case (Rhinehart et al., 2019), latent variables are *i.i.d*, therefore, independent of the traffic state. In the second case (Casas et al., 2018; Salzmann et al., 2020), the latent variables are internal to the network and generated as intermediate results. In either case, it is not straightforward to include such latent variables as part of the traffic state.

2.4 Autonomous Driving in POMDP frameworks

In terms of planning motions for autonomous vehicles, most existing works focus on deterministic motion planning algorithms (Montemerlo et al., 2008; Urmson et al., 2008; McNaughton et al., 2011; Ding et al., 2019; Sun et al., 2020a; Paden et al., 2016) for efficiency. Similar to the work in this dissertation, a few consider the uncertain nature of traffic dynamics and sensor measurements, and plan motion for the autonomous vehicle in a POMDP framework. To reduce the complexity, works in the latter category often simplify the autonomous driving problem by either focusing on specific scenarios (Brechtel et al., 2014; Hubmann et al., 2018b), like lane change, or only considering an over-simplified dynamics for the ego vehicle (Hubmann et al., 2018a; Liu et al., 2015; Sunberg et al., 2017), like only planning velocity profiles. In addition, despite the minor data-driven components (Liu et al., 2015), most works assume a handcrafted traffic model. Predictions with such models may not agree with real driver behaviors. As a result, it might be hard to argue that the motion planned with the above methods should adapt to actual traffic scenarios.

Methodology-wise, Liu et al. (2015); Hubmann et al. (2018a,b, 2019) solve for the policy with discrete POMDP solvers (Ye et al., 2017; Kurniawati and Yadav, 2016). Hubmann et al. (2018a,b, 2019) attempt to avoid continuous measurement space by assuming noise-free measurements. Effectively, the ego vehicle measures the reference paths that the agents are following which only include finite options. The assumed measurement model fails to capture the intrinsic uncertainty of observations. The problem quickly degenerates to a deterministic problem once the ambiguity of reference paths is resolved. Brechtel et al. (2014) propose to apply an offline continuous POMDP method for autonomous driving. However, the solved policy only targets a specific scenario. Resolving the continuous POMDP is required if a new scenario is encountered. Sunberg et al. (2017) applies a continuous online POMDP solver, to plan motions for an autonomous vehicle in highway lane change scenarios. To simplify the problem, vehicle dynamics is modeled with a second-order integrator in Sunberg et al. (2017), where vehicle orientation is ignored.

Closest to the problem setup in this dissertation is the MultiPolicy Decision Making method (MPDM) (Galceran et al., 2017) and an improved variant of it, Efficient Uncertainty-aware Decision Making (EUDM) (Zhang et al., 2020). Both methods consider uncertainty resulting from traffic dynamics and sensor observations, and can be applied in general scenarios. Compared to MPDM, a major difference of EUDM is being able to switch policies for the ego within the planning horizon. Generally speaking, MPDM is a sampling-based POMDP method. To simplify the solution, MPDM ignores the change of belief caused by different possible future measurements. Instead, within each planning horizon, it considers the belief to be fixed at the current estimation provided by the external state estimator.

Chapter 3

Belief Space Planning with Iterative Linear Quadratic Gaussian (iLQG)

As reviewed in Sec. 2.1, compared to graph-based solutions, iLQG (van den Berg et al., 2012b) could be more efficient in solving the stochastic motion planning problem since it exploits the gradient of the system and task models when available. However, few success has been reported in applying iLQG in practice to plan motions for real robots equipped with commonly used sensors like Lidars or cameras. The rare application is caused by two limitations of iLQG. First, iLQG requires differentiable system models. Although the motion models of many robots are differentiable, it is not the case for measurement models for commonly used sensors. For example, measurement models for Lidars are not differentiable due to the discontinuity in the environment. Second, informative measurements, *i.e.* measurements that are effective in reducing uncertainty, are sparse due to the limited sensing range. In this chapter, we extend iLQG and address both of the above mentioned limitations.

Related to the improvements introduced in this work, van den Berg et al. (2012b) briefly mention that Unscented Kalman Filters (UKF) could replace Extended Kalman Filters (EKF) to approximate belief dynamics. Nishimura and Schwager (2018) also applies

a UKF as belief dynamics approximation in an active multi-target tracking problem with (unlimited) range measurements. In both works, the application of UKFs are not intended for nondifferentiable system models. Bachrach et al. (2012) share the same spirit with this work, where a UKF is applied in the belief roadmap framework (Prentice and Roy, 2009) for nondifferentiable measurement models of RGB-D cameras. One technical flaw cannot be avoided. With the application of UKF in BRM, the computation of "transfer functions" of covariance requires the prior uncertainty, which is unavailable while constructing the roadmap.

A few works consider randomness in measurement acquisition, which is similar to the sparse informative measurements problem. Patil et al. (2014) introduce signed distance field to model sensing regions. The use of signed distance field implicitly assumes the direction of movement for obtaining measurements is unique and known *a priori*, which is often not the case. Indelman et al. (2015) and Chaves et al. (2015) introduce Bernoulli random variables to model the randomness in measurement acquisition. In the work of Indelman et al. (2015), the binary random variable is computed with the expected state and is fixed during the optimization. Chaves et al. (2015), on the other hand, assume the random variable is independent of the state. The issue of sparse informative measurements cannot be addressed by either method because of the assumed independence between the state and measurement acquisition during the optimization.

Contributions

In this work, we address the limitations of iLQG when working with commonly used sensors like Lidars or cameras so that the proposed solution can be applied in practice to solve the stochastic motion planning problem.

First, the application of iLQG only requires the belief dynamics to be differentiable. We show that the differentiability of the true discrete-time belief dynamics, modeled by a Bayes filter, is independent of the underlying motion or measurement models. The explicit

requirement for differentiability (van den Berg et al., 2012b) is because of the use of an EKF in approximating the belief dynamics. To overcome this issue, we propose to use a derivative-free filter based on a UKF for belief dynamics modeling.

Second, the discontinuity of the noise standard deviation of a sensor model can be understood as the cause of sparse informative measurements. The noise standard deviation is infinite when the measurement is outside the sensing range. We propose to approximate the scaling function of measurement noise with a sigmoid function, which is equivalent to introducing artificial gradient into the optimization, therefore, densifying the informative measurements. By properly scheduling the sigmoid parameters in an outer loop of iLQG, the modified measurement model eventually converges to the true model, ensuring that the resulting control policy is designed for the original system.

In the following of this chapter, we first present the problem formulation in Sec. 3.1. In Sec. 3.2, we provide a quick overview of iLQG proposed by van den Berg et al. (2012b). In Sec. 3.3 and 3.4, we address the two previously mentioned limitations of iLQG, namely nondifferentiable system models and sparse informative measurements. The work in this chapter is published in Sun and Kumar (2021).

3.1 Problem Formulation

Consider a system modeled as follows,

$$\begin{aligned} \boldsymbol{x}_{t+1} &= f(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{m}_t), \quad \boldsymbol{m}_t \sim \mathcal{N}(\boldsymbol{0}, \Sigma_{\boldsymbol{m}}), \\ \boldsymbol{z}_t &= h(\boldsymbol{x}_t, \boldsymbol{n}_t), \qquad\quad \boldsymbol{n}_t \sim \mathcal{N}(\boldsymbol{0}, \Sigma_{\boldsymbol{n}}). \end{aligned} \tag{3.1}$$

In (3.1), $f(\cdot) : \mathcal{X} \times \mathcal{U} \times \mathbb{R}^m \mapsto \mathcal{X}$ and $h(\cdot) : \mathcal{X} \times \mathbb{R}^n \mapsto \mathcal{Z}$ are the motion and measurement models. $\boldsymbol{x} \in \mathcal{X}$, $\boldsymbol{u} \in \mathcal{U}$, $\boldsymbol{z} \in \mathcal{Z}$ are the state, control, and measurement. $\boldsymbol{m} \in \mathbb{R}^m$ and $\boldsymbol{n} \in \mathbb{R}^n$ are the Gaussian motion and measurement noises with zero mean and covariance $\Sigma_{\boldsymbol{m}}$ and $\Sigma_{\boldsymbol{n}}$.

Assuming one does not have access to the perfect state information, but is given the initial belief $\boldsymbol{b}_0 \in \mathcal{B}$, with which we could model the task as the following objective function,

$$V_0(\boldsymbol{b}_0) = \mathbb{E}\left\{\sum_{t=0}^{N-1} c_t(\boldsymbol{b}_t, \boldsymbol{u}_t) + c_N(\boldsymbol{b}_N)\right\}. \tag{3.2}$$

In (3.2), $c_t(\cdot) : \mathcal{B} \times \mathcal{U} \mapsto \mathbb{R}$ and $c_N(\cdot) : \mathcal{B} \mapsto \mathbb{R}$ are the stage and terminal costs respectively[1]. In the rest of this chapter, we assume that both $c_t(\cdot)$ and $c_N(\cdot)$ are second-order differentiable with positive-(semi)definite Hessians.

The problem is to find a control policy, $\pi_t(\cdot) = \{\mu_0, \mu_1, \ldots, \mu_{N-1}\}$ with $\mu_t(\cdot) : \mathcal{B} \mapsto \mathcal{U}$, that drives the robot to complete a task, which could be obtained by minimizing the objective function in (3.2),

$$\pi_t^* = \arg\min_{\pi_t} V_0(\boldsymbol{b}_0) = \arg\min_{\pi_t} \mathbb{E}\left\{\sum_{t=0}^{l-1} c_t(\boldsymbol{b}_t, \mu_t(\boldsymbol{b}_t)) + c_N(\boldsymbol{b}_N)\right\}. \tag{3.3}$$

Compared the general problem formulation in Sec. 1.2, the problem formulated in this chapter assumes more structure on the system and task models. To summarize, (3.1) implicitly assumes known system formulations (generative models are insufficient) with Gaussian noises. Meanwhile, the stage and terminal cost functions should be second-order differentiable with positive-(semi)definite Hessians.

3.2 Preliminaries on iLQG

Assuming both the motion, $f(\cdot)$, and measurement, $h(\cdot)$, models in (3.1) are differentiable, an EKF can be applied to approximate the belief dynamics.

In EKFs, belief is approximated with Gaussian distributions, $\mathcal{N}(\hat{\boldsymbol{x}}, \Sigma)$, where $\hat{\boldsymbol{x}}$ is the estimated mean of the state, while Σ is the covariance. With the latest control and

[1] Cost, instead of reward, is used in (3.2) to align with the existing literature in belief space planning. It should be further noted that we assume costs are functions of belief directly instead of states as in (1.2). Effectively, we are directly modeling the expected stage cost, *i.e.* $c_t(\boldsymbol{b}_t, \boldsymbol{u}_t) = \mathbb{E}\{c'_t(\boldsymbol{x}_t, \boldsymbol{u}_t)\}$.

measurement, the belief is updated as,

$$\begin{aligned} \hat{\boldsymbol{x}}_{t+1} &= f(\hat{\boldsymbol{x}}_t, \boldsymbol{u}_t, \mathbf{0}) + K_{t+1}(\boldsymbol{z}_{t+1} - h(f(\hat{\boldsymbol{x}}_t, \boldsymbol{u}_t, \mathbf{0}), \mathbf{0})), \\ \sqrt{\Sigma_{t+1}} &= \sqrt{\Gamma_t - K_{t+1} H_{t+1} \Gamma_t}. \end{aligned} \tag{3.4}$$

where,

$$\begin{aligned} \Gamma_t &= A_t \Sigma_t A_t^\top + M_t M_t^\top, \\ K_{t+1} &= \Gamma_t H_{t+1}^\top (H_{t+1} \Gamma_t H_{t+1}^\top + N_{t+1} N_{t+1}^\top)^{-1}, \\ A_t &= \frac{\partial f}{\partial \boldsymbol{x}}(\hat{\boldsymbol{x}}_t, \boldsymbol{u}_t, \mathbf{0}), \\ M_t &= \frac{\partial f}{\partial \boldsymbol{m}}(\hat{\boldsymbol{x}}_t, \boldsymbol{u}_t, \mathbf{0}), \\ H_{t+1} &= \frac{\partial h}{\partial \boldsymbol{x}}(f(\hat{\boldsymbol{x}}_t, \boldsymbol{u}_t, \mathbf{0}), \mathbf{0}), \\ N_{t+1} &= \frac{\partial h}{\partial \boldsymbol{n}}(f(\hat{\boldsymbol{x}}_t, \boldsymbol{u}_t, \mathbf{0}), \mathbf{0}). \end{aligned} \tag{3.5}$$

Note, in (3.4), the square root of Σ, $\sqrt{\Sigma}$ (the Cholesky factorization of Σ), is used for numerical stability. We may also take advantage of the sparsity of $\sqrt{\Sigma}$, and represent the belief in a vector form, *i.e.* $\boldsymbol{b} = \left(\hat{\boldsymbol{x}}^\top, \text{vech}(\sqrt{\Sigma})^\top\right)^\top$, where $\text{vech}(\sqrt{\Sigma})$ stacks the lower (or equivalently upper) triangular entries of $\sqrt{\Sigma}$ as a vector. With the new belief representation, the belief dynamics in (3.4) can be rewritten as,

$$\boldsymbol{b}_{t+1} = \begin{pmatrix} f(\hat{\boldsymbol{x}}_t, \boldsymbol{u_t}, \mathbf{0}) + \sqrt{K_{t+1} H_{t+1} \Gamma_t} \boldsymbol{w}_{t+1} \\ \text{vech}\big(\sqrt{\Gamma_t - K_{t+1} H_{t+1} \Gamma_t}\big) \end{pmatrix}, \tag{3.6}$$

where $\boldsymbol{w}_{t+1} \sim \mathcal{N}(\mathbf{0}, I)$ is the normalized innovation.

The key observation by van den Berg et al. (2012b) is that the belief dynamics approximated with an EKF has the form,

$$\boldsymbol{b}_{t+1} = \Phi(\boldsymbol{b}_t, \boldsymbol{u}_t, \boldsymbol{z}_{t+1}), \tag{3.7}$$

sharing the same form of a motion model. Specially, $\boldsymbol{b}$ is the (belief) state, $\boldsymbol{u}$ is the control

Algorithm 1: The iLQG Algorithm (van den Berg et al. (2012b))

Input: initial belief trajectory $\{\bar{\boldsymbol{b}}_0, \bar{\boldsymbol{b}}_1, \ldots, \bar{\boldsymbol{b}}_N\}$,
initial policy $\{\mu_0, \mu_1, \ldots, \mu_{N-1}\}$.
Output: local optimal policy $\{\mu_0^*, \mu_1^*, \ldots, \mu_{N-1}^*\}$.
1 Initialize $\{\mu_0^*, \ldots, \mu_{N-1}^*\}$ with $\{\mu_0, \ldots, \mu_{N-1}\}$.
2 **while** *policy has not converged* **do**
3 Generate the nominal trajectory $\{\bar{\boldsymbol{b}}_0, \bar{\boldsymbol{u}}_0, \ldots, \bar{\boldsymbol{b}}_N\}$ with $\bar{\boldsymbol{u}}_t = \mu_t^*(\bar{\boldsymbol{b}}_t)$.
4 Approximate the final step optimal value function V_N^* with a second-order Taylor expansion of c_N in (3.2) around $\bar{\boldsymbol{b}}_N$.
5 **for** *t from* $N-1$ *to* 0 **do**
6 Approximating Φ in (3.7) with a first-order Taylor expansion around $\bar{\boldsymbol{b}}_t$ and $\bar{\boldsymbol{u}}_t$ to produce a linear belief dynamics Φ'.
7 Approximating c_t in (3.2) with a second-order Taylor expansion around $\bar{\boldsymbol{b}}_t$ and $\bar{\boldsymbol{u}}_t$ to produce a quadratic stage cost c_t'.
8 Solve for μ_t^* and V_t^* through minimizing $c_t'(\boldsymbol{b}_t, \boldsymbol{u}_t) + \mathbb{E}\{V_{t+1}^*(\Phi'(\boldsymbol{b}_t, \boldsymbol{u}_t, \boldsymbol{z}_{t+1}))\}$.
9 **end**
10 **end**

input, and $\boldsymbol{z}$, or equivalently innovation, plays the role of motion noise. Therefore, iterative Linear Quadratic Regulator (iLQR) (Li and Todorov (2004)) can be applied on (3.7) to obtain a local optimal feedback policy of belief.

The general framework of iLQG is summarized in Alg. 1. Given an initial control policy, the original nonlinear optimization problem is converted to an LQG by approximating the belief dynamics (Line 6) and the cost functions (Line 4 and 7) through first and second order Taylor expansions respectively. The control policy for the LQG can be solved recursively through dynamic programming (Line 8). With the new control policy, the nominal trajectory is updated (Line 3), which is then used to update the approximations for the belief dynamics and costs for the next iteration. Note steps like line search and trust region methods are omitted for brevity, which are necessary in practice to ensure convergence.

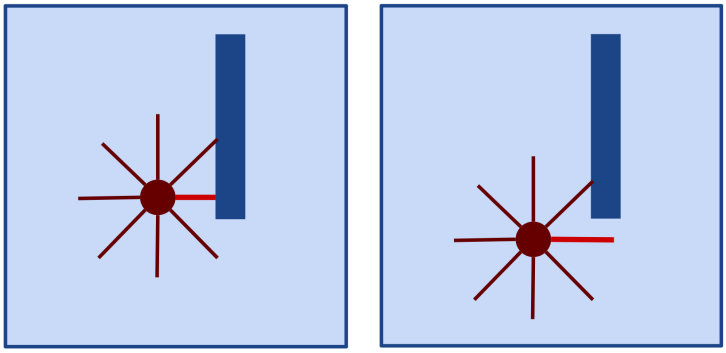


Figure 3.1: Nondifferentiable range sensor measurements. If the position of the robot (dark red circle) is perturbed downward, the light red beam no longer hits the obstacle (dark blue rectangle). As a result, the range measured by the light red beam can produce a significant change.

3.3 Nondifferentiable System Models

One major limitation of iLQG that prevents its wide application comes from the requirement of differentiable motion and measurement models of the system. However, sensors that are often used in practice, such as Lidars and cameras, have nondifferentiable measurement models. Here, by "nondifferentiable measurement models", we specifically mean that the measurement model, $h(\cdot)$ in (3.1), is not differentiable *w.r.t.* the state, *i.e.* $\frac{\partial h}{\partial \boldsymbol{x}}$ does not exist for all $\boldsymbol{x} \in \mathcal{X}$. Figure 3.1 illustrates this limitation with an example of a range sensor.

In the rest of this section, we introduce modifications of iLQG in order to work with systems with nondifferentiable models.

3.3.1 Differentiability of the Belief Dynamics

The key insight is that the application of iLQG only requires differentiability of the belief dynamics in (3.7). The requirement of explicit differentiability of the motion and measurement models in the work of van den Berg et al. (2012b) actually comes from the usage of an EKF to approximate belief dynamics. In the following, we show that the true belief dynamics is differentiable by itself with mild assumptions.

The true discrete-time belief dynamics is modeled by the Bayes filter in the following

form,

$$\begin{aligned} \boldsymbol{p}_{t+1}(\boldsymbol{x}) &= \int t(\boldsymbol{x}|\boldsymbol{y}, \boldsymbol{u}_t)\boldsymbol{b}_t(\boldsymbol{y})d\boldsymbol{y}, \\ \boldsymbol{q}_{t+1}(\boldsymbol{x}) &= l(\boldsymbol{z}_{t+1}|\boldsymbol{x})\boldsymbol{p}_{t+1}(\boldsymbol{x}), \\ \boldsymbol{b}_{t+1}(\boldsymbol{x}) &= \frac{1}{\int \boldsymbol{q}_{t+1}(\boldsymbol{y})d\boldsymbol{y}}\boldsymbol{q}_{t+1}(\boldsymbol{x}). \end{aligned} \tag{3.8}$$

In (3.8), $t(\cdot) : \mathcal{X}^2 \times \mathcal{U} \mapsto \mathbb{R}$ is the transition probability. $l(\cdot) : \mathcal{X} \times \mathcal{Z} \mapsto \mathbb{R}$ is the measurement likelihood. $\boldsymbol{p}_{t+1}$ is the prior *p.d.f.*. $\boldsymbol{q}_{t+1}$ is the unnormalized posterior *p.d.f.*, normalizing which gives the belief at $t+1$, $\boldsymbol{b}_{t+1}$.

Assumption 3.1. *We assume the following properties on the transition probability, measurement likelihood, and initial belief.*

(1) $t(\cdot)$ *is continuous on* $\mathcal{X}^2 \times \mathcal{U}$*, and* $0 < t(\cdot) < \infty$*.*

(2) $\partial t/\partial \boldsymbol{u}_t$ *exists and is continuous on* $\mathcal{X}^2 \times \mathcal{U}$*.*

(3) $l(\cdot)$ *is continuous on* $\mathcal{X} \times \mathcal{Z}$*, and* $0 < l(\cdot) < \infty$*.*

(4) $\partial l/\partial \boldsymbol{z}_{t+1}$ *exists.*

(5) $\boldsymbol{b}_t$*, as a p.d.f., is continuous on* $\mathcal{X}$*.*

We note that the above properties are mild assumptions of a system. The first four items are trivially satisfied by assuming both the motion and measurement noises are of Gaussian distributions, which align with the system models in (3.1). Meanwhile, we show in the following that the continuity of $\boldsymbol{b}_t$ can be preserved by (3.8). Therefore, it is sufficient to assume the continuity of $\boldsymbol{b}_0$ in order to ensure the continuity of $\boldsymbol{b}_t$'s for future steps.

Theorem 3.1 (Preservation of Belief Continuity)**.** *With Assumption 3.1, continuity of belief is preserved by the Bayes filter in* (3.8)*, i.e.* $\boldsymbol{b}_{t+1}$ *is continuous on* $\mathcal{X}$ *given* $\boldsymbol{b}_t$ *is so.*

Proof. Since $t(\cdot)$, $l(\cdot)$, and $\boldsymbol{b}_t$ are continuous functions, $\boldsymbol{p}_{t+1}$ and $\boldsymbol{q}_{t+1}$ are continuous. Given that $0 < t(\cdot),\ l(\cdot) < \infty$ and $\int \boldsymbol{b}_t(\boldsymbol{y})d\boldsymbol{y} = 1$, we have $0 < \int \boldsymbol{q}_{t+1}(\boldsymbol{y})d\boldsymbol{y} < \infty$. Therefore, $\boldsymbol{b}_{t+1}$

is continuous. □

Next we show the differentiability of $\boldsymbol{b}_{t+1}$ *w.r.t.* $\boldsymbol{u}_t$, $\boldsymbol{z}_t$, and $\boldsymbol{b}_t$. Note here we consider $\boldsymbol{b}_t$'s as probability density functions. Therefore, it does not make sense to discuss the differentiability of a function *w.r.t.* vectors or other functions directly. Instead, we consider the pointwise differentiability, *i.e.*, the relationships between $d\boldsymbol{b}_{t+1}(\boldsymbol{x})$ and $d\boldsymbol{u}_t$, $d\boldsymbol{z}_{t+1}$, and $d\boldsymbol{b}_t$. Specially, $\boldsymbol{b}_{t+1}(\boldsymbol{x})$ should be considered as a functional of $\boldsymbol{b}_t$.

Theorem 3.2 (Differentiability of the Discrete-time Belief Dynamics)**.** *With Assumption 3.1,* $\boldsymbol{b}_{t+1}$ *is pointwise differentiable w.r.t.* $\boldsymbol{u}_t$*,* $\boldsymbol{z}_t$*, and* $\boldsymbol{b}_t$*.*

Proof. The differentiability of $\boldsymbol{b}_{t+1}(\boldsymbol{x})$ could be shown by construction.

$$
\begin{aligned}
d\boldsymbol{p}_{t+1}(\boldsymbol{x}) &= \int \frac{\partial t}{\partial \boldsymbol{u}_t}(\boldsymbol{x}|\boldsymbol{y}, \boldsymbol{u}_t) d\boldsymbol{u}_t \boldsymbol{b}_t(\boldsymbol{y}) d\boldsymbol{y} + \int t(\boldsymbol{x}|\boldsymbol{y}, \boldsymbol{u}_t) d\boldsymbol{b}_t(\boldsymbol{y}) d\boldsymbol{y}, \\
d\boldsymbol{q}_{t+1}(\boldsymbol{x}) &= \frac{\partial l}{\partial \boldsymbol{z}_{t+1}}(\boldsymbol{z}_{t+1}|\boldsymbol{x}) d\boldsymbol{z}_{t+1} \boldsymbol{p}_{t+1}(\boldsymbol{x}) + l(\boldsymbol{z}_{t+1}|\boldsymbol{x}) d\boldsymbol{p}_{t+1}(\boldsymbol{x}), \\
d\boldsymbol{b}_{t+1}(\boldsymbol{x}) &= \frac{d\boldsymbol{q}_{t+1}(\boldsymbol{x})}{\int \boldsymbol{q}_{t+1}(\boldsymbol{y}) d\boldsymbol{y}} - \frac{\boldsymbol{q}_{t+1}(\boldsymbol{x}) \cdot \int d\boldsymbol{q}_{t+1}(\boldsymbol{y}) d\boldsymbol{y}}{\left(\int \boldsymbol{q}_{t+1}(\boldsymbol{y}) d\boldsymbol{y}\right)^2}.
\end{aligned}
\tag{3.9}
$$

The differential equations hold because of Assumption 3.1, which allows the application of Leibniz's rule to switch the order of differentiation and integration. Applying the chain rule to (3.9) produces the required partial derivatives, $\partial \boldsymbol{b}_{t+1}(\boldsymbol{x})/\partial \boldsymbol{b}_t(\boldsymbol{y})$, $\partial \boldsymbol{b}_{t+1}(\boldsymbol{x})/\partial \boldsymbol{u}_t$, and $\partial \boldsymbol{b}_{t+1}(\boldsymbol{x})/\partial \boldsymbol{z}_{t+1}$. □

Therefore, the differentiability of belief dynamics is independent of differentiability of the system motion, $f(\cdot)$, and measurement, $h(\cdot)$, models in (3.1).

In Gaussian filters, such as KFs and EKFs, (multivariate) Gaussian distributions are used to approximate the underlying belief with the first and second moments. In the following, we show the differentiability of the first two moments of $\boldsymbol{b}_{t+1}$ by construction, assuming $\boldsymbol{b}_t$ is Gaussian.

Corollary 3.2.1 (Differentiability of the First Two Moments)**.** *The first two moments of*

$\boldsymbol{b}_{t+1}$ are differentiable w.r.t. $\boldsymbol{u}_t$ and $\boldsymbol{z}_t$, and the first two moments of $\boldsymbol{b}_t$ assuming $\boldsymbol{b}_t$ is Gaussian.

Proof. Consider the mean, $\mathbb{E}\{\boldsymbol{x}_{t+1}\}$ of $\boldsymbol{b}_{t+1}$. Differentiating $\mathbb{E}\{\boldsymbol{x}_{t+1}\}$ *w.r.t.* $\boldsymbol{b}_{t+1}$ while applying Theorem 3.2 gives,

$$\begin{aligned} d\mathbb{E}\{\boldsymbol{x}_{t+1}\} &= \int \boldsymbol{x} \cdot d\boldsymbol{b}_{t+1}(\boldsymbol{x}) d\boldsymbol{x}, \\ d\boldsymbol{b}_{t+1}(\boldsymbol{x}) &= \frac{\partial \boldsymbol{b}_{t+1}(\boldsymbol{x})}{\partial \boldsymbol{u}_t} d\boldsymbol{u}_t + \frac{\partial \boldsymbol{b}_{t+1}(\boldsymbol{x})}{\partial \boldsymbol{z}_{t+1}} d\boldsymbol{z}_{t+1} + \int \frac{\partial \boldsymbol{b}_{t+1}(\boldsymbol{x})}{\partial \boldsymbol{b}_t(\boldsymbol{y})} d\boldsymbol{b}_t(\boldsymbol{y}) d\boldsymbol{y}. \end{aligned} \tag{3.10}$$

Recall $\boldsymbol{b}_t$ is an (multivariate) Gaussian parameterized by $\mathbb{E}\{\boldsymbol{x}_t\}$ and $\mathbb{E}\{\boldsymbol{x}\boldsymbol{x}^\top\}$. Therefore,

$$d\boldsymbol{b}_t(\boldsymbol{x}) = \frac{\partial \boldsymbol{b}_t(\boldsymbol{x})}{\partial \mathbb{E}\{\boldsymbol{x}_t\}} d\mathbb{E}\{\boldsymbol{x}_t\} + \frac{\partial \boldsymbol{b}_t(\boldsymbol{x})}{\partial \mathbb{E}\{\boldsymbol{x}_t \boldsymbol{x}_t^\top\}} d\mathbb{E}\{\boldsymbol{x}_t \boldsymbol{x}_t^\top\}. \tag{3.11}$$

Details of $\partial \boldsymbol{b}_t(\boldsymbol{x}) / \partial \mathbb{E}\{\boldsymbol{x}_t\}$ and $\partial \boldsymbol{b}_t(\boldsymbol{x}) / \partial \mathbb{E}\{\boldsymbol{x}_t \boldsymbol{x}_t^\top\}$ are provided by Petersen and Pedersen (2012). Combining (3.10) and (3.11) produces $\partial \mathbb{E}\{\boldsymbol{x}_{t+1}\} / \partial \mathbb{E}\{\boldsymbol{x}_t\}$ and $\partial \mathbb{E}\{\boldsymbol{x}_{t+1}\} / \partial \mathbb{E}\{\boldsymbol{x}_t \boldsymbol{x}_t^\top\}$. Similar steps can be applied for $\partial \mathbb{E}\{\boldsymbol{x}_{t+1} \boldsymbol{x}_{t+1}^\top\} / \partial \mathbb{E}\{\boldsymbol{x}_t\}$ and $\partial \mathbb{E}\{\boldsymbol{x}_{t+1} \boldsymbol{x}_{t+1}^\top\} / \partial \mathbb{E}\{\boldsymbol{x}_t \boldsymbol{x}_t^\top\}$, which completes the proof. □

As a result of Corollary 3.2.1, if $\boldsymbol{b}_{t+1}$ is approximated as a Gaussian distribution using the first two moments, the parametric representation of the belief is differentiable.

3.3.2 Approximate Belief Dynamics with Unscented Kalman Filters

The results of the previous section confirm that the successful application of iLQG as reported in van den Berg et al. (2012b) does not rely on the EKF belief dynamics approximation, but the nice properties of the underlying true belief dynamics. Furthermore, the differentiability of the discrete-time belief dynamics, modeled as a Bayes filter, does not depend on the differentiability of the actual motion and measurement models in (3.1). Therefore, in theory, we could apply iLQR directly on the Bayes filter to generate control policies for systems with nondifferentiable models, although it could be computationally

intractable in practice.

The insight inspires us to approximate the belief dynamics with a derivative-free filter. In this work, we use a UKF to avoid explicitly differentiating system models. In the following, we provide the details of using a UKF for belief dynamics approximation. Specially, we consider an on-manifold UKF (Brossard et al., 2017) instead of a plain UKF (Thrun et al., 2005, Ch.3), since in robotic applications, the state spaces are often Lie groups, such as $SE(2)$ and $SE(3)$.

In an on-manifold UKF, belief, $\boldsymbol{b} = \mathcal{N}(\boldsymbol{x}, \Sigma)$, is approximated with $2n+1$ sigma points with n for the state dimension.

$$\begin{aligned} \boldsymbol{s}^i &= \boldsymbol{x} \circ \exp\Big(\sqrt{(n+\lambda)\Sigma}^{(i)}\Big), \\ \boldsymbol{s}^{i+n} &= \boldsymbol{x} \circ \exp\Big(-\sqrt{(n+\lambda)\Sigma}^{(i)}\Big), \\ \boldsymbol{s}^0 &= \boldsymbol{x},\ i = 1, 2, \ldots, n. \end{aligned} \tag{3.12}$$

In (3.12), $\lambda = \alpha^2(n+\kappa) - n$, where α and κ control the spread of the sigma points. We use $\circ$ and $\exp(\cdot)$ to denote the composition and exponential operations on Lie groups. For brevity, we omit $\hat{\cdot}$ or $\check{\cdot}$ operations that convert elements in Lie algebra between matrix and vector representations. The notation $A^{(i)}$ refers to the i^{th} column in matrix A. The weights for the sigma points are,

$$\begin{aligned} w_m^0 &= \frac{\lambda}{n+\lambda}, \\ w_m^i &= \frac{\lambda}{n+\lambda} + (3 - \alpha^2), \\ w_c^0 = w_c^i &= \frac{1}{2(n+\lambda)},\ i = 1, 2, \ldots, 2n. \end{aligned} \tag{3.13}$$

w_m's and w_c's are used to recover the first and second moments of the Gaussian distribution respectively. We define $\mathcal{S} = s(\boldsymbol{x}, \Sigma)$ with $\mathcal{S} = \big\{\big(\boldsymbol{s}^i, w_m^i, w_c^i\big), i = 0, 1, \ldots, 2n\big\}$, to represent the process of generating sigma points, (3.12), and the weights, (3.13), from a Gaussian distribution, $\mathcal{N}(\boldsymbol{x}, \Sigma)$.

At the prediction step, sigma points generated with $\boldsymbol{b}_t = \mathcal{N}(\hat{\boldsymbol{x}}_t, \Sigma_t)$ are propagated through the motion model to obtain the prior belief $\bar{\boldsymbol{b}}_t = \mathcal{N}\big(\bar{\boldsymbol{x}}_{t+1}, \bar{\Sigma}_{t+1}\big)$,

$$
\begin{aligned}
\mathcal{S}_t &= s(\hat{\boldsymbol{x}}_t, \Sigma_t), \\
\boldsymbol{\xi}_t^i &= \log\big(f^{-1}(\boldsymbol{s}_t^0, \boldsymbol{u}_t, \boldsymbol{0}) \circ f(\boldsymbol{s}_t^i, \boldsymbol{u}_t, \boldsymbol{0})\big), \\
\bar{\boldsymbol{x}}_{t+1} &= f(\boldsymbol{s}_t^0, \boldsymbol{u}_t, \boldsymbol{0}) \circ \exp\left(\sum_{i=0}^{2n} w_m^i \boldsymbol{\xi}_t^i\right), \\
\bar{\Sigma}_{t+1} &= \sum_{i=0}^{2n} w_c^i \boldsymbol{\xi}_t^i \boldsymbol{\xi}_t^{i\top} + M_t \Sigma_{\boldsymbol{m}} M_t^\top,
\end{aligned}
\tag{3.14}
$$

where $M_t = \partial f(\hat{\boldsymbol{x}}_t, \boldsymbol{u}_t, \boldsymbol{0})/\partial \boldsymbol{m}_t$ as in (3.5) for an EKF. Note here we implicitly assume that the motion model, $f(\cdot)$, is differentiable *w.r.t.* the motion noise, $\boldsymbol{m}_t$. The assumption is less restrictive compared to assuming the differentiability of $f(\cdot)$ *w.r.t.* the state or control. One sufficient condition to satisfy the assumption is that the motion model is in, or can be approximated with, the form of $\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{m}_t) = \bar{f}(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{m}_t$. Although being more restricted compared to (3.1), it is also widely acceptable to robotics application.

At the update step, the posterior belief, $\boldsymbol{b}_{t+1} = \mathcal{N}(\hat{\boldsymbol{x}}_{t+1}, \Sigma_{t+1})$, is obtained based on the difference between the actual measurement and the measurement predicted with the sigma points,

$$
\begin{aligned}
\bar{\mathcal{S}}_{t+1} &= s(\bar{\boldsymbol{x}}_{t+1}, \bar{\Sigma}_{t+1}), \\
\boldsymbol{\xi}_{t+1}^i &= \bar{\boldsymbol{x}}_{t+1}^{-1} \circ \bar{\boldsymbol{s}}_{t+1}^i, \\
\bar{\boldsymbol{z}}_{t+1} &= \sum_{i=0}^{2n} w_m^i h\big(\bar{\boldsymbol{s}}_{t+1}^i, \boldsymbol{0}\big), \\
\boldsymbol{\delta}_{t+1}^i &= h\big(\bar{\boldsymbol{s}}_{t+1}^i, \boldsymbol{0}\big) - \bar{\boldsymbol{z}}_{t+1},
\end{aligned}
\tag{3.15}
$$

and,

$$
\begin{aligned}
\hat{\boldsymbol{x}}_{t+1} &= \bar{\boldsymbol{x}}_{t+1} \circ \exp(K_{t+1}(\boldsymbol{z}_{t+1} - \bar{\boldsymbol{z}}_{t+1})), \\
\Sigma_{t+1} &= \bar{\Sigma}_{t+1} - K_{t+1} V_{t+1} K_{t+1},
\end{aligned}
\tag{3.16}
$$

where,

$$
\begin{aligned}
V_{t+1} &= \sum_{i=0}^{2n} w_c^i \boldsymbol{\delta}_{t+1}^i \boldsymbol{\delta}_{t+1}^{i\top} + N_{t+1} \Sigma_{\boldsymbol{n}} N_{t+1}^\top, \\
P_{t+1} &= \sum_{i=0}^{2n} w_c^i \boldsymbol{\xi}_{t+1}^i \boldsymbol{\delta}_{t+1}^{i\top}, \\
K_{t+1} &= P_{t+1} V_{t+1}^{-1}.
\end{aligned} \tag{3.17}
$$

Again, we assume the measurement model $h(\cdot)$ is differentiable *w.r.t.* the measurement noise $\boldsymbol{n}_t$ with $N_{t+1} = \partial h(\bar{\boldsymbol{x}}_{t+1}, \mathbf{0})/\partial \boldsymbol{n}_{t+1}$. As for the motion model discussed previously, the assumption can be trivially satisfied by formulating the measurement model in the form of $\boldsymbol{z}_t = h(\boldsymbol{x}_t, \boldsymbol{n}_t) = \bar{h}(\boldsymbol{x}_t) + \boldsymbol{n}_t$.

By representing the belief, $\boldsymbol{b} = \mathcal{N}(\hat{\boldsymbol{x}}, \Sigma)$, in the vector form, $\big(\hat{\boldsymbol{x}}^\top, \mathrm{vech}(\Sigma)^\top\big)^\top$, the belief dynamics modeled by a UKF can be rewritten compactly as,

$$
\boldsymbol{b}_{t+1} = \begin{pmatrix} \bar{\boldsymbol{x}}_{t+1} \circ \exp\Big(\sqrt{K_{t+1} V_{t+1} K_{t+1}^\top}\, \boldsymbol{w}_{t+1}\Big) \\ \mathrm{vech}\big(\bar{\Sigma}_{t+1} - K_{t+1} V_{t+1} K_{t+1}^\top\big) \end{pmatrix}. \tag{3.18}
$$

Note that (3.18) is in the same form of (3.6). Therefore, the iLQG as in Alg. 1 can be applied seamlessly. Meanwhile, we have avoided explicitly differentiating the motion or measurement models.

3.4 Sparse Informative Measurements

The iLQG algorithm also suffers from the sparsity of informative measurements of common sensors, such as Lidar and cameras, because of the limited sensing range. Here, by "informative measurements", we refer to measurements that are effective in reducing the uncertainty of belief. Consider Lidar as an example, informative measurements are only available when the sensor is close to an obstacle. If the sensor is sufficiently far from the obstacles with all measurements saturated at the maximum range, it cannot be known where to move to collect informative measurements by just locally perturbing the sensor pose.

In terms of optimization, it means a majority of states are saddle points, providing trivial gradient information and preventing iLQG from converging to a sensible solution.

To further understand the issue, we may take a closer look at the measurement model. The measurement models for commonly used sensors with limited sensing range are often in the form of,

$$\boldsymbol{z}_t = h(\boldsymbol{x}_t, \boldsymbol{n}_t) = \bar{h}(\boldsymbol{x}_t) + N(\boldsymbol{x}_t)\boldsymbol{n}_t. \tag{3.19}$$

In (3.19), $N(\cdot) : \mathcal{X} \mapsto \mathbb{R}$ plays the role of a state-dependent noise scaling factor. To model the limited sensing range, $N(\cdot)$ is defined as,

$$N(\boldsymbol{x}) = \begin{cases} 1 & \text{if informative measurements are available at } \boldsymbol{x} \\ \infty & \text{otherwise (features are outside sensing range).} \end{cases} \tag{3.20}$$

When $N(\boldsymbol{x})$ is infinity, it implies the Kalman gain in (3.5) or (3.17) remains zeros regardless of the local motion of the robot. Therefore, the measurements have no effect in updating the belief, which only propagates with the motion model.

In our work, we address this issue by approximating $N(\cdot)$ with a sigmoid function,

$$N_s(\boldsymbol{x}) = \frac{\xi}{1 + e^{-\nu(\bar{h}(\boldsymbol{x}) - r_m)}} + 1. \tag{3.21}$$

In (3.21), r_m represents maximum sensing range. ν controls the gradient of $N_s(\cdot)$. ξ controls the maximum value of $N_s(\cdot)$.

With the modified sensor model, iLQG can be applied. As shown in Alg. 2, an outer loop, controlling the change of ξ and ν, wraps around the original iLQG algorithm in Alg. 1. When both ξ and ν are relatively small, unlike $N(\cdot)$, $N_s(\cdot)$ provides necessary gradient information which guides the robot to collect informative measurements if necessary. By increasing both ν and ξ monotonically, $N_s(\cdot)$ converges to $N(\cdot)$ [2]. The measurement model

[2]Numerically, $N(\cdot)$ in (3.20) is not well defined because of the involvement of infinity. Here we assume $N(\cdot)$ can be approximated by $N_s(\cdot)$ at $\xi = \xi_m$ and $\nu = \nu_m$ with sufficient accuracy.

Algorithm 2: iLQG with a modified sensor model

Input: initial belief trajectory $\{\bar{\boldsymbol{b}}_0, \bar{\boldsymbol{b}}_1, \ldots, \bar{\boldsymbol{b}}_N\}$,
initial policy $\{\mu_0, \mu_1, \ldots, \mu_{N-1}\}$.

Output: local optimal policy $\{\mu_0^*, \mu_1^*, \ldots, \mu_{N-1}^*\}$.

// ξ_0 and ν_0 are predefined initial values of the sigmoid function parameters.

$\xi \leftarrow \xi_0$, $\nu \leftarrow \nu_0$

// ξ_m and ν_m are the predefined upper bounds.

while $\xi < \xi_m$ *and* $\nu < \nu_m$ **do**

  Apply Alg. 1 to obtain $\{\mu_0^*, \mu_1^*, \ldots, \mu_{N-1}^*\}$.

  $\xi \leftarrow \lambda_\xi \xi$, $\nu \leftarrow \lambda_\nu \nu$, $(\lambda_\xi, \lambda_\nu > 1)$.

  Update the belief in the nominal trajectory of the feedback policy with the new measurement model.

end

used by iLQG would eventually converge to (3.19). Therefore, the final policy is optimized for the original system.

Chapter 4

A General Online POMDP Solver

In this chapter, we consider a more general problem formulation, compared to Ch. 3, for the stochastic motion planning. We propose a general POMDP solver based on MCTS to find the control policy for continuous systems.

Recent developments on tree search based general POMDP solvers, POMCP (Silver and Veness, 2010) and DESPOT (Ye et al., 2017), have demonstrated effective solutions to large scale problems. However, two difficulties follow immediately in applying the general tree-based POMDP solvers, such as POMCP and DESPOT, to solve stochastic motion planning problems considered in this dissertation.

First, the general POMDP solvers usually assume a (finite) discrete system, but robotic systems are continuous. The continuity of the state space should not pose a problem. Instead of being represented explicitly as a finite dimensional vector, the belief of the state can be approximated with samples. The continuity of the action space can also be removed by considering only a finite set of motion primitives. However, it remains unclear how to handle the continuous observation space. A naive application of the existing tree search methods will result in an overly shallow tree, which is unlikely to produce informed actions that account for a longer horizon, as is necessary in planning problems.

The second difficulty, which is less obvious, comes from estimating the value of the

rollout policy. Consider using deterministic motion planning methods as the rollout policy. During the rollout phase of tree search based POMDP solvers, one has to find an action sequence based on a sample at a leaf node; apply the actions to the same sample; use the simulated results to estimate the value of the rollout policy. This is equivalent to the assumption of a known state at the start of rollout phase, which results in an overly optimistic estimated value. The limitation can lead to incorrect action selection in the tree policy.

Contributions

In this work, we address the stochastic motion planning problem for robotic systems with continuous measurements. Building upon POMCP (Silver and Veness, 2010), a MCTS solution of general POMDPs, we propose a new algorithm, named POMCP++.

Our POMCP++ algorithm addresses the above mentioned difficulties by introducing two major improvements to POMCP. First, measurement likelihood of the original system is distorted so that existing measurement branches in the tree can be revisited with nontrivial probability. Second, a group of samples, instead of one, is enforced to traverse downstream the same set of nodes in the tree simultaneously. The belief at a leaf node can be well represented, resulting in more accurate value estimate of the rollout policy. The correct update of the values at the traversed nodes is still guaranteed through the application of importance sampling.

Because of the distorted measurement likelihood, the estimated value from the simulation episodes is biased for the current policy. Notwithstanding, we prove that such an estimate is unbiased as the number of simulations, and the size of sample group tend to infinity. Therefore, the proposed POMCP++ algorithm is shown to be a valid MCTS algorithm.

In the following, we first formulate the problem in Sec. 4.1. In Sec. 4.2, we provide a brief overview of POMCP (Silver and Veness, 2010). Then, we introduce the improvements

of POMCP++ that address the previously mentioned limitations in Sec. 4.3. Meanwhile, we study the theoretical properties of the proposed algorithm. Supplementary proofs for the theoretical results can be found in Appendix A. The work in this chapter is published in Sun et al. (2021).

4.1 Problem Formulation

Compared to the iLQG algorithm introduced in Ch. 3, the general POMDP solvers are more forgiving in system modeling. In this case, we assume that the motion and measurement models of the system are in the forms of transition probability and measurement likelihood as in Sec. 1.2,

$$\begin{aligned} P(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{a}_t), \\ P(\boldsymbol{z}_t|\boldsymbol{x}_t), \end{aligned} \tag{4.1}$$

where $\boldsymbol{x} \in \mathcal{X}$, $\boldsymbol{a} \in \mathcal{A}$, $\boldsymbol{z} \in \mathcal{Z}$ are the state, action[1], and measurement. To reduce the complexity of the problem, we assume $\mathcal{A}$ is a finite discrete set consisting of predefined motion primitives, while both $\mathcal{X}$ and $\mathcal{Z}$ are continuous. An additional assumption of (4.1) is that one should be able to compute the measurement likelihood given a state and measurement pair. Meanwhile, a generative motion model remains sufficient.

Given the initial belief $\boldsymbol{b}_0 \in \mathcal{B}$, we assume the task could be modeled as an objective function in the form of discounted summation of stage reward up to an infinite horizon. Although summation of a finite horizon is often applied to model a task, situations are common where the number of steps to complete the task is not known a priori. Objective functions of infinite horizon, in the following form, provide the flexibility in letting the control policy implicitly determine the total number of steps.

$$V_0(\boldsymbol{b}_0) = \mathop{\mathbb{E}}_{\substack{\boldsymbol{x}_t, \boldsymbol{z}_t \\ t=0,1,\dots}} \left\{ \sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{x}_t, \boldsymbol{a}_t, \boldsymbol{x}_{t+1}) \right\}, \tag{4.2}$$

[1]We use the term "action" instead of "control" as a reminder of the assumed discrete control space.

where $r(\cdot) : \mathcal{X}^2 \times \mathcal{U} \mapsto \mathbb{R}$ is a bounded stage reward function, with $|r(\cdot)| < \infty$ and $\gamma \in (0, 1)$ is the discount factor. The bounded stage reward, together with the discounted factor within the unit interval, guarantees the objective function is always well defined.

The problem is to find a control policy $\pi = \{\boldsymbol{a}_0, \boldsymbol{a}_1, \dots\}$ that maximizes the objective function in (4.2),

$$\pi_t^* = \arg\max_{\pi_t} V_0(\boldsymbol{b}_0) = \arg\max_{\pi_t} \mathop{\mathbb{E}}_{\substack{\boldsymbol{x}_0, \boldsymbol{z}_t \\ t=0,1,\dots}} \left\{ \sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{x}_t, \boldsymbol{a}_t, \boldsymbol{x}_{t+1}) \right\} \tag{4.3}$$

The problem formulated in this chapter is close to the general problem formulation in Sec. 1.2. The mild additional assumptions include a discrete control space consisting of motion primitives and a known measurement likelihood.

4.2 Partially Observable Monte Carlo Planning (POMCP)

In this section, we provide a brief overview of POMCP (Silver and Veness, 2010), which forms the basis of our work to be introduced later in this chapter.

Alg. 3 shows the steps of POMCP. Some notations in Alg. 3 are defined as follows. $h = (\boldsymbol{a}_0, \boldsymbol{z}_0, \dots)$ represents historical information containing a sequence of actions and measurements. Together with $\boldsymbol{b}_0$, h uniquely determines the current belief, *i.e.* a node in the tree. Appending actions and measurements to h, such as $h\boldsymbol{a}$ or $h\boldsymbol{a}\boldsymbol{z}$, represent a node in the subtree of h. Specially, $h = \emptyset$ is the root node. h's in the form of $(\boldsymbol{a}_0, \boldsymbol{z}_0, \dots, \boldsymbol{z}_t)$ ended with a measurement represent the belief nodes, while those ended with an action represent the belief-action nodes. Functions $V(\cdot)$ and $N(\cdot)$ keep track of the estimated value and the number of visits of a node.

For each iteration of the algorithm, a state sample, $\boldsymbol{x}_0$ is drawn from the initial belief $\boldsymbol{b}_0$ (Line 4). The state sample is first simulated recursively following the tree policy (Line 5 and 22). Once a leaf node in the tree is met, it is expanded (Line 15). The sample is

forward simulated following the predefined rollout policy (Line 16) as a continuation. The traversed nodes are updated based on the simulation results (Line 24-25). The tree policy is, therefore, implicitly updated due to the addition of the new leaf nodes and the update of the corresponding parent nodes.

POMCP is for discrete systems. It is the first algorithm that extends MCTS (Kocsis and Szepesvári, 2006) to POMDPs. However, it cannot be directly applied to solve the problem formulated in Sec. 4.1 because of the continuous measurement space. In the following of this chapter, we introduce POMCP++, an improved version of POMCP addressing its limitations, which could work with the problem in Sec. 4.1.

Algorithm 3: POMCP

Function $\boldsymbol{a}$ = `search(`$\boldsymbol{b}_0$`)`
  $h = NULL$
  while *not* `termination()` **do**
    $\boldsymbol{x}_0 \leftarrow$ draw one sample w.r.t. $\boldsymbol{b}_0$
    `simulate(`$\boldsymbol{x}_0$, h, 0`)`
  end
  return $\arg\max_{\boldsymbol{a}} V(h\boldsymbol{a})$
end

Function $\boldsymbol{a}$ = `selectAction(`h`)`
  // c trades off exploitation and exploration.
  return $\arg\max_{\boldsymbol{a}} V(h\boldsymbol{a}) + c \cdot \sqrt{\frac{\log N(h)}{N(h\boldsymbol{a})}}$
end

Function r = `simulate(`$\boldsymbol{x}$, h, d`)`
  if $\gamma^d < \epsilon$ **then return** 0
  if h *is NULL* **then**
    foreach $\boldsymbol{a} \in \mathcal{A}$ **do** $h\boldsymbol{a} \leftarrow (N_{init}, V_{init})$
    $r =$ `rollout(`$\boldsymbol{x}$, d`)`
    return r
  end
  $\boldsymbol{a} \leftarrow$ `selectAction(`h`)`
  $\boldsymbol{x}, r_s \leftarrow$ `process(`$\boldsymbol{x}, \boldsymbol{a}$`)`
  $\boldsymbol{z} \leftarrow$ `sampleMeasurement(`$\boldsymbol{x}$`)`
  $r_t \leftarrow$ `simulate(`$\boldsymbol{x}, h\boldsymbol{a}\boldsymbol{z}, d+1$`)`
  $r \leftarrow r_s + \gamma \cdot r_t$
  $N(h) \leftarrow N(h) + 1$
  $N(h\boldsymbol{a}) \leftarrow N(h\boldsymbol{a}) + 1$
  $V(h\boldsymbol{a}) \leftarrow V(h\boldsymbol{a}) + \frac{1}{N(h\boldsymbol{a})}(r - V(h\boldsymbol{a}))$
end

4.3 POMCP++

Alg. 4 shows the proposed algorithm POMCP++, with a graphical illustration in Figure 4.1. The overall procedure in POMCP++ follows POMCP in Alg. 3. One obvious modification is the change from UCB1 (Auer et al., 2002) (Alg. 3 Line 9) to ϵ-greedy (Sutton and Barto, 2018, Ch.5) (Alg. 4 Line 26) in action selection, both of which trade off exploitation versus exploration. Kuleshov and Precup (2014) show experimentally that the relative performance of the two methods vary with problem setups. In this work, ϵ-greedy policy is applied in order to ease the theoretical analysis.

Other major changes of POMCP++ include 1) replacing forward sampling measurements with measurement selection (Alg. 4 Line 34); 2) using a group of samples (Alg. 4 Line 4), instead of a single one, to approximate belief while ensuring correct update of nodes through importance sampling; 3) estimating the value of the rollout policy (Alg. 4 Line 9) using the sample group in order to overcome over-optimism. Details of the three major changes in POMCP++ are discussed in the following subsections.

In addition to the notations defined for Alg. 3, a few more are introduced in Alg. 4. $\mathcal{S} \coloneqq \{(\boldsymbol{x}, w)\}$ is a set consisting of pairs of states and corresponding weights, approximating the belief. Function $C(\cdot)$ returns all child nodes of the input node, while $|C(\cdot)|$ represents the number of child nodes.

4.3.1 Measurement Selection

In POMCP, measurements are forward sampled (Alg. 3 Line 21) to determine whether to traverse along the tree or expand a new node. It is valid for cases of small-size finite discrete measurement space or large ones with concentrated distributions. In such cases, repeated measurements can be forward sampled with high probability, leading to the growth of the tree in depth. However, it is no longer true for continuous measurement space. The probability of generating a repeated measurement is zero for a general continuous

measurement *p.d.f.*.

To overcome the issue, our proposed algorithm POMCP++ uses a new measurement selection strategy, which distorts the measurement likelihood and enforces the selection of existing measurements. In the function of measurement selection (Alg.4 Line 34), a new measurement branch is only created with probability $(|C(h\boldsymbol{a})|+1)^{\epsilon_z}$. Recall that $|C(h\boldsymbol{a})|$ is the number of child nodes, *i.e.* existing measurement branches, for the belief-action node $h\boldsymbol{a}$. Otherwise, an existing measurement is enforced. With $\epsilon_z < 0$, a new measurement is less likely to be generated as $C(h\boldsymbol{a})$ grows.

A similar notion of measurement selection appears in another recent work, POMCPOW (Sunberg and Kochenderfer, 2018). As action selection, measurement selection in POMCPOW favors measurement branches that promise high reward. This may result in a biased estimation of the policy value. In other words, the estimated policy value obtained by following the tree may not be a valid estimation of the actual value obtained by applying the policy to the real system. The underlying reason is that, unlike actions, the robot has no control over what measurement should be acquired in the future. Future measurement is completely determined by the measurement likelihood. In principle, measurement should be forward sampled to ensure that "measurement selection" exactly follows the measurement likelihood. However, this leads to an overly shallow tree, the problem to be addressed in the first place.

With the proposed method selection method, POMCP++ could circumvent the limitation and produce unbiased policy value estimation. We postpone the theoretical analysis of the proposed measurement selection method and the induced property of the overall POMCP++ algorithm to Sec. 4.4.

4.3.2 Importance Sampling

Albeit the enforcement in selecting measurement branches, the selected measurement sequence may not be representative for a single sample from the initial belief. Instead, K

samples are drawn from the initial belief (Alg. 4 Line 4). All of the K samples are pushed downstream the tree, traversing the same set of nodes. Simply averaging the simulation results leads to inaccurate policy evaluation, since each initial state sample has different likelihood of generating the measurement sequence in the traversed path.

The theory of importance sampling comes to the rescue. Assume the selected action and measurement sequence following the belief node h_t is $\boldsymbol{a}_t, \boldsymbol{z}_t, \boldsymbol{a}_{t+1}, \boldsymbol{z}_{t+1}, \ldots$. Then the current policy value at the belief action node $h_t\boldsymbol{a}_t$ is,

$$\mathbb{E}_{X_t}\{R(X_t, A_t, Z_t)\} = \int_{X_t} R(X_t, A_t, Z_t)p(X_t|A_t, Z_t)dX_t, \tag{4.4}$$

where $X_t \coloneqq \boldsymbol{x}_{t:\infty}$, $A_t \coloneqq \boldsymbol{a}_{t:\infty}$, $Z_t \coloneqq \boldsymbol{z}_{t:\infty}$ for simplicity. $R(\cdot)$ computes the discounted summation of reward of a simulation episode. Applying Bayes rule,

$$p(X_t|A_t, Z_t) = \frac{p(Z_t|X_t, A_t)p(X_t|A_t)}{p(Z_t|A_t)}. \tag{4.5}$$

In the spirit of importance sampling, $p(X_t|A_t)$ is the proposal distribution, while $p(X_t|A_t, Z_t)$ is the target distribution. $p(X_t|A_t)$, the proposal distribution, is approximated by forward sampling. The measurement likelihood $p(Z_t|X_t, A_t)$ serves as the importance weight. $p(Z_t|A_t)$, as the normalization factor, is constant for all state sequences. Therefore, the current policy value at node $h_t\boldsymbol{a}_t$ can be approximated with,

$$\frac{1}{\eta}\sum_{i=0}^{K-1} p\big(Z_t|X_t^i, A_t\big) \cdot R(X_t^i, A_t, Z_t), \tag{4.6}$$

with $\eta = \sum_i p\big(Z_t|X_t^i, A_t\big)$ (Alg. 4 Line 62- 66).

4.3.3 The Rollout Policy

It is often desirable to apply domain knowledge in designing the rollout policy. However, since such rollout policies are usually state-dependent, direct application of them in the framework of POMCP can lead to over-optimism. More specifically, in POMCP or POM-

CPOW, a single state sample, $\boldsymbol{x}_0$, is simulated from the root to a leaf node to obtain $\boldsymbol{x}_t$, which is then fed to the rollout policy to find an action sequence, $\boldsymbol{a}^r_{t:T}$. T is implicitly determined by the length of the rollout action sequence. In the case that a feasible rollout policy cannot be found, T can be set large enough so that $\gamma^T \ll 1$ and the reward-to-go beyond T can be ignored. The reward-to-go of the rollout policy is estimated with $R(\boldsymbol{x}_{t:T}, \boldsymbol{a}^r_{t:T})$, whose actual value is,

$$\begin{aligned} &\mathop{\mathbb{E}}_{p(\boldsymbol{x}_{t:T}|\boldsymbol{a}^r_{t:T})}\{R(\boldsymbol{x}_{t:T}, \boldsymbol{a}^r_{t:T})\} \\ &= \int_{\boldsymbol{x}_{t:T}} R(\boldsymbol{x}_{t:T}, \boldsymbol{a}^r_{t:T}) p(\boldsymbol{x}_{t:T}|\boldsymbol{a}^r_{t:T}) d\boldsymbol{x}_{t:T}. \end{aligned} \tag{4.7}$$

Note that $\boldsymbol{x}_t$ may be from belief with high uncertainty at the leaf node. Since $\boldsymbol{a}^r_{t:T}$ is found using $\boldsymbol{x}_t$ in the first place, the rollout action sequence is expected to obtain high reward-to-go starting from $\boldsymbol{x}_t$. However, $\boldsymbol{a}^r_{t:T}$ may not perform well with states that are significantly different from $\boldsymbol{x}_t$. Therefore, $\mathop{\mathbb{E}}_{p(\boldsymbol{x}_{t:T}|\boldsymbol{a}^r_{t:T})}\{R(\boldsymbol{x}_{t:T}, \boldsymbol{a}^r_{t:T})\}$ may be much less than $R(\boldsymbol{x}_{t:T}, \boldsymbol{a}^r_{t:T})$. If $R(\boldsymbol{x}_{t:T}, \boldsymbol{a}^r_{t:T})$ is then back propagated to the parent nodes, the high reward-to-go may mislead action selection in later steps to favor this branch albeit the high uncertainty at the leaf node. In other words, the implicit assumption of perfect state information of $\boldsymbol{x}_t$ should be accused for the over-optimism.

Because of the measurement selection and importance sampling introduced previously, in the proposed POMCP++ algorithm, one has K state samples when reaching a leaf node. One of the K samples, $\boldsymbol{x}_t$, can be chosen and used to find the rollout action sequence $\boldsymbol{a}^r_{t:T}$. The estimated reward for the rollout policy is then $\sum_{i=0}^{K-1} R\big(\boldsymbol{x}^i_{t:T}, \boldsymbol{a}^r_{t:T}\big)$ (Alg. 4 Line 9).

Removal of the implicit perfect state information assumption at the leaf nodes helps address the active information acquisition problem which often shows up in robotic applications. Active information acquisition requires the robot to actively collect informative measurements to reduce uncertainty and, therefore, complete the task with high confidence. In POMCP++, since the action sequence $\boldsymbol{a}^r_{t:T}$ is constructed with $\boldsymbol{x}_t$, the rollout policy only attains high reward-to-go when all samples are close to $\boldsymbol{x}_t$. Such mechanism implicitly motivates the robot to reduce uncertainty whenever necessary.

Algorithm 4: POMCP++

Fn $\boldsymbol{a}$ = `search(`$\boldsymbol{b}_0$`)`
  $h = NULL$
  while *not* `termination()` **do**
    $\mathcal{S} \leftarrow$ draw K samples w.r.t $\boldsymbol{b}_0$
    `simulate(`$\mathcal{S}$, h, 0`)`
  end
  return $\arg\max_{\boldsymbol{a}} V(h\boldsymbol{a})$
end

Fn $\boldsymbol{r}$ = `rollout(`$\mathcal{S}$, d`)`
  if $\gamma^d < \epsilon$ **then return 0**
  sample $\boldsymbol{x} \sim \mathcal{S}$ w.r.t $\boldsymbol{w}$
  $\{\boldsymbol{a}_r\} \leftarrow$ `rolloutPolicy(`$\boldsymbol{x}$`)`
  $\boldsymbol{r} \leftarrow \mathbf{0}$
  foreach $\boldsymbol{x}_i, r_i \in \mathcal{S}, \boldsymbol{r}$ **do**
    foreach $\boldsymbol{a}_{r,j} \in \{\boldsymbol{a}_r\}$ **do**
      if $\gamma^{j+d} < \epsilon$ **then** break
      $\boldsymbol{x}_i, r \leftarrow$ `process(`$\boldsymbol{x}_i$, $\boldsymbol{a}_{r,j}$`)`
      $r_i \leftarrow r_i + \gamma^j \cdot r$
    end
    if $\boldsymbol{x}_i$ *is not at the target location* **then**
      // $\bar{r}$ is the average stage reward.
      $r_i \leftarrow r_i + \gamma^{len(\boldsymbol{a}_d)} \cdot \bar{r}$
    end
  end
  return $\boldsymbol{r}$
end

Fn $\boldsymbol{a}$ = `selAction(`h`)`
  if `rand()` $> \epsilon_a$ **then**
    $\boldsymbol{a} \leftarrow \arg\max_{\boldsymbol{a}} V(h\boldsymbol{a})$
  else
    uniformly choose $\boldsymbol{a} \in \mathcal{A}$
  end
  return $\boldsymbol{a}$
end

Fn $\boldsymbol{z}$ = `selMeasurement(`$h\boldsymbol{a}$, $\mathcal{S}$`)`
  if `rand()` $< \left(|C(h\boldsymbol{a})| + 1\right)^{\epsilon_z}$ **then**
    sample $\boldsymbol{x} \sim \mathcal{S}$ w.r.t $\boldsymbol{w}$
    $\boldsymbol{z} \leftarrow$ `sampleMeasurement(`$\boldsymbol{x}$`)`
  else
    uniformly choose $\boldsymbol{z}$ from $C(h\boldsymbol{a})$
  end
  return $\boldsymbol{z}$
end

Fn $\boldsymbol{r}, \boldsymbol{w}$ = `simulate(`$\mathcal{S}$, h, d`)`
  if $\gamma^d < \epsilon$ **then return** $\mathbf{0}, \mathbf{1}$
  if h *is NULL* **then**
    foreach $\boldsymbol{a} \in \mathcal{A}_m$ **do**
      $h\boldsymbol{a} \leftarrow (N_{init}, V_{init})$
    end
    $\boldsymbol{r} =$ `rollout(`$\mathcal{S}$, d`)`
    return $\boldsymbol{r}, \mathbf{1}$
  end
  $\boldsymbol{a} \leftarrow$ `selAction(`h`)`
  $\mathcal{S}, \boldsymbol{r}_s \leftarrow$ `process(`$\mathcal{S}$, $\boldsymbol{a}$`)`
  $\boldsymbol{z} \leftarrow$ `selMeasurement(`$h\boldsymbol{a}$, $\mathcal{S}$`)`
  $\boldsymbol{w}_s \leftarrow$ `likelihood(`$\mathcal{S}$, $\boldsymbol{z}$`)`
  $\boldsymbol{w} \leftarrow \{w_i\}$ s.t. $(\boldsymbol{x}_i, w_i) \in \mathcal{S}$
  foreach $w_i, w_{s,i} \in \boldsymbol{w}, \boldsymbol{w}_s$ **do**
    $w_i \leftarrow w_i \cdot w_{s,i}$
  end
  $\boldsymbol{r}_t, \boldsymbol{w}_t \leftarrow$ `simulate(`$\mathcal{S}$, $h\boldsymbol{a}\boldsymbol{z}$, $d+1$`)`
  $\boldsymbol{r} = \boldsymbol{r}_s + \gamma \cdot \boldsymbol{r}_t$
  foreach $w_i, w_{s,i}, w_{t,i} \in \boldsymbol{w}, \boldsymbol{w}_s, \boldsymbol{w}_t$ **do**
    $w_i \leftarrow w_{s,i} \cdot w_{t,i}$
  end
  $\boldsymbol{w} \leftarrow \boldsymbol{w} / \|\boldsymbol{w}\|_2$
  $r \leftarrow \boldsymbol{w}^\top \boldsymbol{r}$
  $N(h\boldsymbol{a}) \leftarrow N(h\boldsymbol{a}) + 1$
  $V(h\boldsymbol{a}) \leftarrow V(h\boldsymbol{a}) + \frac{r - V(h\boldsymbol{a})}{N(h\boldsymbol{a})}$
  return $\boldsymbol{r}$, $\boldsymbol{w}$
end

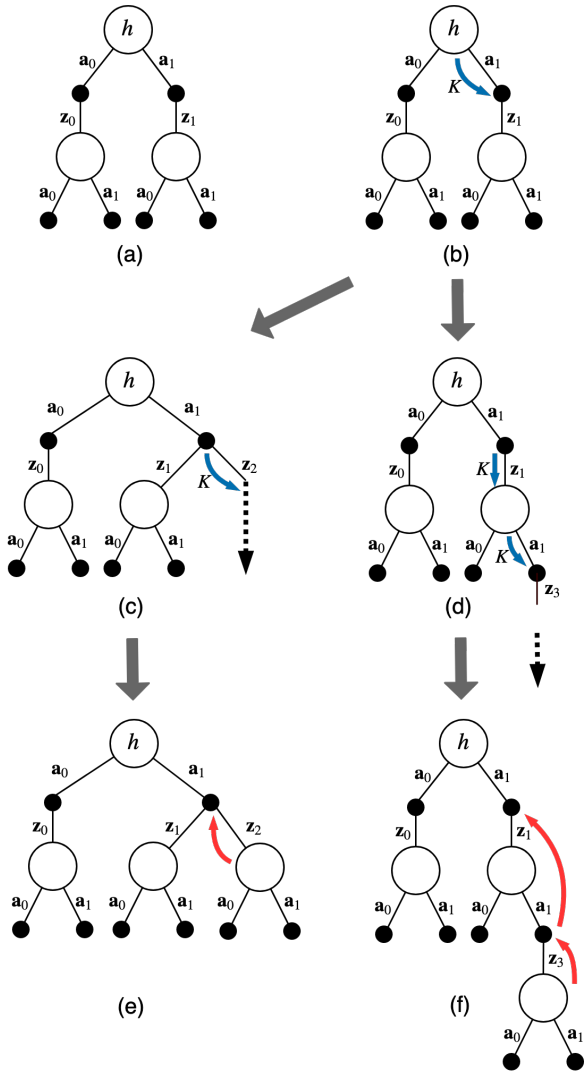


Figure 4.1: (a) shows a starting tree configuration with two possible actions, $\boldsymbol{a}_0$ and $\boldsymbol{a}_1$, rooted at h. "◯" and "•" denote the belief and belief action nodes respectively. (b) assumes K samples are drawn from the initial belief and $\boldsymbol{a}_1$ is selected. The flow of the samples are marked by "→". After forward simulating the motion model, different cases happen depending on whether (c) a new measurement branch should be created or (d) an existing measurement branch should be followed. In the case of creating a new measurement branch as in (c), rollout policy is followed afterwards, denoted as "···". (e) shows the creation of the new leaf nodes. The simulation results are back propagated to the belief action node, shown as "→". The difference in (d) and (f) is that action and measurement selection procedure are repeated until a leaf node is encountered. The simulation results are, again, backpropagated to all traversed belief action nodes.

4.4 Theoretical Analysis of POMCP++

In this section, we analyze the theoretical properties of the proposed POMCP++ algorithm. For brevity, we only list the conclusions here, while intermediate results and proofs are provided in Appendix A.

First, we attempt to understand the properties of the measurement selection method introduced in Sec. 4.3.1. We would like to study, under the measurement selection method, how often a node in the tree is visited and how many child nodes a belief action node would spawn. The answer to the above questions are summarized by the following two theorems.

Theorem 4.1. *If the root node is visited i.o., i.e.* $N \to \infty$, $0 < \epsilon_a < 1$, *and* $\epsilon_z < 0$, *all nodes will be visited i.o..*

Theorem 4.2. *If a belief action node* h *is visited i.o., it spawns child nodes i.o. given* $-1 \leq \epsilon_z < 0$.

Next, we would like to show that POMCP++ is a valid MCTS algorithm, which is, at its base, an asynchronous value iteration algorithm interleaving policy evaluation and policy improvement. Since POMCP++ already shares the algorithm structure with MCTS, the validity of POMCP++ depends on whether the value computed from the simulation episodes serves as unbiased estimates of value at the corresponding belief-action nodes. It seems straightforward, at first glance, that estimation from simulation episodes is unbiased. However, one should be careful in drawing the conclusion since the measurements in simulation episodes are not always obtained by forward simulating the measurement model, but are enforced to be repeated most of the time.

The key in filling the gap comes from Theorem 4.1 and Theorem 4.2. Combining the two theorems, it is implied that a belief node in the tree will spawn infinitely many different measurement branches with a properly chosen ϵ_z. As a result, selecting an existing child measurement branch of a belief-action node following the measurement selection method

would accurately approximate the process of sampling a measurement from the underlying measurement likelihood. In other words, the measurement branches/samples of a belief-action node provide an accurate approximation of the corresponding measurement distribution over the continuous measurement space. Therefore, we have the following theorem under the condition of sufficient simulation episodes, N, and samples, K, in approximating the belief.

Theorem 4.3. *The discounted summation of rewards of the simulation episode is an unbiased estimate of the value at all traversed nodes as* $N \to \infty$ *and* $K \to \infty$.

Finally, we would like to provide a few remarks on the convergence of POMCP++. Combining unbiased policy evaluation (Theorem 4.3) and valid policy improvement (ϵ-greedy at Line 26 in Alg. 4), POMCP++ is a valid MCTS method. However, to the best of our knowledge, there is no formal proof yet that MCTS would converge to the optimal action at the root when applied to stochastic systems. Kocsis and Szepesvári (2006) presents conclusions on the convergence of UCT. Unfortunately, the details of the proof are not available. On a related note, it is also commented by Sutton and Barto (2018) and quoted "*Convergence to this optimal fixed point seems inevitable as the changes to the action-value function decrease over time, but has not yet been formally proved. In our opinion, this is one of the most fundamental open theoretical questions in reinforcement learning (for a partial solution, see Tsitsiklis (2003)).*"[2]

[2]The comment of Sutton and Barto (2018) is on the convergence of Monte Carlo Control, a more general algorithm that MCTS is based on.

Chapter 5

Stochastic Motion Planning with System (Internal) Uncertainty

Ch. 3 and 4 introduce different stochastic motion planning solutions with an abstract problem formulation. In this chapter, we apply both methods in a concrete application where the dominant uncertainty comes from the robot itself. More specifically, we consider the application of navigating a car-like robot with range sensors in a known environment.

Navigating a mobile robot within a known environment may be considered as a well-studied and solved problem in the robotics community (Thrun et al., 2005; LaValle, 2006; Choset et al., 2005). Besides the advancement of estimation, planning, and control algorithms, much of the simplicity of the problem also benefits from the improvement of sensors, such as high-frequency and accurate IMUs, dense and long-range Lidars, and high-resolution and low-cost cameras. State estimation algorithms could often produce close-to-ground-truth estimates with just passively collected data[1]. As a result, it may be reasonable to assume perfect state knowledge and leverage the separation theorem in similar applications.

However, when the sensor has a relatively small footprint compared to the deployed environment, the assumption of perfect state knowledge fails. Meanwhile, given the nonlin-

[1]By "passively collected data", we refer to the sensor measurements collected by the robot without intentionally caring about the actual content of the data.

ear nature of the considered application, the separation theorem no longer applies. Naively applying the practical solutions in this case would no longer produce expected outcome. Instead, the robot has to make decisions respecting uncertainties from different sources in order to complete the task with a high probability while maintaining safety. By demonstrating the superior performance of the proposed algorithms in such a familiar application, we hope more researchers could be convinced with the importance of stochastic motion planning.

Contributions

In this chapter, we formulate the mobile robot navigation problem as required in Ch. 3 and 4 respectively, and apply the proposed methods in the previous two chapters to plan motions for the robot.

The proposed methods are tested through extensive simulations. Meanwhile, We compare the performance of the proposed methods with other methods that could also work with the same problem formulations, including practical solutions assuming the separation theorem, belief space planning methods assuming maximum likelihood measurements, and other general POMDP solutions. The performance of the methods are compared not only in terms of the objective functions, but also other metrics that are often of concern in robotics applications, such as failure rates due to different causes, traveled distance, total number of steps, final entropy of belief, *etc.* We show that the proposed stochastic motion planning algorithms excel in terms of the considered metrics. Meanwhile, we demonstrate the active localization behavior of the robot controlled with the proposed methods, even though the behavior is never explicitly encoded into the objective functions.

The rest of the chapter is organized as follows. We first apply the improved version of iLQG proposed in Ch. 3 to the application. In Sec. 5.1, we formulate the navigation problem as required by Sec. 3.1. Following which, we present the simulation setup and report the performance of improved iLQG together with other methods considered in comparisons.

A similar organization is employed in Sec. 5.2, where we apply POMCP++, proposed in Ch. 4, in the same application. The reported experimental results in this chapter are also published in Sun and Kumar (2021) and Sun et al. (2021).

5.1 Motion Planning with iLQG

In this section, we try to plan motions for the mobile robot using the improved iLQG algorithm introduced in Ch. 3.

We start by formulating the models for the system and the task in the form of (3.1). Then, we test the performance the algorithm in simulated environments. First, with the considered application, we show the effectiveness of the proposed modifications of iLQG, namely UKF belief dynamics and informative measurement densification, introduced in Sec. 3.3 and 3.4 respectively. Second, we apply the proposed algorithm in simulations of large scale real world environments. By comparing to state-of-the-art solutions that either assume the separation theorem or the maximum likelihood measurements, we show the proposed algorithm could achieve superior performance.

5.1.1 System and Task Models

We assume the mobile robot is controlled with velocity command. The discrete-time dynamics is,

$$\boldsymbol{\xi} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} (\boldsymbol{u}_t + \boldsymbol{m}_t),\ \boldsymbol{m}_t \sim \mathcal{N}\left(\mathbf{0}, \begin{pmatrix} \sigma_v & 0 \\ 0 & \sigma_\omega \end{pmatrix}\right), \tag{5.1}$$

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{m}_t) = \boldsymbol{x}_t \circ \exp(\boldsymbol{\xi} \cdot \tau).$$

In (5.1), $\boldsymbol{x} \in SE(2)$ is the state of the mobile robot. $\boldsymbol{u} = (v, \omega)^\top \in \mathbb{R}^2$ is the control input, consisting of linear velocity, v, and angular velocity, ω. $\boldsymbol{m}$ is the $i.i.d.$ motion noise of a Gaussian distribution. $\boldsymbol{\xi}$ is the 2-D twist. τ is the duration of each discrete time step. Within each discrete time step, the control input and motion noise are assumed to

be constant. Finally, $\circ$ and $\exp(\cdot)$ denote the composition and exponential operations on $SE(2)$. For brevity, we omit $\hat{\cdot}$ or $\check{\cdot}$ operations that convert elements in Lie algebra between matrix and vector representations.

The range sensor on the robot is modeled as follows,

$$\begin{aligned}
\boldsymbol{n}_t &= \left(n_t^1, n_t^2, \ldots, n_t^m\right)^\top \sim \mathcal{N}(\mathbf{0}, \sigma_n \cdot I), \\
z_t^i &= r(\boldsymbol{x}_t, \boldsymbol{o}, \theta^i) + N\big(r(\boldsymbol{x}_t, \boldsymbol{o}, \theta^i)\big) n_t^i, \quad i = 1, \ldots, m, \\
N(r) &= \begin{cases} 1 & \text{if } r < r_m, \\ \infty & \text{otherwise,} \end{cases} \\
\boldsymbol{z}_t &= h(\boldsymbol{x}_t, \boldsymbol{n}_t) = \left(z_t^1, z_t^2, \ldots, z_t^m\right)^\top.
\end{aligned} \tag{5.2}$$

In (5.2), $\boldsymbol{z} \in \mathbb{R}^m$ consists of range measurements from m beams. $\boldsymbol{n}$ is the *i.i.d* Gaussian measurement noise. $\boldsymbol{o}$ is the known map in the form of an occupancy grid. $r(\boldsymbol{x}, \boldsymbol{o}, \theta)$ models the ray casting process of a beam oriented at θ in the frame $\boldsymbol{x}$. The noise of a range measurement is scaled by $N(r)$, depending on whether the beam hits an obstacle or not. As in (3.20), the noise has standard deviation σ_n if the predicted range r is within the maximum sensing range r_m. Otherwise, the noise standard deviation is infinite, indicating that measurements that beyond sensing range are not effective in update the belief[2].

To model the motion planning task, the cost functions used in this work include three kinds of objectives, namely reaching the goal, minimizing control effort, and avoiding colli-

[2]The measurement model in (5.2) does not exactly match the behavior of a range sensor in practice when $r \geq r_m$. In the cases that $r \geq r_m$, range measurements of real sensors often saturate at r_m. For the considered application, the measurement model is only used to construct belief dynamics. Scaling the noise with infinity simply means that measurements with $r \geq r_m$ have no effect in belief update. In practice, the same assumption is often applied in range sensor based estimation algorithms (Pomerleau et al., 2013).

sions. More specifically, the stage cost is in the form of,

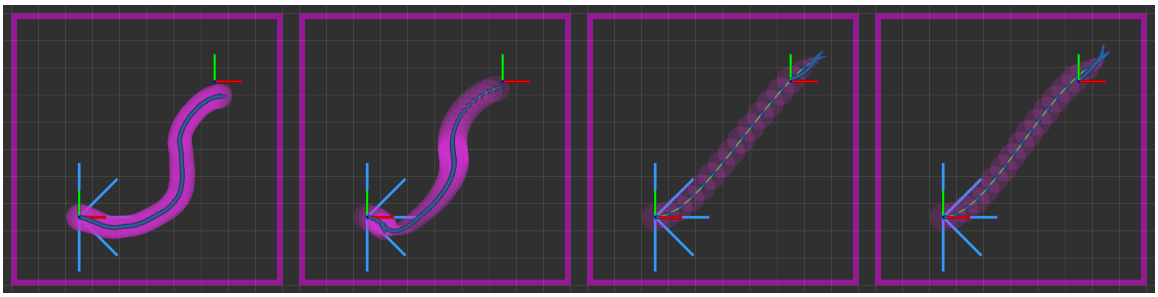
$$
\begin{aligned}
c_t(\boldsymbol{b}_t, \boldsymbol{u}_t) = & \log^{\top}(\boldsymbol{b}_t^{-1} \circ \boldsymbol{b}_g) \cdot Q_t^{\boldsymbol{b}} \cdot \log(\boldsymbol{b}_t^{-1} \circ \boldsymbol{b}_g) \\
& + \boldsymbol{u}_t^{\top} R_t \boldsymbol{u}_t \\
& - q_c \cdot \log\left(\Gamma^{-1}(1) \cdot \gamma\left(1, \frac{\sigma^2(\boldsymbol{b}_t, \boldsymbol{o})}{2}\right)\right) \\
& + q_d \cdot d^{-1}(\boldsymbol{b}_t, \boldsymbol{o}).
\end{aligned}
\tag{5.3}
$$

In (5.3), the first two terms are quadratic costs trying to reduce the distance to the goal belief state, and the control magnitude. We approximate the belief space $\mathcal{B}$ with $SE(2) \times \mathbb{R}^6$, with $SE(2)$ and $\mathbb{R}^6$ for mean and uncertainty respectively. Compared to approximating $\mathcal{B}$ with $\mathbb{R}^9$, which combines mean and uncertainty into a single vector space, $SE(2) \times \mathbb{R}^6$ respects the kinematic constraints of the state space, therefore, can more accurately reflect the difference between belief states.

To avoid collisions, we use the method proposed by van den Berg et al. (2012b) to approximate the collision probability. The function $\sigma(\boldsymbol{b}, \boldsymbol{o})$ is defined as $\min_{\boldsymbol{c}} \left\|\hat{\boldsymbol{t}} - \boldsymbol{c}\right\|_{\Sigma}$, the minimum normalized distance (normalized with the uncertainty Σ) between the position estimate $\hat{\boldsymbol{t}}$ and occupied cells $\boldsymbol{c}$ in $\boldsymbol{o}$. With $\sigma(\boldsymbol{b}, \boldsymbol{o})$, the regularized gamma function, $\Gamma^{-1}(1)\gamma(\cdot)$, provides a lower bound for the probability of not colliding with obstacles. The optimization increases the clearance probability by minimizing the negative logarithm of the regularized gamma function.

In practice, we find the collision probability cost may degenerate when the uncertainty, Σ, is small. For the extreme case when $\Sigma = 0$, the collision probability cost is zero regardless of the robot position. To resolve this issue, we introduce an additional cost, $d^{-1}(\boldsymbol{b}, \boldsymbol{o})$, depending on the absolute Euclidean distance between the position estimate and the occupied cells. $d(\boldsymbol{b}, \boldsymbol{o})$ is defined as $\min_{\boldsymbol{c}} \|\hat{\boldsymbol{t}} - \boldsymbol{c}\|_2$.

The terminal cost c_N is defined in a similar way as c_t, The minor differences between



(a) initial guess (b) N-iLQG (c) E-iLQG (d) U-iLQG

Figure 5.1: Belief nominal trajectories of (a) the initial guess, (b) N-iLQG, (c) E-iLQG, and (d) U-iLQG on a boundary map (100×100). In each figure, the frames at the bottom left and upper right mark the start and goal locations. The light blue lines represent five range sensor beams at the maximum range (2m). The dark blue arrows and purple ellipses mark the expected poses and one-standard-deviation position uncertainty.

c_N and c_t include the change of parameter, such as $Q^{\boldsymbol{b}}$, and the removal of control effort cost.

5.1.2 Experiments

For all simulations, the resolution of the occupancy grid map, *i.e.* cell size, is assumed to be 0.1m. In the motion model, (5.1), the time interval is set to $\tau = 0.1$s, motion noise is set to $\sigma_v = 0.5$m/s, $\sigma_\omega = 0.05$rad/s. We assume the application of a low-cost range sensor with small footprint. The range sensor has five beams oriented at $\{-\pi/2, -\pi/4, 0, \pi/4, \pi/2\text{rad}\}$ with maximum range $r_m = 2$m. The measurement noise in (5.2) is set to $\sigma_z = 0.5$m. Sigmoid function parameters ξ_m and ν_m, in Alg. 2, are both fixed at 1000 for all simulations, while ξ_0, ν_0, λ_ξ, and λ_ν are reported for individual test cases. The initial trajectories are created with stable sparse RRT (Li et al., 2016) implemented in the Open Motion Planning Library (OMPL) (Şucan et al., 2012). The algorithms are timed on a laptop scale computer with Intel i7-6670HQ CPU (4 cores at 2.6GHz) and 32GB RAM.

An Ablation Study

We perform an ablation study to demonstrate the effectiveness of the proposed improvements to iLQG in Ch. 3. In the first variant, we naively apply iLQG on the measurement

Table 5.1: Comparing N-iLQG, E-iLQG, and U-iLQG on the boundary map[1]

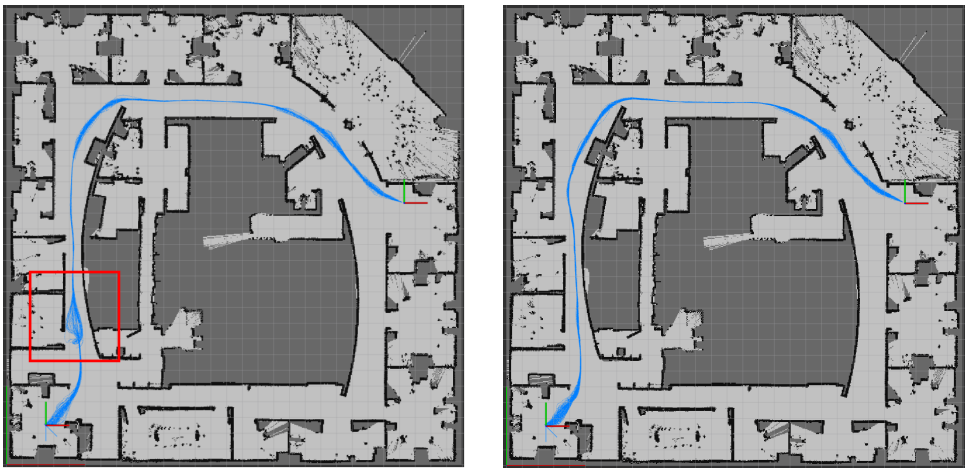
method	ξ_0/ν_0	λ_ξ/λ_ν	$\mathbb{E}(c)$	CR[2]	iter.[3]	time (s)
N-iLQG	$1e3/1e3$[4]	–	2229.89	0.00	200[5]	14.71
E-iLQG	10.0/5.0	2.0/2.0	872.94	0.02	112	9.13
U-iLQG	10.0/5.0	1.2/1.2	853.04	0.02	227	47.90
		2.0/2.0	**850.78**	0.01	91	16.79
		4.0/4.0	861.97	0.02	82	13.61
	5.0/2.5	2.0/2.0	1581.43	0.00	143	21.87
	20.0/10.0		1802.00	0.00	157	22.05

[1] The cost and collision rate are evaluated over 100 Monte Carlo simulations.
[2] CR refers to collision rate.
[3] The total number of iLQG iterations summed over all outer loops if any. The inner loops of iLQG that adjust the damping factor of the Levenberg–Marquardt algorithm are not counted.
[4] For N-iLQG, we directly set $\xi_0 = \xi_m = 1e3$ and $\nu_0 = \nu_m = 1e3$.
[5] For each iLQG optimization, the number of iterations is capped at 200, which is otherwise terminated based on either absolute or relative cost reduction.

model, (5.2), without using the sigmoid function approximation. In the second variant, we use an EKF to approximate the belief dynamics, where numerical differentiation is applied to approximate the measurement Jacobians. For sanity, we refer to the two variants as N-iLQG and E-iLQG. The proposed method with both modifications is named as U-iLQG.

As show in Figure 5.1, N-iLQG is not able to utilize the boundaries for localization without the sigmoid function approximation. In contrast, both E-iLQG and U-iLQG are able to utilize the top right corner to reduce the localization uncertainty, even though the structure is not within the measurement range of the initial trajectory. Comparing E-iLQG and U-iLQG, the nominal trajectories in the optimized feedback control policies are similar. However, the lower cost of U-iLQG, shown in Table 5.1 (comparing the entries with $\xi_0 = 10.0$, $\nu_0 = 5.0$, and $\lambda_\xi = \lambda_\nu = 2.0$), confirms that UKFs could more accurately model the belief dynamics.

Meanwhile, we study the effect of the parameters in the sigmoid function on U-iLQG.



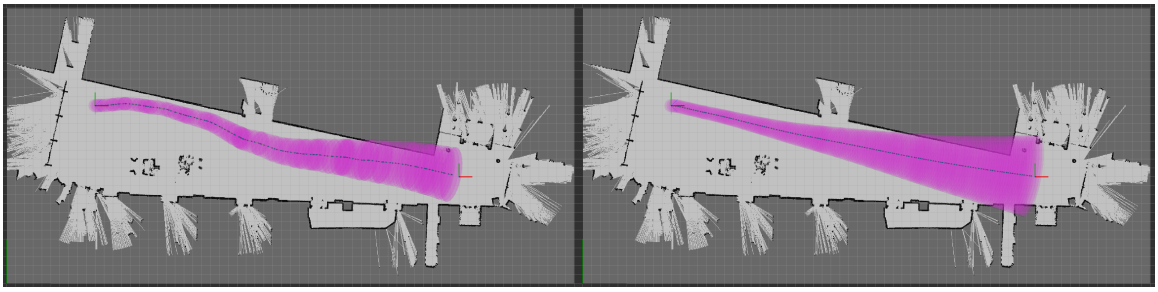
(a) M-iLQG (b) U-iLQG

Figure 5.2: Mean of simulated belief trajectories using policies optimized with (a) M-iLQG and (b) U-iLQG on "intel". Note the marked region in (a). The robot just enters the corridor with relatively large uncertainty. The measurements collected at the doorway induce large innovation, which leads to instability of the policy optimized with M-iLQG. The same problem does not appear with U-iLQG.

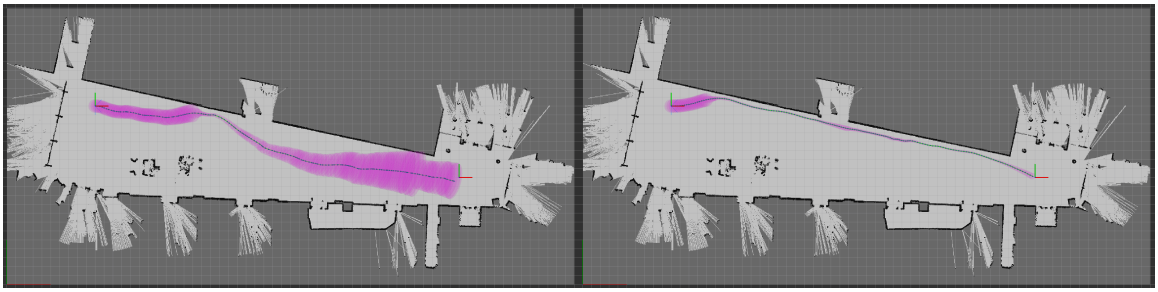
In the simulations, a reasonable combination of the parameters, $\xi_0 = 10.0$, $\nu_0 = 5.0$, $\lambda_\xi = \lambda_\nu = 2.0$, is determined through trial and error. By changing the parameters in the neighborhood, it could be observed in Table 5.1 that the optimized cost is more sensitive to the initial values, ξ_0 and ν_0, compared to the scaling factors, λ_ξ and λ_ν. As general nonlinear optimization problems (Nocedal and Wright, 2006), systematically determining and scheduling the parameters λ_ξ and λ_ν are hard, but remain as a promising future research direction.

Real World Environments

We also apply the proposed method to large scale maps of real world environments constructed with 2-D Lidar (Cyrill, 2020). The two maps, named "fr101" (1279×620) and "intel" (579×581), are representative for environments of different clutteredness. In order to be used in this work, the probabilistic cells in the map are classified as free, occupied, or unknown through thresholding. The unknown cells are treated as occupied for collision detection in the motion model, while treated as free for ray casting in the measurement



(a) initial guess (b) iLQR



(c) M-iLQG (d) U-iLQG

(e) initial guess (f) iLQR (g) M-iLQG (h) U-iLQG

Figure 5.3: Belief nominal trajectories of (a, e) the initial guess, (b, f) iLQG, (c, g) M-iLQG, and (d, h) U-iLQG in real world environments (a-d) "fr101" (1279×620) and (e-h) "intel" (579×581). In each figure, the frames on the left and right mark the start and goal location. The dark blue arrows and purple ellipses mark the expected poses and three-standard-deviation position uncertainty.

Table 5.2: Comparing iLQR, M-iLQG, and U-iLQG on map "fr101" and "intel"[1, 2]

map	method	ξ_0/ν_0	$\mathbb{E}(c)$	CR	iter.	time (s)
fr101	iLQR	–	$1.90e4$	0.00	15	58.94
	M-iLQG	50.0/25.0	$9.17e3$	0.00	34	654.56
	U-iLQG	50.0/25.0	**3.26e3**	0.00	26	703.41
intel	iLQR	–	$2.15e3$	0.06	11	45.72
	M-iLQG	250.0/125.0	$1.92e3$	0.22	25	369.03
	U-iLQG	250.0/125.0	**1.79e3**	0.00	22	631.78

[1] See Table 5.1 for related notes.
[2] For M-iLQG and U-iLQG, all simulations use $\lambda_\xi = \lambda_\nu = 2.0$.

model.

The performance of the proposed method is compared with the existing state-of-the-art methods. In practice, the separation theorem is often extended to nonlinear systems to solve stochastic motion planning problems. In this case, iLQR (Li and Todorov, 2004) is directly applied to the motion model in (5.1). At the online phase, a UKF is used to estimate belief, the mean of which is fed back to the iLQR policy to generate control inputs. In the following, we refer to this method as iLQR. We also compare the proposed method with methods that assume maximum likelihood measurements. The assumption is equivalent to assuming $\boldsymbol{w}_{t+1} = \mathbf{0}$ in (3.18). We refer to this method as M-iLQG in the following. Note that the modifications in Sec. 3.3 and 3.4 are also applied to M-iLQG in the comparison.

As shown in Figure 5.3a-d, the feedback policy of U-iLQG is able to actively localize the robot by moving along the wall in an open environment. Therefore, as shown in Table 5.2, the cost of U-iLQG is significantly lower compared with the other two methods. In a more cluttered environment, Figure 5.3e-h, active localization is no longer necessary. The nominal trajectories are similar for all methods. However, Table 5.2 shows the collision rate of iLQR and M-iLQG are higher compared to U-iLQG. The high collision rate has different causes for iLQR and M-iLQG. In iLQR, the nominal trajectory is over close to the wall

since the state estimation uncertainty is ignored. For M-iLQG, the feedback policy is less robust to innovation noise as a result of assuming deterministic belief dynamics. Especially, when $\partial \boldsymbol{b}_{t+1}/\partial \boldsymbol{w}_{t+1}$ is large, the nonzero innovation may result in a large update in belief which deviates significantly from the nominal trajectory, and possibly leads to collisions in M-iLQG. Figure 5.2 illustrates the issue of M-iLQG with simulated belief trajectories.

5.2 Motion Planning with POMCP++

In this section, we apply the POMCP++ algorithm introduced in Ch. 4 to the same application of mobile robot navigation.

As shown in Sec. 4.1, POMCP++, as a general POMDP solver, allows a more general problem formulation. Therefore, noise characteristics of the system can be better captured in both motion and measurement models. Meanwhile, there is more flexibility in modeling the task where we could avoid juggling the weights for different cost terms as in (5.3).

In the experiments, we first compare POMCP++ with other state-of-the-art methods in relatively simple simulated environments, where the optimal policy can be anticipated. The performance of different methods are compared in terms of the summation of discounted reward. The discounted reward is the objective to be maximized, and is widely used as a metric to quantify the performance of general POMDP solvers in the literature. Second, we evaluate the methods in a more realistic hallway environment. The performance of the methods are compared in terms of metrics that are of more interest to robotics community, such as the rate of collisions, total number of actions, the entropy of the final distribution, *etc.* Finally, we demonstrate the usage of POMCP++ by transitioning the simulation in the hallway environment to hardware experiments.

5.2.1 System and Task Models

The discrete-time dynamics of the robot is modeled as (Thrun et al., 2005).

$$\begin{pmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{pmatrix} = \begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix} + \begin{pmatrix} v_t\tau_t \cdot \mathrm{sinc}\left(\frac{\omega_t\tau_t}{2}\right)\cos(\theta_t + \frac{\omega_t\tau_t}{2}) \\ v_t\tau_t \cdot \mathrm{sinc}\left(\frac{\omega_t\tau_t}{2}\right)\sin(\theta_t + \frac{\omega_t\tau_t}{2}) \\ \omega_t\tau_t \end{pmatrix}, \tag{5.4}$$

where $\boldsymbol{x} = (x, y, \theta)^\top \in SE(2)$ is the state of the robot representing the position and orientation. $\boldsymbol{a} = (v, \omega)^\top \in \mathcal{A} \subset \mathbb{R}^2$ represents the velocity command consisting of linear velocity, v, and angular velocity, ω. In this work, we restrict the allowable controls to a finite subset $\mathcal{A} \subseteq \mathbb{R}^2$, *i.e.* motion primitives, of the control space in order to reduce the complexity of the problem.

To model the uncertainty of system dynamics, the input command $\boldsymbol{a}$ is distinguished from the executed command $\tilde{\boldsymbol{a}} = (\tilde{v}, \tilde{\omega})$,

$$\begin{aligned} \tilde{v} &= v + n_v, \\ \tilde{\omega} &= \omega + n_\omega + n_\gamma, \\ n_v &\sim \mathcal{N}(0, \alpha_v(v)^2 + \beta_v(\omega)^2), \\ n_\omega &\sim \mathcal{N}(0, \alpha_\omega(v)^2 + \beta_\omega(\omega)^2), \\ n_\gamma &\sim \mathcal{N}(0, \alpha_\gamma(v)^2 + \beta_\gamma(\omega)^2). \end{aligned} \tag{5.5}$$

where $\alpha_{\{\alpha,\omega,\gamma\}}$ and $\beta_{\{\alpha,\omega,\gamma\}}$ are empirical parameters controlling the scale of the motion noise.

Measurements, $\boldsymbol{z} \in \mathbb{R}^m$, are ranges of the beams produced by a 2-D Lidar. Assuming the range of each beam, z_i $i = 0, 1, \ldots, m-1$ is independent, the measurement likelihood of $\boldsymbol{z}$ is the product of the *p.d.f.* of z_i's, *i.e.*

$$p(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{o}) = \prod_i p(z_i|\boldsymbol{x}, \boldsymbol{o}), \tag{5.6}$$

where $\boldsymbol{o}$ is a known occupancy grid map. Therefore, it is sufficient to model a single beam. In this work, we use the beam model (Thrun et al., 2005),

$$p(z|\boldsymbol{x},\boldsymbol{o}) = \begin{pmatrix} z_h \\ z_s \\ z_m \\ z_r \end{pmatrix}^{\top} \cdot \begin{pmatrix} p_h(z|\boldsymbol{x},\boldsymbol{o}) \\ p_s(z|\boldsymbol{x},\boldsymbol{o}) \\ p_m(z|\boldsymbol{x},\boldsymbol{o}) \\ p_r(z|\boldsymbol{x},\boldsymbol{o}) \end{pmatrix} \tag{5.7}$$

where,

$$\begin{aligned} p_h(z|\boldsymbol{x},\boldsymbol{o}) &= \begin{cases} \eta_h \cdot \mathcal{N}(r, \sigma_h^2), & \text{if } 0 \le z \le r_m, \\ 0, & \text{otherwise,} \end{cases} \\ p_s(z|\boldsymbol{x},\boldsymbol{o}) &= \begin{cases} \eta_s \cdot \lambda_s \exp(-\lambda_s z), & \text{if } 0 \le z \le r, \\ 0, & \text{otherwise,} \end{cases} \\ p_m(z|\boldsymbol{x},\boldsymbol{o}) &= \delta(z - r_m), \\ p_r(z|\boldsymbol{x},\boldsymbol{o}) &= \begin{cases} \frac{1}{r_m}, & \text{if } 0 \le z \le r_m, \\ 0, & \text{otherwise,} \end{cases} \end{aligned} \tag{5.8}$$

are four different types of measurement errors due to Gaussian noise, unexpected objects, sensor failure and unexplainable noise respectively. The probability of each noise mode is captured by (z_h, z_s, z_m, z_r) with $z_h + z_s + z_m + z_r = 1$. In Eq. (5.8), r is the nominal range measurement predicted with ray-casting, while r_m represents the maximum range of the sensor. For p_h and p_s, σ_h is the standard deviation for the normal distribution, λ_s is the rate of the exponential distribution, and $\eta_{\{h,s\}}$ are the normalization factors. For p_m, $\delta(\cdot) : \mathbb{R} \mapsto \{0, \infty\}$ is the Dirac delta function.

For the task, we define a stage reward function in the objective function, (4.2), as

follows:

$$r(\boldsymbol{x}_t, \boldsymbol{a}_t, \boldsymbol{x}_{t+1}) = \begin{cases} -5 & \text{if } \boldsymbol{o}(\boldsymbol{x}_{t+1}) = 1, \\ -5 & \text{if } \boldsymbol{a}_t = \mathbf{0} \text{ and } d(\boldsymbol{x}_{t+1}, \boldsymbol{x}_g) > \epsilon_g, \\ 0 & \text{if } \boldsymbol{a}_t = \mathbf{0} \text{ and } d(\boldsymbol{x}_{t+1}, \boldsymbol{x}_g) \leq \epsilon_g, \\ -1 & \text{otherwise,} \end{cases} \tag{5.9}$$

$$d(\boldsymbol{x}_{t+1}, \boldsymbol{x}_g) = \sqrt{\|x_{t+1} - x_g\|^2 + \|y_{t+1} - y_g\|^2},$$

where $\boldsymbol{x}_g$ is the target state. Cost is induced if the robot keeps moving, which encourages fewer actions. Colliding with obstacles, or stopping outside the target location, will introduce additional cost. The accumulation of cost only stops if the robot determines to stop within the target region, which is defined by ϵ_g. Note the orientation of the $\boldsymbol{x}_g$ is ignored in determining task completion.

5.2.2 Experiments

In the simulations, we compare the performance of POMCP++ with RHC, DESPOT (Ye et al., 2017), and POMCPOW (Sunberg and Kochenderfer, 2018). In practice, the idea of certainty equivalence control and separation principle is often extended and applied in the RHC framework to solve the stochastic motion planning problem. In the following, methods of this kind is called RHC for short. Particularly, to solve the proposed problem, one state sample is drawn from the belief at each step, which is regarded as the true state. A*, with inflated heuristics (Likhachev et al., 2004), is then used to find the next action assuming a deterministic motion model. DESPOT (Ye et al., 2017) is the state-of-the-art online POMDP solver designed for discrete systems. The simulation results of DESPOT serve as the benchmark for naively applying discrete POMDP solvers to continuous systems. POMCPOW (Sunberg and Kochenderfer, 2018), also an extension of POMCP, is the most similar to our work. Comparison with POMCPOW shows the importance of the improvements in POMCP++.

There are minor modifications to POMCPOW and DESPOT to adapt to the proposed

Table 5.3: Hyper-parameters for different methods

DESPOT	tree depth $= \{10, \mathbf{50}, 100\}$
	scenario # $= \{30, 100, \mathbf{500}, 1000\}$
	per-step planning time $= \{\mathbf{30}\}$
POMCPOW	$\epsilon_a = \{0.05, 0.1, 0.2, \mathbf{0.3}\}$
	$\alpha = \{0.1, 0.5, \mathbf{1}, 2\}$
	$\kappa = \{\mathbf{0.1}, 0.5, 1, 5\}$
	$N = \{1000, \mathbf{3000}, 5000\}$
POMCP++	$\epsilon_a = \{0.05, \mathbf{0.1}, 0.2, 0.3\}$
	$\epsilon_z = \{\mathbf{-1}, -3, -5, -7\}$
	$K = \{16, 32, \mathbf{64}\}$
	$N = \{\mathbf{3000}\}$

problem. POMCPOW is designed to handle continuous control space in addition to continuous state and measurement spaces. The action space in the problem setup of this work is discrete and finite, consisting of motion primitives. Therefore, we apply the same ϵ-greedy action selection strategy (Alg 4 Line 26) in the POMCPOW framework. For all DESPOT, POMCPOW and POMCP++, we use Anytime A* (Likhachev et al., 2004), with the inflation factor set to infinity, to generate the action sequence as the rollout policy. Although the action sequence could be sub-optimal, it is efficient in finding a feasible solution.

Simulated Environments

In the simulated environments, the cell size of the used occupancy grid maps is set to 0.1m. The action space consists of six different motion primitives, $\mathcal{A}_m = \{0, 0.2\text{m/s}\} \times \{-\frac{\pi}{2}\, 0, \frac{\pi}{2}\text{rad/s}\} \times \{0.5\text{s}\}$. The robot models a differential drive ground vehicle with forward motion only. The range sensor installed on the robot has seven beams covering 2π uniformly. Each beam has a maximum range of $z_{\max} = 0.3$m. ϵ_g in (5.9), the radius of the goal region, is 0.05m.[3] γ, the discount factor, is set to 0.99.

[3]The setup is merely for the ease to convert quantities to metric length units. All related quantities can be scaled proportionally.

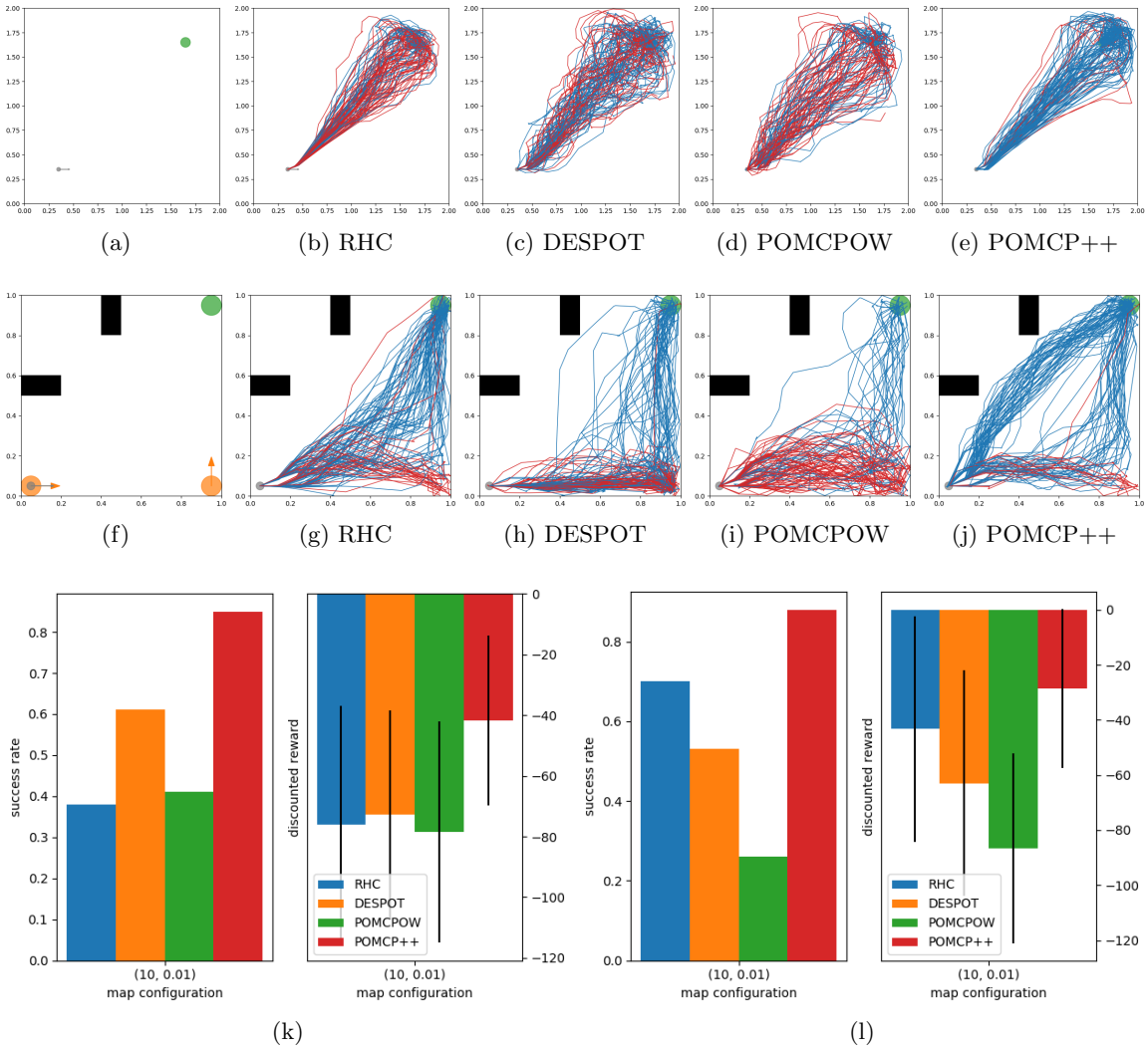
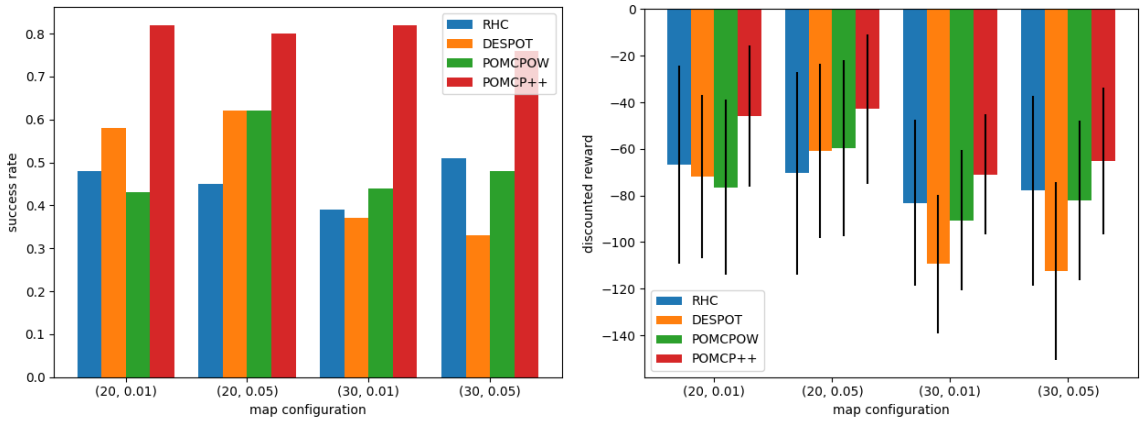


Figure 5.4: (a) and (f) show the initial configurations for the experiments in fixed simulated environements. Occupied cells in the maps are marked with black squares. The green circular region is the target location. The gray circle is the initial state of the robot. The orange circles with the arrows in (f) are the two modes in the initial belief. (b), (c), (d), and (e) are the paths executed with RHC, DESPOT, POMCPOW, and POMCP++ in setup (a) over the 100 experiments. Blue lines are the paths for the successful trials, while red lines are the failed ones. The paths executed by the methods in setup (f) are shown in (g), (h), (i), and (j). (k) and (l) are the success rate and average discounted reward with standard deviation obtained through the 100 experiments for setup (a) and (f) respectively.



(a) (b)

Figure 5.5: For each method and map configuration, the performance of the method is measured with 100 experiments on randomly generated maps. (a) The success rate of different methods. Recall (5.9), a trial is successful when the robot actively chooses to stop around the target location. (b) The average summation of discounted reward with standard deviation. For failure trials, it is assumed that the robot will take feasible actions (with reward -1 per step) onward but never reach the target location.

We first show both qualitative and quantitative results of the methods on two fixed representative maps. In the first setup, Figure 5.4a, we use a 20×20 map with no obstacles. The initial state is known to the robot. In order to reach the target location with high confidence, it is expected that the robot will take advantage of the map boundaries for localization. The second setup, Figure 5.4f, is a 10×10 map, designed to be more challenging in that the initial distribution is ambiguous, consisting of two modes located at the bottom corners. The obstacles in this setup are also set in a way such that the robot is not able to remove the state ambiguity passively, but has to actively approach the obstacles. We run each method 100 times on the two setups to obtain the average performance.

Figure 5.4k and 5.4l show the success rate and the average discounted reward of different methods. More insightful observations can be obtained by plotting the paths as shown in Figure 5.4. In the first setup, the proposed POMCP++ is able to reliably utilize the right upper corner to reduce the uncertainty accumulated during the movement, and then come back to the target location. This capability explains the high success rate of

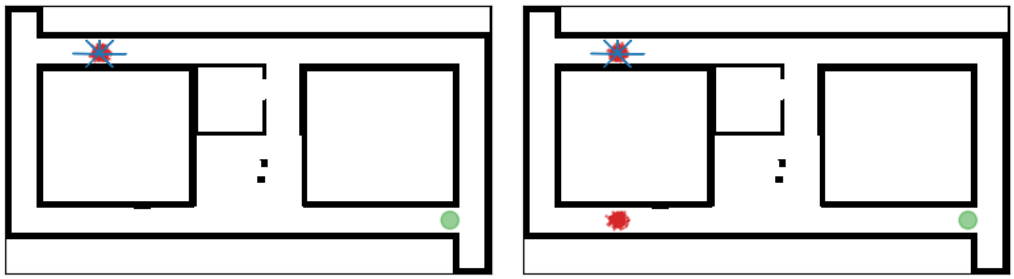
POMCP++. For the second setup, an immediate observation is that all methods try to go along the bottom and right side of the map. This is due to the best-first-search nature of the algorithms. Going along the bottom edge is the fastest way to move one of the belief modes to the target location. Occasionally, the planning algorithms may wrongly choose to stop at this moment. More often, exploring the policies helps the algorithms realize that moving along the right side can further reduce the uncertainty and eventually leave a single belief mode stopping at the target. A more interesting observation is that the proposed POMCP++ is able to find another cluster of paths by moving the robot diagonally across the map. The paths in this cluster are able to remove the ambiguity in the belief within the first few steps by utilizing the obstacles on the left. Although such paths deviate from the "promising" path (going right, then up) of the short horizon, they obtain higher overall reward.

In addition, we evaluate the average performance of the methods using random maps with different configurations. Four map configurations are included by permuting different map size, $\{20 \times 20, 30 \times 30\}$, and obstacle density, $\{0.01,\ 0.05\}$. 100 experiments are performed for each method and map configuration. In each experiment, a random map is generated as per the map configuration. For all experiments, the robot starts with a known initial state.

Figure 5.5 shows the success rate and summation of discounted reward for different methods. POMCP++ is able to achieve much higher success rate and reward, consistently outperforming the other three methods. It should be also noted that RHC performs reasonably well in this experiment, better than POMDP methods, DESPOT and POMCPOW, on some map configurations.

A Hallway Environment

In this section, we compare the performance of POMCP++ with RHC and POMCPOW in a real world hallway environment. The dimension of the map is 153×277, with 0.1m



(a) (b)

Figure 5.6: (a) shows an initial configuration with the initial belief consisting of a single mode. The green circle marks the goal region. The initial belief is represented with red particles. blue lines represents the sparse beams of a range sensor with small footprint. The initial configuration in (b) is more challenging with two modes in the initial belief. The second scenario requires the robot to move out of the corridors in order to collapse the belief into a single mode distribution.

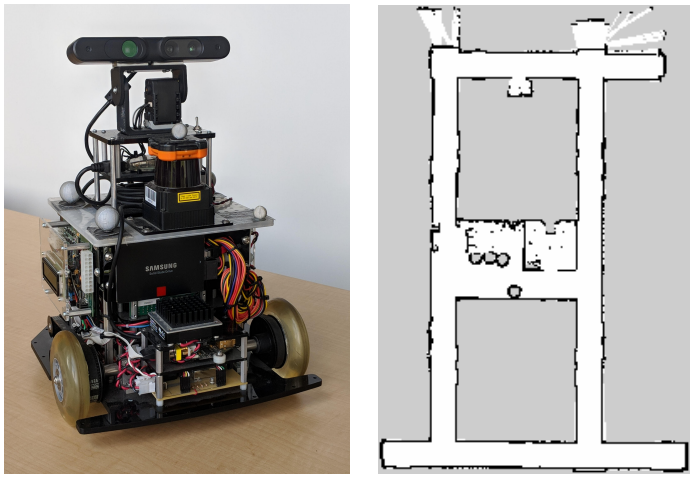
resolution. To adapt to the enlarged map and narrow corridors, the motion primitives are changed to $\mathcal{A} = \{0, 0.5\text{m/s}\} \times \{-\frac{\pi}{6}\, 0, \frac{\pi}{6}\text{rad/s}\} \times \{1.0\text{s}\}$. The radius of the goal region, ϵ_g, is enlarged to 0.5m. Seven Lidar beams are used as sensor measurements covering $\frac{3}{2}\pi$, with $z_{\max} = 1.5\text{m}$. As in the experiments in simulated environments, γ is 0.99. The algorithms are compared in two scenarios with different initial belief shown in Figure 5.6. The hyper-parameters for POMCPOW and POMCP++ remain the same as in Table 5.3.

The comparison with DESPOT is omitted, since the robot fails to reach the goal for all trials with DESPOT. Considering the enlarged environment, experiments for DESPOT are also performed by increasing tree depth to 100 and per-step planning time to 120s. However, there is still no successful trails for either of the initial belief setups. The failure of DESPOT may be caused by using a single simulation episode to estimate the value at a belief node. To elaborate, a naive application of DESPOT to systems with continuous measurement causes the degeneracy of the belief tree. All nodes in the tree, other than the root, are only visited once, *i.e.* when the node is constructed. Therefore, only one simulation episode is available in estimating the value at the nodes, which may be far from adequate in providing an accurate estimate. The inaccurate estimation at the nodes,

eventually, leads to the wrong selection of optimal policy. In the experiments of small simulated environments, simulation episodes are short. It takes around 10 steps to drive the robot to the goal. Simulation episodes obtained with the same action sequence may not differ much. Therefore, using only one simulation episode in estimating the value at the nodes may be sufficient and leads to meaningful actions. However, the number of actions required to reach the goal in the hallway experiments is over 60. The effect of degeneracy for DESPOT becomes more apparent.

Table 5.4 summarizes the performance of RHC, POMCPOW, and POMCP++. For each algorithm and initial belief setup, we run the experiment 50 times. The paths of all trials are shown in Figure 5.9. It is expected that RHC takes fewer actions on average for successful trials, whose planning is based on A*. However, the total traveled distance is similar among the algorithms. This implies that, under POMDP solutions, the robot takes more rotation actions utilizing the local environment to reduce state uncertainty. Figure 5.9 also shows that POMDP solutions are able to maintain the robot in the middle of a narrow corridor, which reduces the chance of collision caused by state uncertainty and motion stochasticity. Comparing POMCPOW and POMCP++, although the performance of the success trails of the two algorithms is similar, POMCP++ is able to achieve higher success rate, especially when the initial belief contains ambiguity. The improvement of POMCP++ is due to the two major modifications of POMCP, namely the new strategy of measurement selection, and the removal of the implicit perfect state assumption at the start of the rollout phase.

We also compare the anytime performance of POMCPOW and POMCP++ in the hallway environment with single mode initial belief setup. Table 5.5 and Figure 5.10 summarize the performance of POMCPOW and POMCP++ under different per-step time constraints. The simulations are conducted on a desktop computer using Intel Core i9-9920X CPU with 12 cores running at 3.5GHz. The maximum per-step planning time is set to 15s, where both POMCPOW and POMCP++ are able to finish 3000 episodes (the default N in Table 5.3).



(a) (b)

Figure 5.7: (a) shows the differential drive ground robot used in the hardware experiments. Seven beams, spanning $\frac{3}{2}\pi$ with $z_{\max} = 1.5$m, are used as sensor measurements. (b) is the map of the hallway environment, built with gmapping (Grisetti et al., 2007). The dimension of the occupancy grid map, 280×186 with 0.1m resolution, is slightly different from the one used in the simulation.

It can be observed in Table 5.5 that POMCP++ is able to consistently outperform POMCPOW in terms of success rate. Table 5.5 and Figure 5.10 also indicate that albeit the anytime feature, sufficient per-step planning time is required for both methods in order to make reasonable decisions.

Hardware Demonstration

To further demonstrate the application of POMCP++, the same experiments in the simulations are carried out with hardware. Figure 5.7 shows the setup of the ground robot and the map of the hallway environment similar to the one used in the simulations.

Figure 5.8 shows paths and the change of belief with the two different initial belief configurations[4]. It should be noted that the hardware experiment only serves as a demonstration that POMCP++ can be applied to actual hardware robots and realistic environments in addition to ideal models in the simulations. In terms of efficiency, POMCP++

[4]See `pomcp++_single_mode_initial_belief.mp4` and `pomcp++_two_mode_initial_belief.mp4` for supplementary videos on the hardware demonstrations.

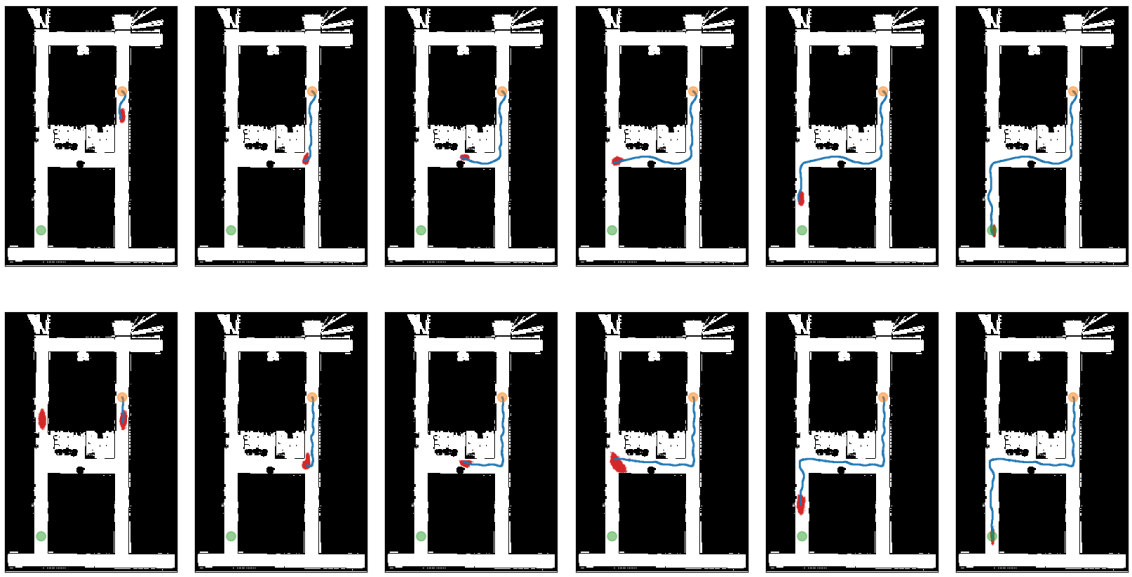
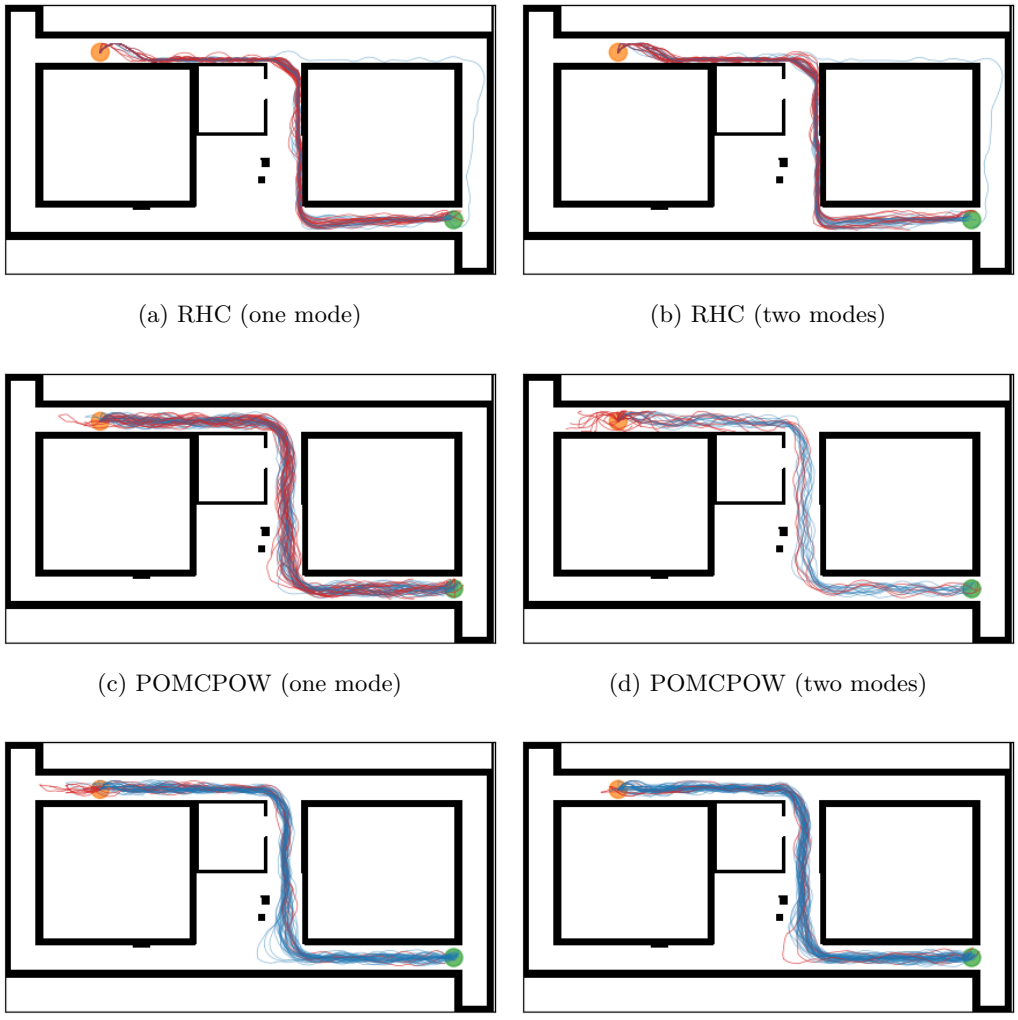


Figure 5.8: Change of belief (red particles) of the robot position in the hardware experiments using POMCP++. The first row is the belief snapshots for the single mode initial belief setup, while the second row are the ones for the two mode initial belief setup. The orange and green patches are the start and target locations. Blue lines are paths estimated with an onboard Lidar odometry.

may not be sufficient for real-time planning purposes, especially with onboard computers. Most of the planning time for each step is spent on constructing rollout policies and forward simulating the system. Using the onboard processor i7-6600U, 2 cores running at 2.6GHz, the planning time for an initial step can take as long as 2min, when the robot is far from the goal. Improving the efficiency of the algorithm is a promising direction of future research.



(a) RHC (one mode)

(b) RHC (two modes)

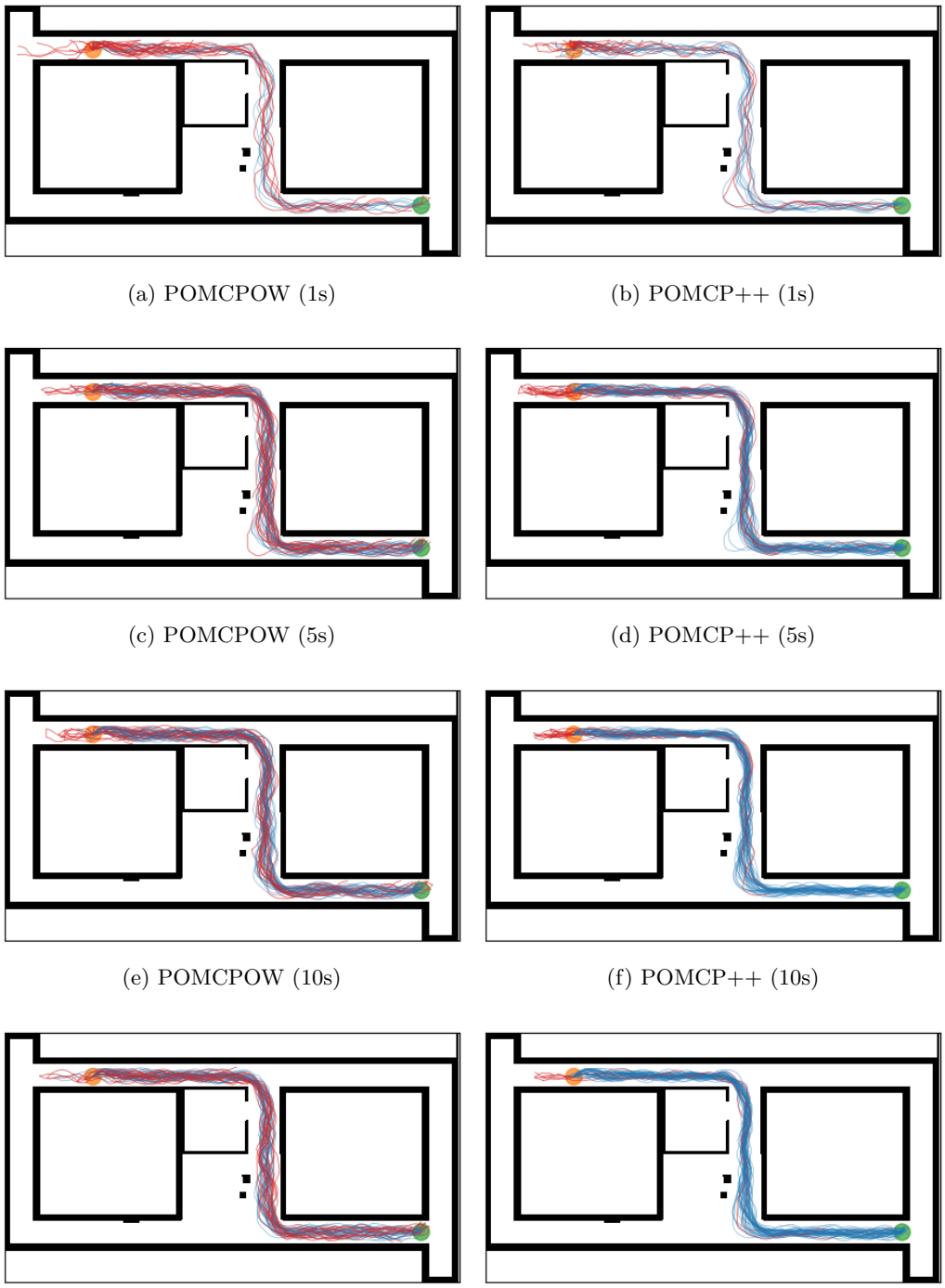
(c) POMCPOW (one mode)

(d) POMCPOW (two modes)

(e) POMCP++ (one mode)

(f) POMCP++ (two modes)

Figure 5.9: (a), (c), and (e) show paths of RHC, POMCPOW, and POMCP++ with single mode initial belief, while (b), (d), and (f) are for the two mode initial belief. The orange and green patches are the start and target locations. Blue and red lines are paths of successful and failure trials. More details of the setup for the two different scenarios can be found in Figure 5.6.



(a) POMCPOW (1s)

(b) POMCP++ (1s)

(c) POMCPOW (5s)

(d) POMCP++ (5s)

(e) POMCPOW (10s)

(f) POMCP++ (10s)

(g) POMCPOW (15s)

(h) POMCP++ (15s)

Figure 5.10: Paths executed by POMCPOW and POMCP++ under different one-step time budgets with the single-model belief set-up. The orange and green patches are the start and target locations. Blue and red lines are paths of successful and failure trials. More details of the setup for the scenario can be found in Figure 5.6.

Table 5.4: Comparison of RHC, POMCPOW, and POMCP++ in a hallway environment

initial belief	method	success/collision/wrong stop/timeout[1][%]	total actions[2]	total distance[2][m]	distance to target[2][m]	final entropy[2]
single mode	RHC	0.66/0.32/0.02/**0.00**	**66.48**	**29.13**	0.45	2.92
	POMCPOW	0.78/0.08/0.10/0.04	74.36	29.64	0.41	2.56
	POMCP++	**0.84**/**0.00**/**0.00**/0.16	75.69	29.51	**0.31**	**2.53**
two modes	RHC	0.58/0.38/0.04/**0.00**	**66.62**	**29.07**	0.44	2.94
	POMCPOW	0.34/0.36/0.30/**0.00**	74.18	29.66	0.39	**2.40**
	POMCP++	**0.90**/**0.00**/**0.02**/0.08	73.38	29.28	**0.26**	2.46

[1] Three causes of a failure trial are collision, wrong stop (over 0.5m away from the target), and timeout (over 100 steps).
[2] These data is averaged over successful trials only.

Table 5.5: Comparison of POMCPOW and POMCP++ with different per-step planning time constraints

per-step time [s]	method	success/collision/wrong stop/timeout[1][%]	total actions[1]	total distance[1][m]	distance to target[1][m]	final entropy[1]
1	POMCPOW	0.18/**0.18**/0.62/**0.02**	**80.33**	**30.00**	0.40	2.52
	POMCP++	**0.22**/0.20/**0.52**/0.06	88.82	30.95	**0.28**	**2.39**
5	POMCPOW	0.64/0.14/0.20/**0.02**	**75.56**	30.10	0.38	**2.46**
	POMCP++	**0.74**/**0.00**/**0.04**/0.22	78.60	**29.70**	**0.28**	2.48
10	POMCPOW	0.68/0.14/0.10/**0.08**	75.24	29.91	0.39	**2.54**
	POMCP++	**0.76**/**0.00**/**0.04**/0.20	**74.39**	**29.31**	**0.22**	2.60
15	POMCPOW	0.68/0.12/0.14/**0.06**	74.35	29.70	0.37	**2.54**
	POMCP++	**0.90**/**0.02**/**0.00**/0.08	**73.58**	**29.41**	**0.28**	2.67

[1] See Table 5.4 for the notes on the metrics.

Chapter 6

Stochastic Motion Planning with Environmental (External) Uncertainty

In Ch. 5, we consider the problem of navigating a car-like robot within a known environment. One feature of the problem is that, the uncertainty of the problem, either motion or measurement noise, is intrinsic to the robot itself. In this chapter, we switch gears and consider the application of autonomous driving. Besides the obvious change in terms of the application context, another major difference is that the dominant uncertainty in autonomous driving originates from the unknown agent driver behaviors. In other words, the major source of uncertainty comes from external environment instead of the robot itself in the application of autonomous driving.

Before considering motion planning, a major challenge is to model the stochastic traffic dynamics. As reviewed in Sec. 2.3, thanks to the recently available traffic datasets, data-driven approaches can now be applied to learn a traffic dynamical model, which could potentially agree with real traffic scenarios better compared to handcrafted traffic models. In learning the traffic dynamics, models with high prediction accuracy is certainly preferred.

It is also important that the model could be applied with common planning algorithms (not limited to the stochastic planning algorithms proposed in this dissertation).

In order to work with common planning frameworks, the learned traffic model should at least share the structure of the motion model in Sec. 1.2, which has already posed minimal assumptions. In terms of structure, the traffic model should be able to make predictions for the next time step based on the current traffic state and immediate motion of the ego vehicle. The former is also known as the Markovian property of a dynamical model. In terms of state representation, the traffic state should consist of the states of all agents, both vehicles and pedestrians, in a local neighborhood. Traffic is also inherently partially observable. Hidden states of agents, often known as "intention", are not directly observable. Modeling the hidden states provides the opportunity to estimate them online and potentially reduce the future prediction uncertainty.

In addition to the traffic dynamical model, we adopt relatively simple models for the measurement and the task that agree with the problem formulation in Sec. 4.1. Given a proper rollout policy, we could therefore apply POMCP++, introduced in Ch. 4, to plan motions for the ego vehicle.

Contributions

In this work, we first propose a recurrent model (Sanchez-Gonzalez et al., 2018) based on GNNs (Battaglia et al., 2018) to model the stochastic traffic dynamics. The proposed traffic model shares the same structure with the motion model in Sec. 1.2, which enables the model to be tightly integrated to common planning frameworks. In terms of learning, the additional model structure and explicit hidden state representation impose regulations. Nevertheless, we show that the proposed traffic model could achieve state-of-the-art prediction accuracy.

In addition to a data-driven traffic model, we apply POMCP++ for motion planning considering uncertainties from traffic dynamics and sensor observations. We show that

the motions planned with POMCP++ agree with human intuition in handcrafted and real traffic scenarios. By comparing to MPDM (Galceran et al., 2017), POMCP++ is able to achieve higher reward in most of the considered scenarios. To the best of our knowledge, this is the first work that applies a full POMDP solution to plan motions for the ego vehicle in general scenarios. Meanwhile, this could also be the first work that tightly couples planning and predictions with a learned traffic model for the autonomous driving problem.

In the following of this chapter, we first introduce the GNN-based traffic dynamical model in Sec. 6.1. Then, we complete the problem formulation by introducing the models for the measurement and task in Sec. 6.2. Finally, in Sec. 6.3, we apply POMCP++ as the motion planning algorithm for the ego vehicle.

6.1 Stochastic Traffic Dynamics Modeling

In this section, we proposed Recurrent Traffic Graph Neural Network (RTGNN) to model the stochastic traffic dynamics. The section starts by introducing the network model as well as the loss function in Sec. 6.1.1. Then, in Sec. 6.1.2, predictions of the proposed model is compared with state-of-the-art solutions both qualitatively and quantitatively on the nuScenes dataset (Caesar et al., 2020). Additional details on the definition of the network and hyperparameters used for training are provided in Appendix B.

6.1.1 Traffic Model Structure

For clarity, we begin by considering the traffic as an autonomous system, *i.e.* the ego vehicle is not distinguished from other agents. Discussions on adapting the model to perform inference conditioned on the ego motion are deferred to later in this section.

We start by modeling the dynamics of a single vehicle as a (dynamically extended) unicycle, which is of the form,

$$\boldsymbol{x}_{t+1} = f_x(\boldsymbol{x}_t, \boldsymbol{u}_t). \tag{6.1}$$

$\boldsymbol{x} = (x, y, \theta, v)^\top \in \mathcal{X}$ is the vehicle state consisting of 2-D position x and y, orientation θ, and speed v. The control input, $\boldsymbol{u} = (a, \omega)^\top \in \mathcal{U}$, includes both linear acceleration, a, and angular velocity, ω. The explicit formulation of (6.1) could be found in LaValle (2006) and Salzmann et al. (2020).

Instead of having a continuous control space $\mathcal{U}$, we define $\mathcal{A} \subseteq \mathcal{U}$, a finite set consisting of predetermined motion primitives, *i.e.* $\mathcal{A} = \{(a_i, \omega_i), i = 0, 1, \ldots, M-1\}$. Meanwhile, we define $\boldsymbol{q} \in \mathbb{R}^{|\mathcal{A}|}$ as a discrete probability distribution over $\mathcal{A}$. In the following, we refer to $\boldsymbol{q}$ as the "intention" of an agent, which determines the probability of taking each motion primitive at a single time step. Combining the vehicle state and intention, the agent state is defined as $\boldsymbol{y}^\top = (\boldsymbol{x}^\top, \boldsymbol{q}^\top)$. In this work, we assume uncertainty of an agent only comes from its intention, or equivalently, the unknown motion primitive to be executed.

Presumably, the intention of an agent is affected by its neighbors. We define $\boldsymbol{X}$, $\boldsymbol{Q}$, and $\boldsymbol{Y}$ as the aggregation of vehicle states, intentions, and agents states of all agents (including the ego) in a local traffic. We assume the dynamics of the intentions is in the form of,

$$\boldsymbol{Q}_{t+1} = f_q(\boldsymbol{Y}_t, \boldsymbol{m}_t) = f_q(\boldsymbol{X}_t, \boldsymbol{Q}_t, \boldsymbol{m}_t), \tag{6.2}$$

where $\boldsymbol{m}$ represents the local environment, reflecting components such as drivable areas, lane geometry, *etc.* The form of (6.2) implicitly assumes deterministic transition of intention. However, it does not imply deterministic transition of agent motion, but deterministic transition of motion distribution.

To reduce the complexity of the traffic model, we only consider vehicles and assume a deterministic first-order motion model for pedestrians. Equivalently, we could keep using (6.1) to model the motion of pedestrians, but with a fixed intention concentrating all probability mass on the motion primitive $(0, 0)$, *i.e.* the motion primitive with zero linear acceleration and angular velocity.

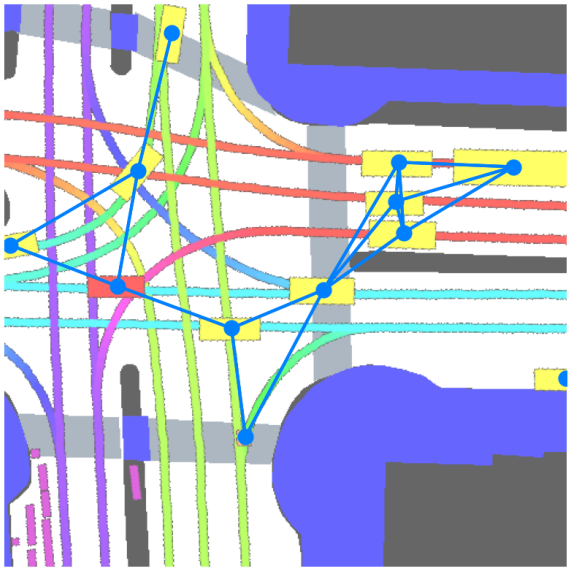


Figure 6.1: An example graph constructed for a local traffic scene. Blue dots represent agents in the scene. Blue dots on the yellow and red boxes represent vehicles. Standalone blue dots are pedestrians. Directed edges (blue lines) between agents are introduced based on spatial proximity. Specially, since we do not learn the behavior of pedestrians, directed edges with pedestrians as the target nodes are removed. Implicitly, this means the motion of pedestrians is not affected by other agents. On the background shows the semantic map where lane directions are color coded as suggested by Cui et al. (2019).

Modeling Intention Dynamics with a GNN

As mentioned earlier, f_x, modeled as a unicycle in this work, is relatively straightforward to specify. However, f_q involves human factor, making it hard to be specified in any simple parametric form. Therefore, we structure f_q as a deep neural network and learn the parameters through data. In this work, we use a GNN for the intention dynamics, f_q. The major motivation is that GNNs are able to handle graphs with various number nodes and edges, as is the case in the traffic scenes where there are multiple agents with complex interactions.

As shown in Figure 6.1 and 6.2, each node in the graph represents an agent in the local traffic, either a vehicle or a pedestrian, while edges are introduced based on spatial proximity. The feature of a node/agent includes vehicle state $\boldsymbol{x}^a = (x^a, y^a, \theta^a, v^a)$, future vehicle states at the next step under different motion primitives $\boldsymbol{w}^a$, intention $\boldsymbol{q}^a$, and a local agent-centric view of semantic map $\boldsymbol{m}^a$ similar to what is shown in Figure 6.1. The

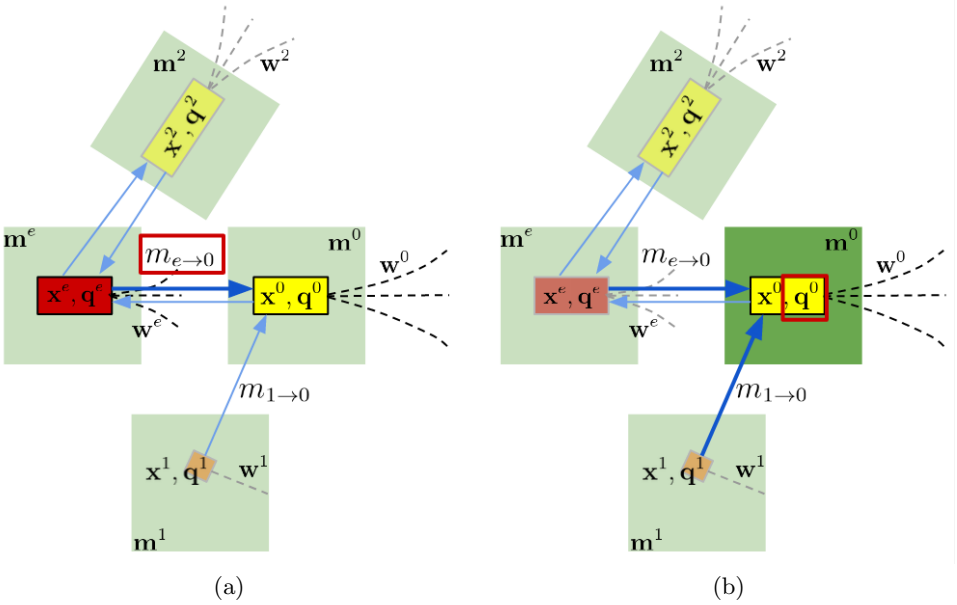


Figure 6.2: Schematic figures showing major steps in GNN for intention update. (a) shows the message computation along an edge. (b) shows the intention update at a node. Red, yellow, and orange boxes represent nodes of the ego vehicle, agent vehicles, and pedestrians. Curves next to each player are the future vehicle states under different motion primitives. Green patches represent the local agent-centric semantic maps. In the figures, the variables to be computed are marked with red boxes. Variables involved in the computation are highlighted while unrelated variables are grayed out. See Sec. 6.1.1 for the definitions of variables and Alg. 5 for the detailed steps.

task of the GNN is to update the intention of agents utilizing the graph structure and the node features. Alg. 5 shows the detailed steps in updating agent intention, which follows the general procedure of a message passing GNN. Within each major iteration, messages to each agent are constructed, followed by update of agent intention. Figure 6.2 delineates the two steps with schematic diagrams.

We make a few remarks on Alg. 5. First, recalling (6.1), the motion primitive for a vehicle is 2-D, consisting of linear acceleration and angular velocity. Therefore, it might be natural to combine the future states, $\boldsymbol{w}$, and the intention, $\boldsymbol{q}$, into a multi-channel 2-D "image". The resulted 3-D tensor can be encoded with a Convolutional Neural Network (CNN) (Line 3). A major benefits of using CNN, compared to using a Multi-Layer Perceptron (MLP), is to reduce the number of parameters in the network. Second, at constructing

Algorithm 5: GNN for intention dynamics

Input: Traffic Graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$.
Output: Intention of all agents Q.

1 **for** k *from* 0 *to* $K-1$ **do** *// K defines iterations of message passing.*
  // Construct the message to each node.
2 **foreach** (b, a) *in* $\mathcal{E}$ **do**
    // T_g^a converts data from global frame to local frame of agent a.
3    $c^a \leftarrow \text{CNN}_w(T_g^a \boldsymbol{w}^a, \boldsymbol{q}^a)$
4    $c^b \leftarrow \text{CNN}_w(T_g^a \boldsymbol{w}^b, \boldsymbol{q}^b)$
5    $m_{b\to a} \leftarrow \text{MLP}_m(T_g^a \boldsymbol{x}^a, c^a, T_g^a \boldsymbol{x}^b, c^b)$
6 **end**
  // Update the intention of each node.
7 **foreach** a *in* $\mathcal{V}$ **do**
8    $m_a \leftarrow \text{Aggr}(\{m_{b\to a} : (b, a) \in \mathcal{E}\})$ *// Element-wise max is used as Aggr.*
9    $c^a \leftarrow \text{CNN}_w(T_g^a \boldsymbol{w}^a, \boldsymbol{q}^a)$
10    $p^a \leftarrow \text{CNN}_m(T_g^a \boldsymbol{m}^a)$
11    $\boldsymbol{q}^a \leftarrow \text{MLP}_q(v^a, c^a, p^a, m^a)$
12 **end**
13 **end**
14 **return** $\{\boldsymbol{q}^a : a \in \mathcal{V}\}$

the messages (Line 5, 9, and 10), data is converted to the target agent frame whenever applicable in order to preserve the symmetry of nodes in the graph. Finally, Alg. 5 only generates the predicted intention for the next step. For multi-step predictions, the model can be applied recurrently using the predicted intention and sampled vehicle states as the input for the next step.

Inference Conditioned on the Ego Motion

Until this point, we have considered the traffic as an autonomous system. However, as discussed in the introduction, it is important to infer the motion of agents conditioned on the planned motion of the ego. The proposed GNN model can be easily adapted for this purpose.

At graph construction, we could remove the directed edges to the ego node, while keeping the intention of the ego vehicle fixed based on the planned control input. Therefore, the ego vehicle would only broadcast its current state and intention to neighbor agents but

would not be affected by their reactions. Effectively, the modifications enable the network to perform inference conditioned on the ego motion.

As a result, f_x in (6.1) and f_q in (6.2), together, model the controlled stochastic traffic dynamics,

$$\boldsymbol{Y}_{t+1} = f(\boldsymbol{Y}_t, \boldsymbol{u}_t^e, \boldsymbol{n}_t), \tag{6.3}$$

which satisfies the assumptions of the general motion model in (1.1). In (6.3), $\boldsymbol{u}^e$ is the control of the ego vehicle. $\boldsymbol{n}$ is the random variable determining the controls to be executed by the agents following their intention. Note we consider the local map $\boldsymbol{m}$ as a time-varying constant in the traffic model. Therefore, $\boldsymbol{m}$ is not explicitly listed as inputs in (6.3).

Training Loss

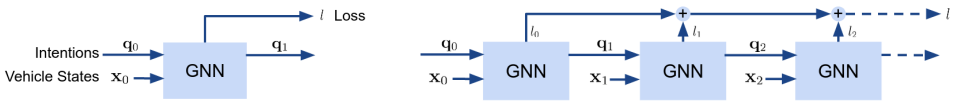
The parameters in f_q are learned by minimizing the cross entropy between the predicted intention $\boldsymbol{q}_t^a$ for agent a and the target (ground truth) intention $\boldsymbol{q}_t^{a*}$ for the same agent. Unfortunately, the ground truth intention of agents cannot be known from a dataset. Instead, we approximate the target intention through the known vehicle states at consecutive time steps. One simple way is to define the target intention $\boldsymbol{q}_t^{a*}$ as a *one-hot* vector, *i.e.*,

$$\boldsymbol{q}_t^{a*}(i) = \begin{cases} 1, \text{ if } i = \arg\min\limits_i \|\boldsymbol{x}_{t+1}^{a*} - f_x(\boldsymbol{x}_t^{a*}, \boldsymbol{u}_i)\|, \\ 0, \text{ otherwise.} \end{cases} \tag{6.4}$$

where $\boldsymbol{x}_t^{a*}$ and $\boldsymbol{x}_{t+1}^{a*}$ are the ground truth states for agent a at consecutive time steps[1]. In practice, we find that it can easily lead to overfitting by defining the target intention as (6.4) since the exact same traffic state hardly shows up repeatedly. To promote generalization, the target intention in (6.4) is "smoothed" to a Gaussian distribution,

$$\boldsymbol{q}_t^{a*}(i) = \frac{1}{\eta} \exp\left(-\frac{1}{2}\|\boldsymbol{x}_{t+1}^{a*} - f_x(\boldsymbol{x}_t^{a*}, \boldsymbol{u}_i)\|_\Sigma^2\right), \tag{6.5}$$

[1]The velocity of agents is obtained through numerical differentiation.



(a) (b)

Figure 6.3: (a) The block structure for one-step learning corresponding to the loss function in (6.6). (b) The block structure for multi-step learning corresponding to the loss function in (6.7) where consecutive traffic snapshots are combined into a sequence in order to improve the consistency of intention transition.

where η serves as a normalization factor. $\Sigma = \text{diag}\big(\sigma_x^2, \sigma_y^2, \sigma_\theta^2, \sigma_v^2\big)$ represents the covariance matrix modeling the empirical uncertainty for the difference between the true and predicted vehicle states after one time step.

With the target intention defined, we could use cross entropy as the loss function shown as the following,

$$l = \sum_n \sum_a \sum_i -\boldsymbol{q}_{t,n}^{a*}(i) \log(\boldsymbol{q}_{t,n}^{a}(i)). \tag{6.6}$$

In (6.6), i indexes the elements in $\boldsymbol{q}$, a indexes agents in a traffic snapshot, and n indexes snapshots in the dataset. We may also chain together consecutive snapshots, as the following, to ensure consistent intention transition over a longer period.

$$l = \sum_n \sum_t \sum_a \sum_i -\boldsymbol{q}_{t,n}^{a*}(i) \log(\boldsymbol{q}_{t,n}^{a}(i)), \tag{6.7}$$

where t indexes different steps in a sequence, while n, instead of indexing snapshots, indexes snapshot sequences in the dataset. In a sequence from the real dataset, an agent may appear or disappear at any intermediate step of the total t steps, making the agent have a shorter lifetime than t. Thanks to the Markovian property of RTGNN, data for agents with a shorter lifetime can still be used in (6.7) where loss of an agent is only considered when it is within the local region.

Figure 6.3 shows the block structures corresponding to the loss functions in (6.6) and (6.7). It should be highlighted that the chained block structure of RTGNN, in Fig-

ure 6.3b, naturally avoids a few common issues in training recurrent neural networks. First, the hidden states, named as intentions in this work, are explicitly defined as discrete probability distribution and therefore remain well-behaved numerically. Second, gradients of parameters in the initial blocks do not disappear because of the existence of "gradient highway" resulting from combining training losses from multiple steps. Such features of RTGNN shares similarity with the Long Short Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997). Compared to LSTM, RTGNN is different in a more explicit and interpretable definition of the hidden states and a special internal block structure adapted to different traffic scenes.

6.1.2 Experiments

In this work, we use the nuScenes dataset (Caesar et al., 2020) to train and evaluate the proposed traffic model. Specially, only the data collected at Boston Seaport[2] is used. Each training sample consists of four-second traffic, which includes eight time steps given the data is labeled at 2Hz. Built upon the official split (Caesar et al., 2020), the training set includes 4055 non-overlapped samples/sequences, while the validation set includes 352 samples. Although the length of training samples is fixed, RTGNN can be applied to make predictions of arbitrary length. The number of testing samples depends on the required prediction length. For four-second prediction, there exists 915 non-overlapped testing samples. We train RTGNN as an autonomous system, *i.e.* treating the ego as a nominal agent vehicle. For experiments on making predictions conditioned on ego motions, the input graph representation is modified as introduced in Sec. 6.1.1, while the parameters for the network remain the same. More details on the network implementation and training are provided in Appendix B.

In the following, we compare the performance of RTGNN with Trajectron++ (Salzmann et al., 2020), a recently open-sourced work on traffic prediction. Within the existing

[2]The rest of the data in nuScenes is collected in Singapore, where vehicles drive on the left side of the roads. The concern is that mixing training data of different driving conventions may have negative impact on the outcome.

literature on this subject, Trajectron++ might be the most similar to this work in terms of the properties of the traffic model, which has also been evaluated on the nuScenes dataset. Trajectron++ is able to predict trajectories for all agents in a scene jointly. The predictions could be made conditioned on a future trajectory of the ego vehicle. On the difference, Trajectron++ is not designed to be used as a Markovian model. It takes historic trajectories of agents and predicts their future for multiple time steps. Meanwhile, Trajectron++ does not explicitly model the hidden states of the agents as in this work, it adopts a CVAE framework (Sohn et al., 2015) which introduces discrete latent variables to encode high-level behaviors.

We make a few minor tweaks on the implementation of Trajectron++[3]. First, in the existing implementation of Trajectron++, future data of an agent are used to compute velocity and acceleration at the exact present time[4]. However, the assumption is invalid in practice due to the unavailability of the future data. Instead, we use delayed velocity and acceleration computed with only past data. Second, parked vehicles are also included in the scene. There are chances that parked vehicles merge into the traffic which may cause collisions with the ego if not anticipated. With the above two modifications, we retrain Trajectron++ for twenty epochs, among which the model with the least validation error is used for testing. In training Trajectron++, we use training samples with four seconds of future traffic data as in training RTGNN. In testing RTGNN, one-second past data is used to compute agent velocity and acceleration at the present. For fair comparison, past traffic data of the same length are used as input for Trajectron++.

Qualitative Results

We compare sampled and maximum likelihood predictions of Trajectron++ and RTGNN qualitatively in different scenarios, where we consider the traffic as an autonomous system. For RTGNN, a sampled prediction is generated by sampling controls of an agent according

[3] `https://github.com/StanfordASL/Trajectron-plus-plus`
[4] `https://github.com/StanfordASL/Trajectron-plus-plus/issues/40`

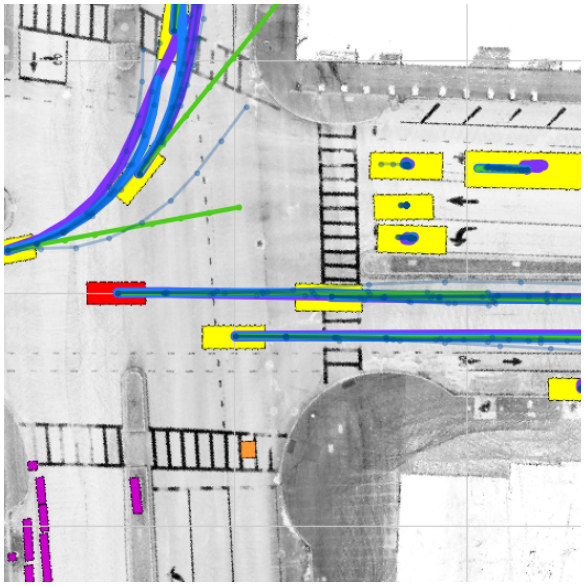


Figure 6.4: An example prediction of RTGNN at an intersection. The red and yellow are the ego and agent vehicles. The orange squares represent pedestrians. Purple lines are the ground truth paths. Green lines are predictions of a constant velocity model, serving as a baseline. Blue lines are the maximum likelihood predictions, while gray lines are five sampled predictions.

to its intention. The maximum likelihood predictions are produced in a similar way. The difference is that the control with the maximum probability is selected at each time step. Note that a trajectory obtained by sequentially selecting maximum likelihood controls may not be the overall maximum likelihood trajectory, but only serves as an approximation. However, it would be computationally intractable to generate the true maximum likelihood predictions with the proposed traffic model in this work. Figure 6.4 shows example predictions of RTGNN along with the legends for different types of predictions[5].

Figure 6.5 shows the prediction results of both methods in four traffic scenarios. It could be observed that overall RTGNN is able to produce more sensible predictions. Compared to Trajectron++, cases are rarer for RTGNN where the sampled or maximum likelihood predictions are infeasible, *i.e.* violate the traffic rules. Meanwhile, RTGNN generates more stable predictions for parked vehicles. It should also be noted that, even though there

[5]See `RTGNN.mp4` for a compilation of RTGNN predictions for different traffic scenes

Table 6.1: Comparison of six-second predictions of the ego vehicle[1]

method	ADE	FDE	$\min_5$ADE	$\min_5$FDE
Const. Vel.	4.61	11.21		
MTP (Cui et al., 2019)	4.41	10.26	2.22	4.83
MultiPath (Chai et al., 2019)	4.43	10.16	**1.78**	**3.62**
CoverNet (Phan-Minh et al., 2020)	5.41	11.36	2.62	
MHA (Messaoud et al., 2020)	**3.69**	**8.57**	1.81	3.72
Const. Vel.[2]	4.39	10.54		
RTGNN	3.74	8.90	2.21	4.51

[1] The results of other methods (Cui et al., 2019; Chai et al., 2019; Phan-Minh et al., 2020; Messaoud et al., 2020) are summarized by Messaoud et al. (2020), and are reproduced here for comparison.
[2] Since we are only able to use the testing set collected at "Boston Seaport", we report the error of a constant velocity model on our testing set for normalization purpose.

is no explicit modeling of high-level behaviors, RTGNN is able to produce predictions of multi-modality, such as lane-change versus lane-keep and right-turn versus forward-move as shown in Figure 6.5.

Figure 6.6 compares the predictions of RTGNN with or without conditioning on the ego motion. For unconditional predictions, the traffic is considered as an autonomous system where RTGNN makes predictions for all vehicles in the scene. For conditional predictions, the traffic is considered as an controlled system, where the ego motion is fixed to its ground truth. In Figure 6.6, different behaviors of agent vehicles can be observed depending on the motion of the ego. We would like to note that such examples are rare in the dataset. In order to observe the difference, strong interactions should exist between the ego and agent vehicles. Meanwhile, there should be significant difference between the predictions for the ego vehicle and the corresponding ground truth. As a result, predictions for agents under conditional and unconditional inferences are similar in most cases. This observation is aligned with the results presented in Tolstaya et al. (2021).

Quantitative Results

To quantitatively measure the prediction performance, we adopt some common metrics in the literature. The definitions of the metrics are listed below, where T represents the prediction horizon. For sanity, the following definitions consider only one agent. In practice, the error of all agents in the testing set should be averaged.

- Average Displacement Error (ADE),

$$\text{ADE} = \frac{1}{T}\sum_{t=0}^{T-1} \|(x_t, y_t) - (x_t^*, y_t^*)\|_2, \tag{6.8}$$

  averages the displacement over the entire predicted path.

- Final Displacement Error (FDE),

$$\text{FDE} = \|(x_T, y_T) - (x_T^*, y_T^*)\|_2, \tag{6.9}$$

  only considers the displacement at the final step, where the displacement is often the largest over the prediction horizon.

- $\min_k$ADE and $\min_k$FDE are for sampled predictions. The metrics report the minimum ADE and FDE over k sampled predictions. In this work, we consider the minimum error over five samples, *i.e.* $k = 5$.

Table 6.2 compares RTGNN with Trajectron++ and a baseline constant velocity model in terms of the above metrics. It could be observed that RTGNN is able to consistently produce more accurate maximum likelihood and sampled predictions with prediction horizons from one to four seconds. Note compared to the results reported by Salzmann et al. (2020), the FDE of Trajectron++ in Table 6.2 shifts backwards in time approximately by one second. This might be related to the fix of using past, instead of future, data to compute velocities and accelerations of the vehicles, mentioned at the beginning of this section.

In addition, we also compare the accuracy of RTGNN with other state-of-the-art methods using the setup specified by the nuScenes prediction challenge[6], which requires the prediction for the future six seconds of the ego given the traffic data for the past two seconds. We make a few remarks on applying RTGNN with this setup. First, RTGNN makes predictions for all players in the scene by considering the traffic as an autonomous system, whereas only the prediction of the ego vehicle is used for error evaluation. Second, since we are not able to use the full testing set[7], we report the error of a constant velocity model in Table 6.1, alongside the prediction error of RTGNN, for normalization purpose. As shown in Table 6.1, despite the regulations resulting from model structure and state representation, RTGNN could achieve similar prediction accuracy as the state-of-the-art methods in terms of displacement errors.

[6] `https://www.nuscenes.org/prediction?externalData=all&mapData=all&modalities=Any`

[7] We only use the data collected at "Boston Seaport" which has the same driving convention as the data used for training.

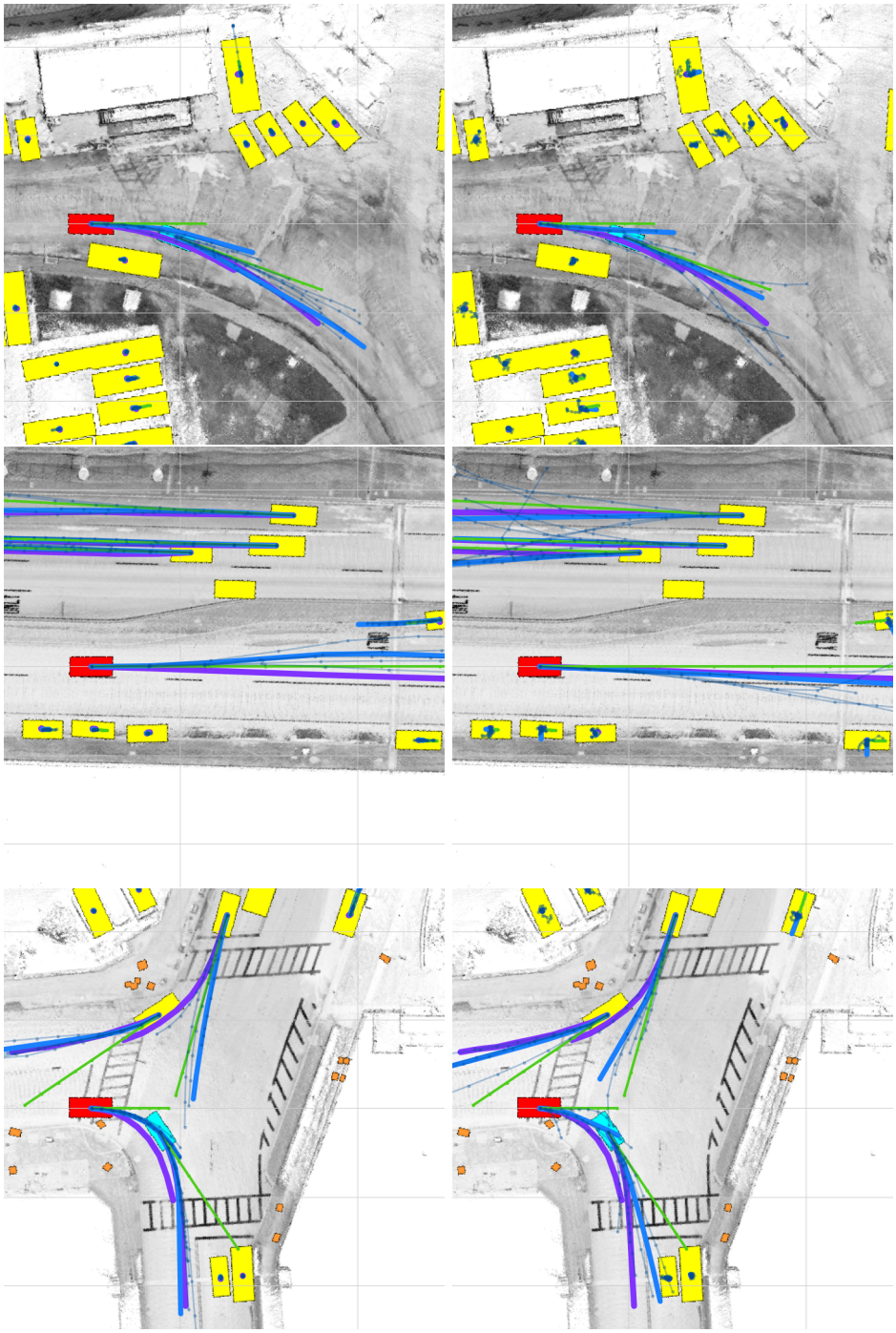


Figure 6.5: Comparisons between (left) RTGNN and (right) Trajectron++ in different scenarios, including (first row) traffic circles, (second row) parking lots, (third row) straight roads and (last row) intersections. Note in the last two scenarios, RTGNN demonstrates predictions of multi-modality, (third row) lane keep v.s. lane change, (last row) forward move v.s. right turn. See Figure 6.4 for legends.

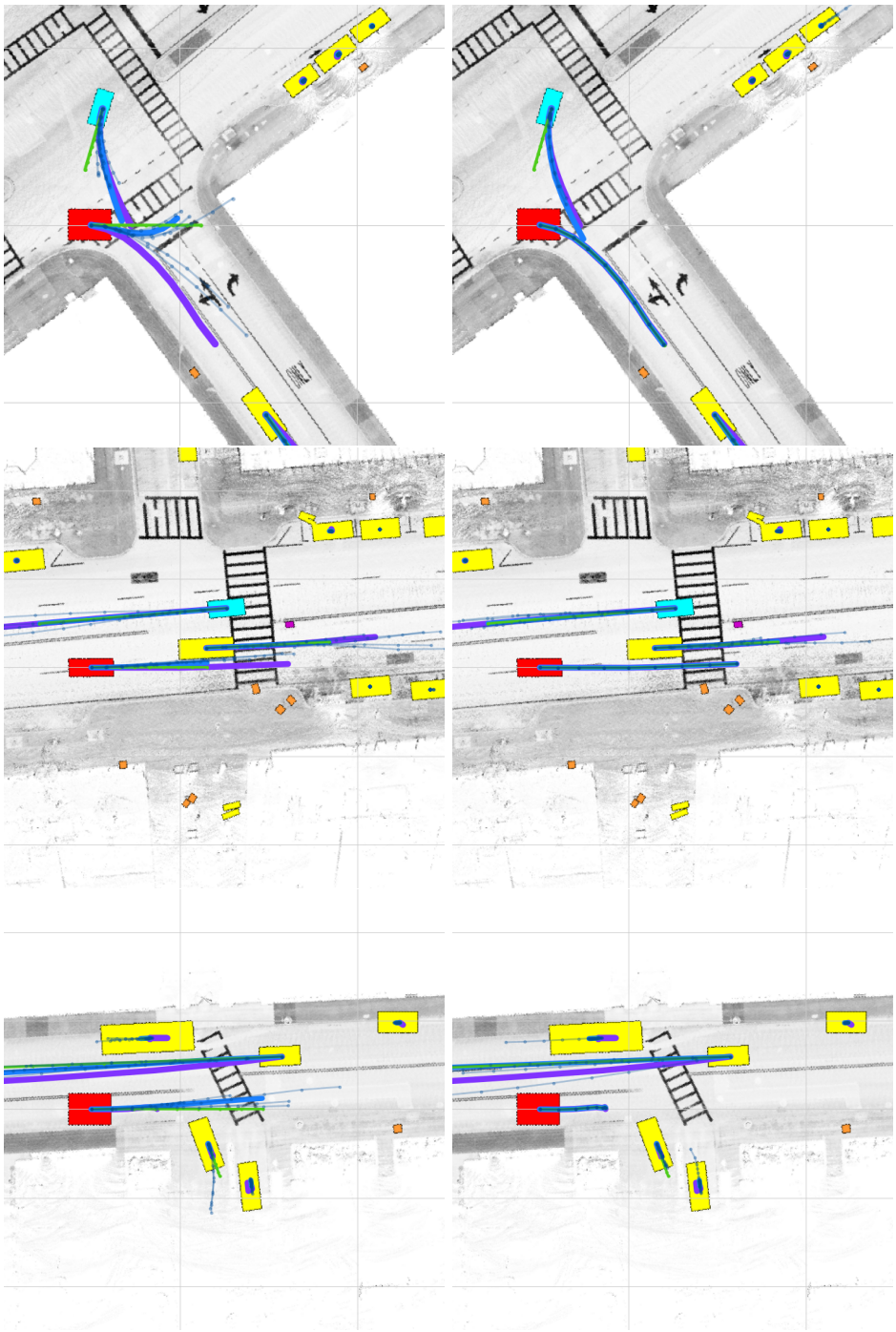


Figure 6.6: Comparisons between (left) unconditional and (right) conditional inferences of RTGNN. (Top) given that the ego is turning right instead of going back to the same lane, the agent at the intersection is able to turn left faster. (Middle) given that the ego is changing lane with a relatively high speed, the leading agent may defer lane changing to a later time step. (Bottom) given that the ego yields, the agent waiting at the intersection may start turning. See Figure 6.4 for legends.

Table 6.2: Comparison of predictions for all agents in a local traffic[1]

Cases[2]	Methods	1s		2s		3s		4s	
		M.L.P.[3]	S.P.[4]	M.L.P.	S.P.	M.L.P.	S.P.	M.L.P.	S.P.
Unconditional Inference	Const. Vel.	0.28/0.39	-	0.57/1.05	-	0.95/1.98	-	1.41/3.16	-
	Trajectron++	**0.26/0.37**	0.19/0.25	0.55/1.02	0.41/0.71	0.93/1.94	0.70/1.41	1.39/3.11	1.08/2.35
	RTGNN	**0.26/0.37**	**0.18/0.23**	**0.50/0.89**	**0.32/0.49**	**0.82/1.65**	**0.50/0.88**	**1.19/2.59**	**0.70/1.32**
Conditional Inference	Const. Vel.	0.25/0.34	-	0.50/0.92	-	0.83/1.74	-	1.22/2.73	-
	Trajectron++	0.25/0.35	0.20/0.27	0.52/0.97	0.42/0.75	0.87/1.83	0.70/1.45	1.27/2.87	1.03/2.30
	RTGNN	**0.23/0.31**	**0.16/0.20**	**0.44/0.78**	**0.28/0.43**	**0.71/1.44**	**0.43/0.76**	**1.01/2.20**	**0.60/1.14**

[1] There are 2714, 1505, 1111, and 915 non-overlap testing sequences for prediction horizon 1s, 2s, 3s, and 4s respectively.
[2] In unconditional and conditional inferences, traffic is considered as an autonomous system and controlled system respectively. In conditional inference, traffic is considered as an controlled system, while the ego motion is set to the corresponding ground truth.
[3] M.L.P. stands for maximum likelihood predictions. The data in the corresponding entries are in the form of ADE/FDE in meters.
[4] S.P. stands for sampled predictions. The data in the corresponding entries are in the form of min_5ADE/$\min_5$FDE in meters.

6.2 Measurement and Task Models

In this section, we provide the measurement model and the objective function in order to complete the problem formulation in Sec. 4.1 for the autonomous driving application.

Instead of using the raw sensor models, we assume an abstract measurement model for the perception system, where the ego vehicle directly measures the vehicle state of itself and 2-D poses of other agents. More specifically, the measurement model is of the form,

$$\begin{aligned} \boldsymbol{z}_t = \left(\boldsymbol{z}_t^{e\top}, \boldsymbol{z}_t^{a\top}\right) = h(Y_t), \quad a = 0, 1, \ldots, N-1, \\ \boldsymbol{z}_t^e = (x_t^e, y_t^e, \theta_t^e, v_t^e)^\top, \\ \boldsymbol{z}_t^a = (x_t^a, y_t^a, \theta_t^a) \circ \boldsymbol{n}, \quad \boldsymbol{n} \sim \mathcal{N}(\mathbf{0}, \Sigma_n), \end{aligned} \tag{6.10}$$

where $\circ$ represents the composition operation for $SE(2)$. In (6.10), we assume the ego vehicle could measure the vehicle state, including both pose and velocity, of itself noise-free. Meanwhile, the ego vehicle could measure the poses of other agents with Gaussian noise. Specially, the measurement noise is specified in the local frame of agents so that the model could capture different noise characteristics for the longitudinal, latitudinal directions and the orientation. For a reasonable perception system on self-driving vehicles, measurement noise along the latitudinal direction and orientation is expected to be small because of the constraints from the structured environment. However, the measurement noise along the longitudinal direction could be large.

In this work, we consider the task of autonomous driving as navigating the ego vehicle to a local goal waypoint. In the objective function, we try to reflect common objectives in the autonomous driving application, such as increasing ride comfort and reducing travel time. More specifically, the stage reward function is in the form of,

$$r(\boldsymbol{Y}_t, \boldsymbol{u}_t^e, \boldsymbol{Y}_{t+1}) = r_a(a_t^e) + r_\omega(\omega_t^e) + r_v\left(v_{t+1}^e\right) + r_d(v_t^e, a_t^e) + r_t + r_c(\boldsymbol{X}_{t+1}). \tag{6.11}$$

The first two terms $r_a(\cdot) : \mathbb{R} \mapsto \mathbb{R}$ and $r_\omega(\cdot) : \mathbb{R} \mapsto \mathbb{R}$ defined as,

$$r_a(a) = \begin{cases} 0 & \text{if } a = 0, \\ -2.5 & \text{otherwise,} \end{cases} \quad \text{and,} \quad r_\omega(\omega) = \begin{cases} 0 & \text{if } \omega = 0, \\ -2.5 & \text{otherwise,} \end{cases} \tag{6.12}$$

are functions of the executed motion primitive and therefore related to the ride comfort. $r_v(\cdot) : \mathbb{R} \mapsto \mathbb{R}$ motivates the ego vehicle to maintain a reasonable speed,

$$r_v(v) = \begin{cases} 0 & \text{if } 10 \leq v \leq 20, \\ -5 & \text{otherwise.} \end{cases} \tag{6.13}$$

$r_d(\cdot) : \mathbb{R} \times \mathcal{A} \mapsto \mathbb{R}$ defined as,

$$r_d(v, a) = -v\tau - \frac{1}{2} a \cdot \tau^2, \tag{6.14}$$

and the scalar $r_t = 10$ tries to reduce the traveled distance and the number of steps taken to get to the waypoint. In (6.14), τ is the fixed duration for each discrete time step. $r_c(\cdot) : \mathcal{X}^{N+1} \mapsto \mathcal{R}$ penalizes collisions between the ego and other agents,

$$r_c(\boldsymbol{X}_t) = \begin{cases} 0 & \text{if the ego vehicle does not collide with other agents,} \\ -1000 & \text{otherwise.} \end{cases} \tag{6.15}$$

In (6.11), terms related to collision with environment or failing in completing the task are not included. At the planning phase, infeasible motion primitives at each step, *i.e.* motion primitives that collide with environment or not result in a feasible trajectory to the goal, can be detected and ignored. This is possible since we assume that the ego state can be measured without noise. Therefore, we could avoid additional terms in the reward function by only considering feasible motions in planning.

Finally, we note that the objective function, (6.11), is over-simplified. Although it is

sufficient to demonstrate the advantage of stochastic motion planning in the autonomous driving application, which is the major purpose of this work, it may not lead to human-like driving behavior. Developing objective functions motivating human-like behaviors (Zeng et al., 2019), or inverse optimal control in general (Ziebart et al., 2008), is an active research area and is beyond the scope of this work.

6.3 Motion Planning for the Ego Vehicle with POMCP++

In this section, we apply POMCP++, introduced in Ch. 4, to plan motions of the ego vehicle within the stochastic traffic. One critical step before the application of POMCP++ is to figure out a suitable rollout policy. In Sec. 6.3.1, we extend hybrid A* (Dolgov et al., 2008) to plan motions in structured environments, which could then be used as a rollout policy to estimate the reward-to-go. In Sec. 6.3.2, we study the behavior of the ego vehicle under POMCP++ with various handcrafted and real traffic scenarios. Meanwhile, we compare the performance of POMCP++ with MPDM (Galceran et al., 2017) in terms of the objective function.

6.3.1 A Hybrid A* Rollout Policy

The hybrid A* algorithm, first introduced by Dolgov et al. (2008), is used to plan motions for Junior (name of the vehicle used by the Stanford team in the DARPA Urban Challenge). Compared to common search-based algorithms solving for trajectories, hybrid A* associates a continuous state, instead of discretized states, to each cell in the graph. When two continuous states fall onto the same cell, one dominates the other even if they are not exactly the same state. As a result, hybrid A* is not guaranteed to find the optimal solution since it violates the principle of optimality. However, as a major advantage of hybrid A*, it is able to directly produce a dynamically feasible trajectory without further refining.

As reported by Dolgov et al. (2008), hybrid A* is used for planning in the unstructured environments, such as parking lots and U-turns. The challenge in applying hybrid A* to

Algorithm 6: Creating the Heuristic Function for Hybrid A*

Input: The graph conformal to the road structure $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$.
The goal node, n_g.
Output: The heuristic function $d(\cdot) : \mathcal{N} \mapsto \mathbb{R}$.
The path retrival function, $c(\cdot) : \mathcal{N} \mapsto \mathcal{N}$.

// Initialize the functions for all nodes in the graph.
foreach $n \in \mathcal{N}$ **do** $d(n) \leftarrow \infty$, $c(n) \leftarrow n$
$d(n_g) \leftarrow 0$, $\mathcal{Q} \leftarrow \{n|(n, n_g) \in \mathcal{E}\}$
while $\mathcal{Q} \neq \emptyset$ **do**
  $n \leftarrow$`pop(`$\mathcal{Q}$`)`, $d_n \leftarrow d(n)$
  foreach $(n, n_o) \in \mathcal{E}$ **do**
    // $\|\cdot\|_2$ computes the Euclidean distance between two nodes.
    if $\|n, n_o\|_2 + d(b_o) < d(n)$ **then**
      $d(n) \leftarrow \|n, n_o\|_2 + d(b_o)$
      $c(n) \leftarrow n_o$
    end
  end
  if $d(n) < d_n$ **then** $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{n|(n_i, n) \in \mathcal{E}\}$
end
return $d(\cdot)$ *and* $c(\cdot)$

normal on-road driving is the lack of a sensible heuristic function that could accelerate the search process. We note that commonly used Euclidean distance is not sufficient in this case. Counter-intuitively, there are plenty of "deadends" on roads, not necessarily because of the undrivable areas, but due to the lane geometries to be respected in planning. As a result, with Euclidean distance as the heuristics, the search process of hybrid A* can be easily trapped, and cannot finish within limited time.

In the following, we extend hybrid A* to on-road motion planning. We first introduce a heuristic function that computes the path length from the given state to the goal respecting the road geometry. Then, we show the performance of hybrid A* with the proposed heuristic function.

The Heuristic Function

Like Euclidean distance, in this work, we use the path length between the given state and the goal as the heuristics. Since, regardless of the objective function used by hybrid A*,

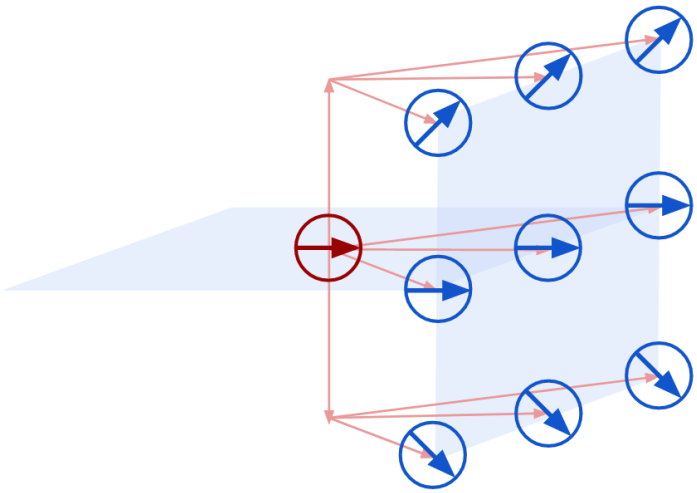


Figure 6.7: An example of edge construction. Considering the orientation of the source node (the red node), it is connected to the target nodes forward with the same direction (blue nodes in the middle row) with possible lane changes. Meanwhile, the source node is also connected to nodes forward with slightly different orientations (blue nodes in the top and bottom rows) to reflect the possible heading change of a vehicle following a path.

nodes that are close to the goal might be always promising to be expanded first. Especially, if path length is part of the objective function of hybrid A*, the heuristics is admissible. The constraint is that paths considered by the heuristic function should be conformal to the road structure.

The general idea for this purpose is to represent the structured environment with a graph. By carefully selecting the nodes and edges, the graph could be considered as a discretized version of the environment. Therefore, paths obtained by searching on the graph are kinematically feasible.

To be more specific, nodes are initialized by discretizing the 3-D space (2-D position and orientation). Instead of preserving all nodes, nodes that are at undrivable area or misaligned with the corresponding lanes are removed. A node is considered misaligned with the underlying lane when the orientation difference between the node and lane is greater than a predefined threshold. Directed edges are introduced based on the closeness of the nodes in the 3-D space. An edge is created between two nodes when it is (approximately)

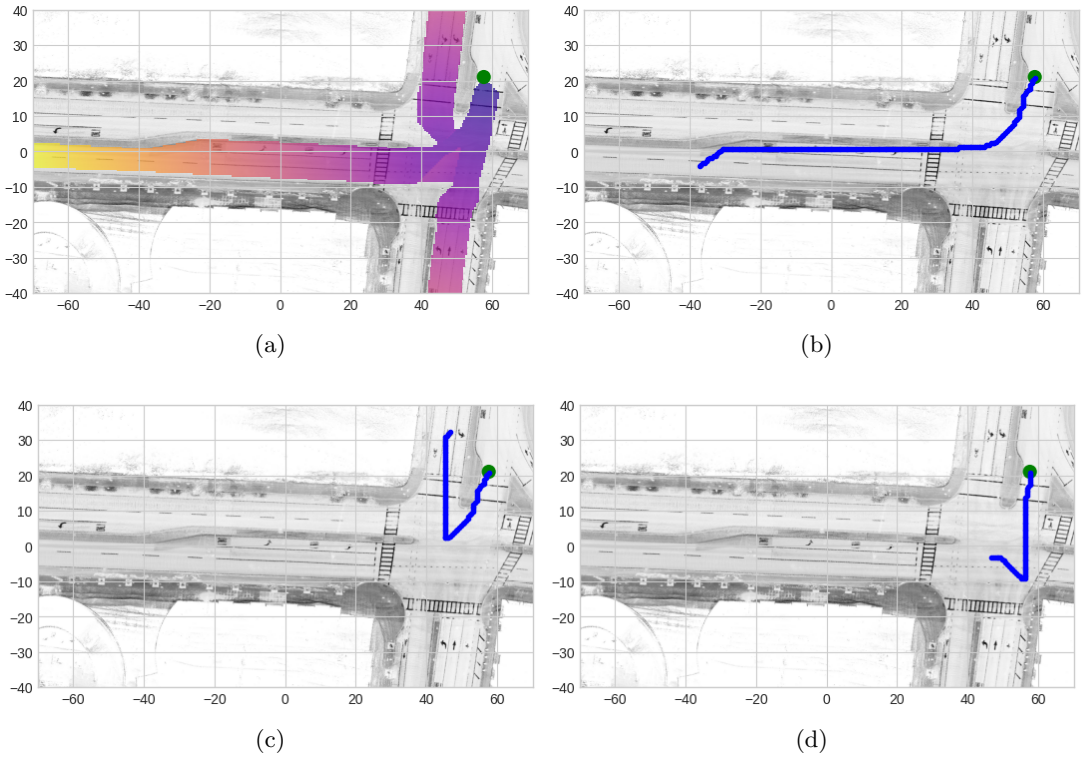


Figure 6.8: Results of Alg. 6 for an intersection environment. Given the goal pose (green circle on the top right corner of each image), (a) color codes the shortest distance from different positions to the goal position (yellow and purple represent large and small values respectively). Area without colors are infeasible. (b)-(d) show the reconstructed paths (blue lines) from different initial poses. Although violating the traffic rules, the path in (d) is considered feasible by the heuristic function. This is because the graph may not accurately reflect all structures of the environment, especially at intersections where lane connections are complex. However, as long as there are feasible paths from the query state to the goal state, cases in (d) can be avoided.

feasible for a vehicle to reach the state at the target node from the state at the source node. Figure 6.7 shows an example of edge construction. With the graph constructed, we could find the length of the shortest path between every node and the goal node as shown in Alg. 6. As a byproduct, we could also retrace the shortest path.

In this work, we set the spatial resolution and orientation resolution to 0.1m and $\frac{\pi}{4}$rad when creating the graph used in the heuristic function. The threshold in determining orientation alignment is set to $\frac{\pi}{8}$rad. Figure 6.8 shows the computed distance map for

a local environment of size 140m×80m, together with reconstructed paths from different initial poses.

Finally, we would like to note that, although the length of the paths are the shortest in terms of the graph, they might be different from the length of the shortest trajectories, where the latter considers the dynamics of a vehicle. Empirically, we find the length of the paths is within 5% error compared to the length of (approximately) shortest trajectories found by hybrid A*. Given the small difference, the path length found with Alg. 6 serves as an effective heuristics in accelerating hybrid A*.

The hybrid A* algorithm

Alg. 7 shows the hybrid A* algorithm implemented in this work using the proposed heuristic function. In the following we make a few remarks on Alg. 7.

First, for convenience, we assume the heuristic function takes vehicle states instead of graph nodes, which is slightly different from the output of Alg. 6. The minor difference can be made up by introducing additional steps that convert vehicle states to node indices based on the graph resolution used in the heuristic function.

Second, in the function `simulateMotion` (Line 10), only the ego vehicle is considered while other agents are ignored. The stage cost returned by the simulation is based on the trajectory length if the input motion primitive is feasible, *i.e.* not collide with the environment and respect the traffic rules. Otherwise, the stage cost is set to infinity. Hybrid A* could support more complicated stage cost in addition to trajectory length. However, given that the heuristic function is based on path length, using the same cost in hybrid A* could improve search efficiency.

Finally, in our implementation, we check the feasibility of the nodes on the fly (Line 15 and 21), *i.e.* check whether there exists a feasible trajectory from the state at the node to the goal. Nodes that fail the test are pruned from further expansion in later steps (Line 6).

In practice, the additional pruning step could significantly accelerate the search process. However, we note that the pruning may be premature, since when revisited, an infeasible node may be associated with a new state and become feasible again. However, the issue does not trigger concerns given that hybrid A* is not guaranteed to find the optimal trajectory in the first place. Meanwhile, the algorithm is used as a rollout policy in POMCP++. Although finding the optimal solution is preferred, it is more important to find a feasible solution efficiently considering the algorithm would be repeatedly called in POMCP++.

Figure 6.9 shows example solutions of hybrid A* with random start and goal poses. In the examples, the set of motion primitives is {-2, 0, 2 m/s^2} $\times$ {-0.25, 0, 0.25 rad/s}. The duration of each discrete time step is 0.5s. The resolution used in the function `toNode` in Alg. 7 is 0.5m for position, 0.1125rad for orientation, and 0.9m/s for velocity. Given the resolution, the vehicle state is guaranteed to land on a different node after one time step with non-zero motion primitives. It could be observed from Figure 6.9 that significantly more nodes are expanded in Figure 6.9f. This is due to the fact that dynamical feasibility is not considered in the heuristic function. For example, at the region close to the goal in Figure 6.9f, the heuristic function assigns shorter paths to the nodes on the diagonal. However, once dynamics is considered, these nodes are infeasible. All such nodes need to be expanded in full before a feasible trajectory can be found. In practice, we find cases like Figure 6.9f are rare. In most configurations, hybrid A* could finish by expanding under 1000 nodes.

Algorithm 7: Hybrid A* based on the Work of Dolgov et al. (2008)

Input: The start state $\boldsymbol{x}_s$. The goal state $\boldsymbol{x}_g$.
The heuristic function $d(\cdot) : \mathcal{X} \mapsto \mathbb{R}$.
Output: The trajectory from the start to the goal $\mathcal{T}$.

// Initialize the state, cost, parent, and motion primitive at the starting node.
$n_s \leftarrow$ `toNode(`$\boldsymbol{x}_s$`)`, $s(n_s) \leftarrow \boldsymbol{x}_s$, $c(n_s) \leftarrow 0$, $p(n_s) \leftarrow n_s$, $a(n_s) \leftarrow nan$
$\mathcal{G} \leftarrow \{n_s\}$ *// Keep track of all visited nodes.*
$\mathcal{Q} \leftarrow \{(d(\boldsymbol{x}_s), n_s)\}$ *// A priority queue of nodes to be expanded.*
while $\mathcal{Q} \neq \emptyset$ **do**
  $n \leftarrow$ `pop(`$\mathcal{Q}$`)`
  if $c(n)$ *is nan* **then continue**
  if `close(`$s(n), \boldsymbol{x}_g$`)` **then** $n_g \leftarrow n$, **break**
  $f(n) \leftarrow False$ *// Mark the feasibility of this node.*
  foreach $\boldsymbol{a} \in \mathcal{A}$ **do**
    $s', g \leftarrow$ `simulateMotion(`$s(n)$, $\boldsymbol{a}$`)`
    $n' \leftarrow$ `toNode(`s'`)`
    if $d(s')$ *is* ∞ **then continue**
    if $n' \in \mathcal{G}$ *and* $c(n')$ *is nan* **then continue**
    if g *is* ∞ **then continue**
    $f(n) \leftarrow True$
    if $n' \notin \mathcal{G}$ *or* $c(n) + g < c(n')$ **then**
      $s(n') \leftarrow s'$, $c(n') \leftarrow c(n) + g$, $p(n') \leftarrow n$, $a(n') \leftarrow \boldsymbol{a}$
      $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(c(n') + d(s'), n')\}$
    end
  end
  // The node is marked if it does not lead to feasible solutions.
  // Next time the same node is visited, we can simply ignore it for efficiency.
  if *not* $f(n)$ **then** $c(n) \leftarrow nan$
  // Backtrace the trajectory as a sequence of motion primitives.
  $\mathcal{T} \leftarrow \{\}$, $n \leftarrow n_g$, $n_p \leftarrow p(n_g)$
  while $n \neq n_p$ **do**
    $\mathcal{T} \leftarrow \{a(n_p)\} \cup \mathcal{T}$
    $n \leftarrow n_p$, $n_p \leftarrow p(n)$
  end
  return $\mathcal{T}$
end

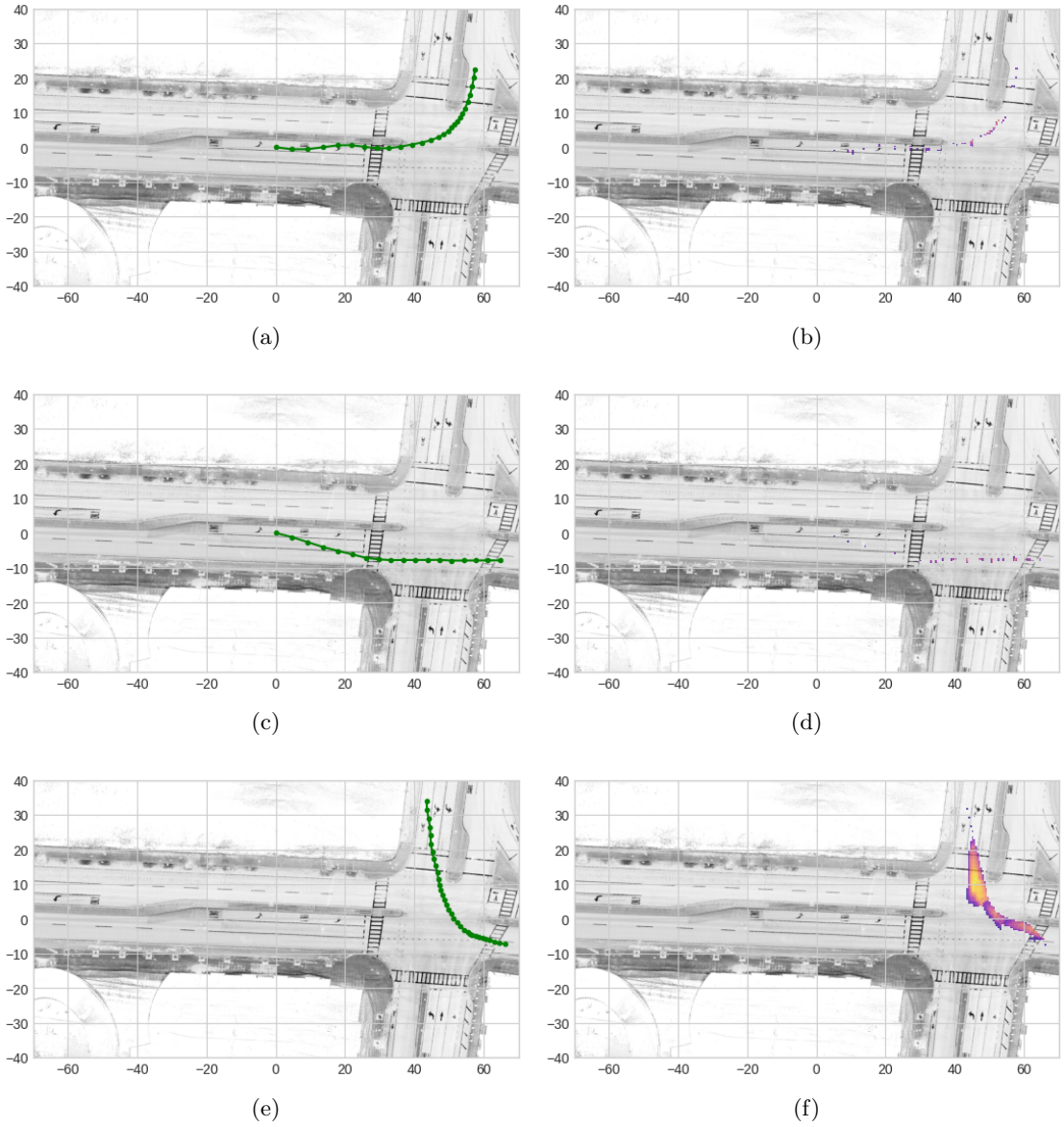


Figure 6.9: Example results of hybrid A* (Alg. 7) in an intersection environment of size 140m×80m with random start and goal poses. (a), (c), and (e) show the final trajectories (green curves). (b), (d), and (f) show the corresponding expanded nodes projected onto the 2-D plane (yellow and purple represent large and small values respectively). For (b), (d), and (f), the number of expanded nodes are 177, 45, and 14665 respectively.

6.3.2 Experiments

With the rollout policy defined, we are able to apply POMCP++ to plan motions for the ego vehicle within stochastic traffic following Alg. 4. In the rest of this section, instead of directly applying POMCP++ with the learned stochastic traffic model, we first study the performance of POMCP++ with handcrafted scenarios. Although we are not able to model the responsive behavior of agents in handcrafted scenarios, it is much easier to create scenarios of interest, which are often tail events in real driving experience. Then, we close the loop between POMCP++ and the learned stochastic traffic dynamical model with real traffic scenarios. In real scenarios, the traffic dynamics is forward simulated using the learned model, of which the initial traffic states are set using the data in nuScenes (Caesar et al., 2020).

In both handcrafted and real traffic scenarios, we compare the performance of POMCP++ with MPDM (Galceran et al., 2017), which might be the most similar to our work in terms of both problem setup and solution. As reviewed at the beginning of this chapter, MPDM also employs a POMDP solution to plan motions for the ego vehicle within a stochastic traffic environment. One special feature of MPDM is that it partially ignores the belief dynamics. At planning, belief is not affected by different possible future measurements. We note that MPDM is a system-level method, for which we are not able to reproduce all details in our implementation. As a result, we adapt POMCP++ to capture the high level idea of MPDM. In our implementation of MPDM, we follow the most of the steps in POMCP++. However, at traversing the tree, belief does not change with different measurement branches.

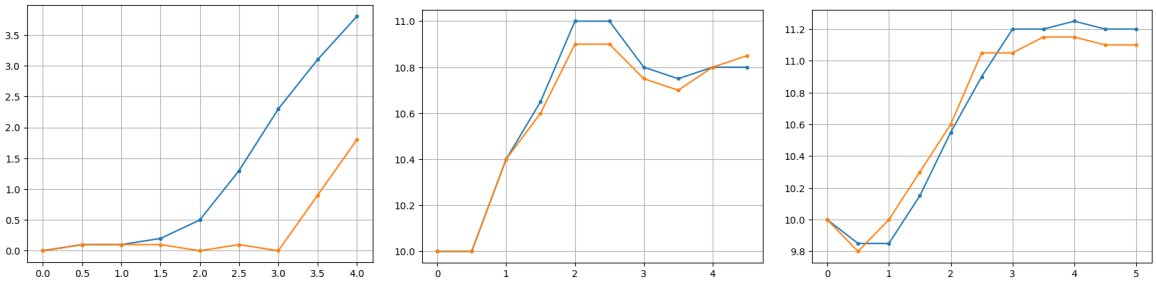
Handcrafted Traffic Scenarios

With the handcrafted traffic scenarios, we create situations where motion of the ego vehicle heavily depends on the behavior of the agent vehicle. Each scenario starts from the same initial traffic state, but rollouts differently based on different behaviors of the agent vehicle, which is either obliging or disobliging. The motion of agents is predetermined, *i.e.* the agent

Table 6.3: Hyper-parameters used in the Handcrafted Traffic Scenarios

γ	ϵ_a	ϵ_z	K[1]	N	$t_{\max}$[2]
0.99	0.2	-2	2	1000	600s

[1] There are two cases for each scenario. Since we assume no motion uncertainty, instead of a particle filter, the belief can be directly tracked with a Bayes filter.
[2] The planning algorithm stops with either the episode limit N or time limit $t_{\max}$ hits.



(a) (b) (c)

Figure 6.10: Average velocity profiles of the ego vehicle planned with POMCP++ in (a) unprotected left turn, (b) lane change, and (c) braking scenarios. Blue curves are the velocity profiles with obliging agent vehicles, while orange curves are obtained with disobliging agents. Each velocity profile is averaged over 10 independent simulations. Note simulations may finish at different time steps. Here, we only consider the average velocity profile up to the shortest duration of all simulations of the corresponding scenario.

would not respond to different motions of the ego vehicle. For handcrafted scenarios, we assume the traffic dynamics is deterministic. However, the initial traffic state is unknown. In the simulations, the ego vehicle is presented with both agent behaviors initially, with each accounts for 50% probability. The ego vehicle has to estimate the real agent behavior on the fly using the noisy measurement, (6.10), and make decisions correspondingly. Table 6.3 summarizes the common hyper-parameters for the planning algorithms used in the handcrafted scenarios. In the following, we describe each scenario in detail.

- Unprotected left turn scenario

  The ego vehicle is at an intersection waiting to turn left. There is an on-coming agent vehicle on the opposite lane. The agent vehicle may accelerate through the intersection, or decelerate to stop before the intersection. The situation is often encountered

Table 6.4: Average Discounted Summation of Rewards of MPDM and POMCP++ in Handcrafted Traffic Scenarios[1]

scenario	method	obliging	disobliging	mean[2]
Unprotected left turn	MPDM	-135.64	-891.23	-513.44
	POMCP++	-155.38	-178.72	**-167.05**
Lane change	MPDM	-113.21	-114.09	-113.65
	POMCP++	-113.81	-112.43	**-113.12**
Braking[3]	MPDM	-137.09	-136.71	**-136.90**
	POMCP++	-139.72	-141.01	-140.37

[1] For each entry in the table, the data is averaged over 10 independent trials.
[2] This column shows the average reward of cases with obliging and disobliging agent drivers. It is the actual objective the planning algorithms try to maximize.
[3] For the braking scenario, the disobliging agent refers to the one that hard-breaks, while the obliging agent refers to the one that maintains speed.

when waiting to turn at intersections with yellow lights. In this scenario, the motion primitives of the ego vehicle is set to {-2, 0, 2 m/s^2}×{-0.25, 0, 0.25 rad/s}.

- Lane change scenario

  The ego vehicle intends to change to the target lane on the right. There is a fast-approaching agent vehicle on the target lane at the back of the ego vehicle. The agent vehicle may decelerate to yield or accelerate to discourage the lane change. In this scenario, the motion primitives of the ego vehicle is set to {-1, 0, 1 m/s^2}×{-0.2, 0, 0.2 rad/s}.

- Braking scenario

  The ego vehicle maintains its lane following a leading agent vehicle. The agent vehicle may keep its speed or break abruptly due to emergencies. In this scenario, the motion primitives of the ego vehicle is set to {-1, 0, 1 m/s^2}×{-0.2, 0, 0.2 rad/s}.

Figure 6.11 shows the paths of the ego vehicle controlled with POMCP++ in different scenarios. Figure 6.10 shows the corresponding velocity profiles. For the unprotected left

turn scenario, although the paths followed by the ego vehicle for different agent behaviors are similar as shown in Figure 6.11a and 6.11b. It is clear from Figure 6.10a the ego vehicle starts to turn much earlier when the agent vehicle yields compared to the case with a disobliging agent. For the lane change scenario, the ego vehicle either merges ahead or behind of the agent vehicle depending on estimated behavior of the corresponding agent. Also from Figure 6.10b, it could be observed that the ego vehicle moves slightly faster when the agent vehicle yields. In the braking scenario, there exists a high reward trajectory for the ego vehicle regardless of the behavior of agents. The ego vehicle could always change to a neighbor lane and then back to avoid potential collisions with the leading agent, while keeping the total number of steps small.

Table 6.4 compares the average discounted summation of rewards achieved by MPDM and POMCP++ in the handcrafted traffic scenarios. Although Table 6.4 reports the reward of both methods under different agent behaviors separately, we note that the average reward obtained under different agent behaviors, the actual objective, should be considered for comparison. In the following, we provide remarks on the reported data. We first comment on the low reward of MPDM with the unprotected left turn scenario. Then we analyze the high reward of MPDM compared to POMCP++ in the braking scenario. In short, the underlying reason for both is that belief dynamics is (partially) ignored in MPDM. It causes collision in the former case, but works favorably in the latter.

It could be observed from Table 6.4 that MPDM has significantly low reward with a disobliging agent in the unprotected left turn scenario. The low reward of MPDM is caused by the high collision rate, 8 out of 10 times. Recall in MPDM, the change of belief due to future measurements is not considered. Therefore, given the initial uncertainty, a collision cannot be avoided without creating deep branches for an initial unpromising node. Considering the nature of MCTS, it is rarely the case to generate a subtree with sufficient depth for an unpromising node under the limitations of episodes and running time. Given that a collision cannot be avoided, under the reward function in (6.11), the logical decision

Table 6.5: Hyper-parameters used in the Real Traffic Scenarios

γ[1]	ϵ_a	ϵ_z	K[2]	N	$t_{\max}$
0.8	0.2	-2	10	1000	600s

[1] Predictions from the learned traffic model is only valid for a short horizon. Compared to handcrafted scenarios, γ in real scenarios is decreased to reduce the effective planning horizon.

[2] Although particle filters are subject to particle deprivation with a small number of particles. Empirically, we find 10 samples are sufficient for a short horizon.

is to start the turn early which minimizes the total time steps. Even though the state estimator is eventually certain that the agent vehicle would not yield, the ego vehicle is not able to stop in time to avoid collisions. For the same reason, MPDM could achieve a higher reward with an obliging agent in the unprotected left turn scenario. With full belief dynamics considered in POMCP++, initial nodes with zero motion is promising since they help in avoiding collisions when the agent is disobliging. Therefore, such nodes are explored more thoroughly. The above issue with MPDM is avoided in POMCP++.

In the braking scenario, MPDM could achieve a higher reward for a similar reason as discussed above. In this scenario, the same trajectory for the ego would yield high reward regardless of the agent behavior. MPDM actually searches for a trajectory that works well for both cases. Therefore, it could lock on to the corresponding branch in the tree early and spend much effort later exploring the branch. In POMCP++, since belief dynamics is considered, extra effort is spent in looking for if there are better options for different cases. As a result, the branch of the final trajectory is not explored thoroughly, which explains the lower reward of POMCP++ compared to MPDM.

Real Traffic Scenarios

In addition to the handcrafted scenarios, we also show the performance of POMCP++ with real traffic scenarios. We create four scenarios, lane following, lane change, left turn, and right turn, with each requiring different maneuver from the ego vehicle. In real scenarios, traffic is initialized with selected data from nuScenes (Caesar et al., 2020) dataset, and

Table 6.6: Average Discounted Summation of Rewards of MPDM and POMCP++ in Real Traffic Scenarios[1]

	Lane following	Lane change	Left turn	Right turn
MPDM	-65.19	-51.25	-65.01	-64.96
POMCP++	**-62.03**	**-46.81**	**-64.18**	**-64.75**

[1] For each entry in the table, the data is averaged over 5 independent trials.

forward simulated with the learned stochastic traffic dynamical model introduced in Sec. 6.1. In contrast to the handcraft scenarios, we assume the initial traffic state, including both vehicle states and intention[8], are known. Instead, uncertainty is introduced by the stochastic traffic model. In our implementation, a particle filter is used to keep track of the belief at each step. The hyper-parameters used for real scenarios are reported in Table 6.5.

Figure 6.12 shows the trajectories of the ego vehicle controlled with POMCP++ in the four real traffic scenarios. For each scenario, we show two independent simulations with different traffic predictions and/or ego trajectories. A few interesting behaviors of the ego vehicle can be observed. In the lane following scenario, the ego vehicle swerves to the right in both simulations. The traffic model predicts the agent vehicle on the neighbor lane may intrude into the lane of the ego. Therefore, the ego vehicle makes room beforehand in order to prevent collisions. Meanwhile, the ego vehicle adjusts its speed according to the neighbor agent so that they would not drive in parallel which agrees with common defensive driving behavior. In the right turn scenario, it could be also observed that the ego vehicle adjusts its speed based on the motion of its leading vehicle. However, we note that most of the simulations of the same scenario are similar. As discussed at the beginning of this section, strong interactions between vehicles are rare cases in actual driving experiences. The ego may always follow the same trajectory, possibly with slight perturbations, regardless of the behavior of other agent vehicles.

Table 6.6 compares the discounted summation of rewards obtained by MPDM and

[8]The intention of an agent is computed based on its past and initial states.

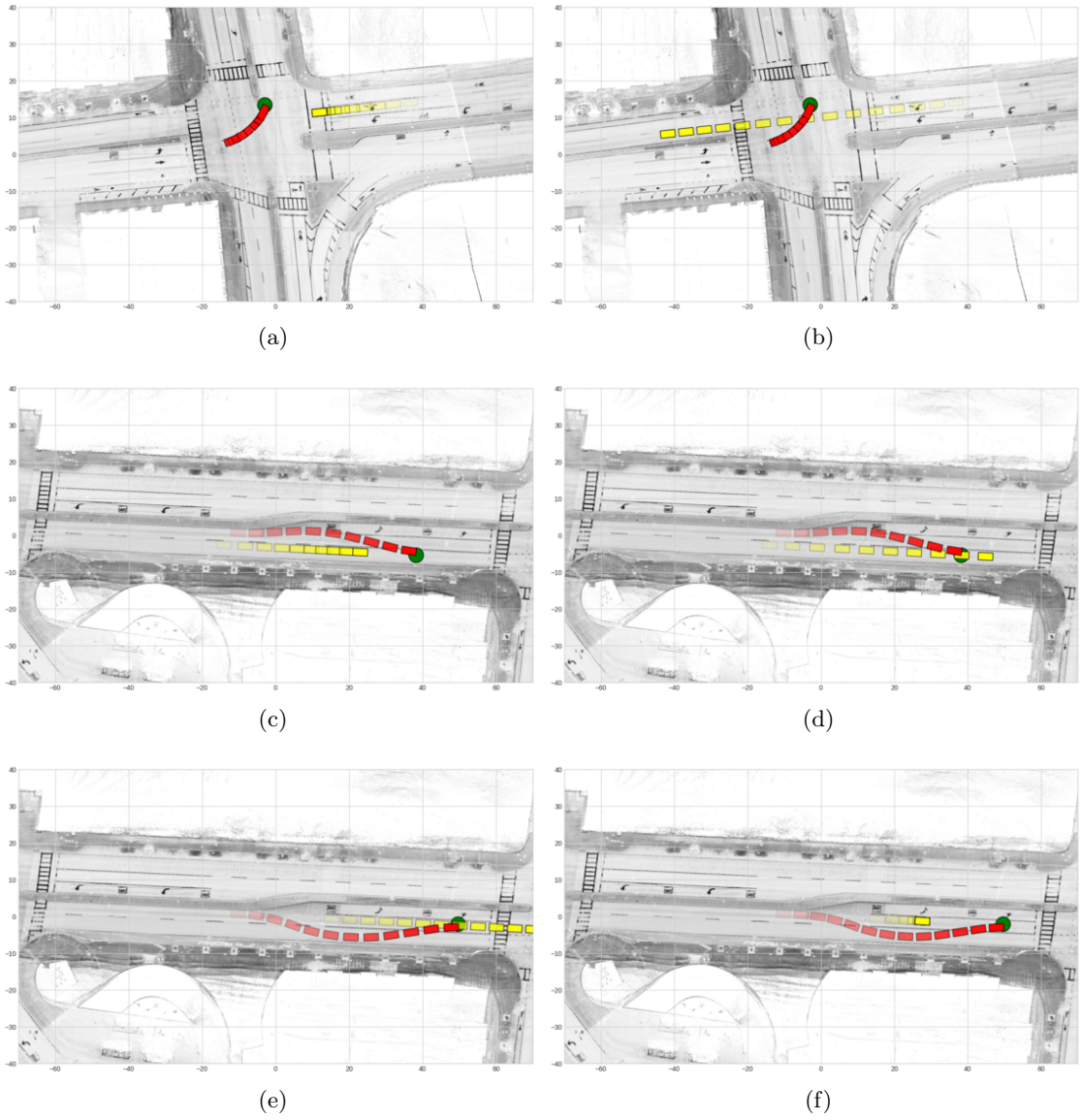
POMCP++ with real traffic scenarios. It could be observed that POMCP++ outperforms MPDM consistently over all scenarios. Compared to the handcrafted scenarios, this could be due to the extra stochasticity introduced by the learned traffic model.

Running Time

Finally, we comment on the running time of POMCP++ in the autonomous driving application. In this section, we demonstrate that POMCP++ is able to make sensible decisions in various scenarios, which is the goal of this work. However, it may not be sufficiently efficient for real time usage in the current form. Each reported simulation of handcrafted or real scenarios may take half to one hour to finish.

One obvious reason for the inefficiency is that POMCP++ is a sampling-based algorithm which requires a large number of episode to generate reasonable actions. In addition, the implementation of this work is constrained to Python. The GNN model for stochastic traffic dynamics is implemented with PyTorch. More importantly, the toolkit[9] to work with the nuScenes dataset is only available in Python. Unfortunately, Python is known to be inefficient with unparallelizable loops that show up frequently not only in POMCP++ but also in the hybrid A* algorithm. In the future work, we may need to transfer the current implementation to C++ in order to better assess the running time of POMCP++ in the autonomous driving application and make further improvements.

[9] `https://github.com/nutonomy/nuscenes-devkit`



(a) (b) (c) (d) (e) (f)

Figure 6.11: Simulations of POMCP++ with handcrafted scenarios. The Red and yellow boxes represent the ego and the agent vehicles. More transparent boxes represent vehicle poses further into the history. The green dots mark the local goal waypoint. The top to bottom rows are the (a, b) unprotected left turn, (c, d) lane change, and (e, f) braking scenarios respectively. The left column (a, c, e) shows the cases with obliging agent vehicles for each scenario, while the right column (b, d, f) shows the cases with disobliging agent vehicles.

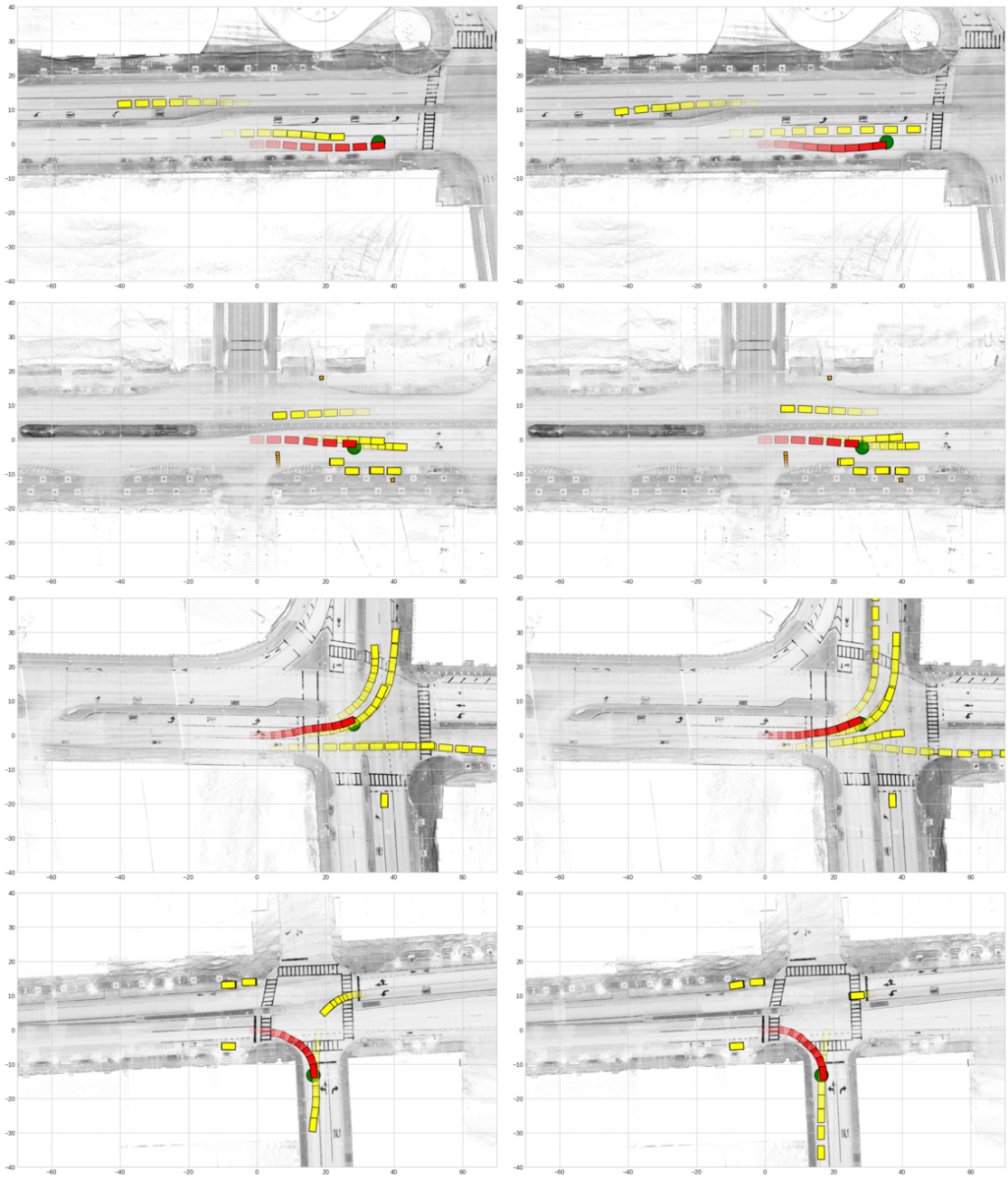


Figure 6.12: Example simulation results of POMCP++ in (first row) lane following, (second row) lane change, (third row) left turn, and (fourth row) right turn real traffic scenarios. For each scenario, the traffic starts from the same initial state but may rollout differently due to the learned stochastic traffic models. See Figure 6.11 for the annotations in the figures.

Chapter 7

Conclusions and Future Work

In this dissertation, we address the stochastic motion planning problem by modeling it as POMDPs and propose two solutions with theoretical justifications featuring different optimization regimes. We demonstrate the effectiveness of the proposed solutions in various applications through comparisons with other applicable approaches.

In Ch. 3, we propose an improved version of iLQG. It is a gradient-based method that can directly work with systems having nondifferentiable models and sparse informative measurements. The improvement promises a wider range of applications compared to the original work on iLQG (van den Berg et al., 2012b). In Ch. 5, we show an application of the proposed solution in navigating a mobile robot with range sensors, which is not previously possible with the original iLQG algorithm.

Ch. 4 presents POMCP++, a sampling-based general POMDP solver. The proposed solution overcomes the degeneracy issue of MCTS when working with continuous measurement spaces. Meanwhile, with sufficient simulation episodes and samples, we show that POMCP++ is a valid Monte Carlo control algorithm alternating unbiased policy evaluation and policy improvement. Again, we apply POMCP++ in navigating a mobile robot with range sensors in Ch. 5. In this application, we show that POMCP++ is able to consistently outperform other similar works.

In addition to the navigation application in Ch. 5 where the dominant uncertainty comes from the robot itself, we show the application of POMCP++ in autonomous driving in Ch. 6 featuring uncertainty from the external environment. In this work, we propose a data-driven model for the stochastic traffic dynamics in the form of a general motion model, where we explicitly model the intention of human drivers. Through comparisons with other similar works, we show the superior performance of POMCP++, which is able to compose intelligent high-level behaviors with motion primitives. To our best knowledge, this is the first work that applies a full POMDP solution to navigate a self-driving car in a general setting. This work might also be the first to close the loop between motion planning and data-driven traffic models.

In the following, we propose future research directions in hard constraints, solution efficiency and system modeling for stochastic motion planning problems.

Hard Constraints

In both proposed solutions in this dissertation, we assume the task can be sufficiently modeled with an objective function. However, hard constraints might be necessary for certain applications and robotic platforms. The hard constraints may be on the control input due to dynamical limitations or on the belief for safety and reliability considerations. The latter is often known as risk-aware (Pereira et al., 2013) or chance-constrained (Vitus and Tomlin, 2011a) motion planning.

Incorporating such constraints transfers iLQG into a constrained nonlinear programming (Nocedal and Wright, 2006), which is notoriously challenging to solve. It might be promising to look for reasonable additional structures on the problem formulation which may lead to efficient and reliable convergence of the solution. On the other hand, hard constraints on control and belief can be incorporated into POMCP++ more easily with careful selection of motion primitives and sufficient number of samples to approximate the underlying belief. However, it still remains as an important question to determine the num-

ber of samples adaptively so that the constraints can be satisfied with high confidence while maintaining low computation burden.

Solution Efficiency

Another limitation of the proposed solutions lies with their runtime complexity. Neither of the work is sufficiently efficient to be applied for real time purposes[1]. This is simply because controlling the stochastic belief dynamics, or stochastic process in general, in an optimal way is genuinely hard.

Fortunately, the optimality of a control policy is not often of critical concern in practical applications. In order to improve efficiency, we may draw insights from the framework of controlling stochastic physical dynamics in practice, which consists of a "planner" and a "controller" as shown in Figure 1.3. The planner generates a reference trajectory, while the controller ensures the trajectory is being closely followed. Given the closeness[2] between the target and current state, the task of the controller might be relatively easy in terms of computation since constraints and objective functions are often no longer of concern.

In the case of controlling stochastic belief dynamics, we may expect the planner to generate a belief reference trajectory possibly assuming a deterministic belief dynamics. Such a planner might already exist given the literature assuming maximum likelihood measurements (Kopitkov and Indelman, 2019; Patil et al., 2014). Developing a spatiotemporal stationary "belief controller", on the other hand, is challenging, which might require in-depth understanding of the underlying belief space and the corresponding belief dynamics. If successful, it might revolutionize the existing practical frameworks applied in relevant applications.

[1] Although iLQG optimizes for control policies offline which can then be deployed online efficiently, it may not meet the real time requirement if replanning is necessary.

[2] The closeness is not necessarily defined with Euclidean space, but any sensible metric adapted to the state space under consideration.

System Modeling

In addition to developing algorithms, system modeling is also of crucial importance since they are tightly coupled into motion planning algorithms for making predictions.

In Ch. 5, we consider the application of navigating a mobile robot in a known environment. For more applications, however, robots are operated in unknown environments. Modeling the belief dynamics for such applications requires the solution of Simultaneous Localization and Mapping (SLAM) (Thrun et al., 2005) problems. More challenging than SLAM problems, it requires the joint distribution of robot state and map instead of simply maximum *a posterior* estimation. Common SLAM solutions, especially ones based on optimization, are not sufficient for this purpose. The key to address the issue may lie with a better map representation. The new map representation should not only lead to compact measurement models but also efficient computation of joint distributions.

We believe the challenge of the autonomous driving application is also from modeling the stochastic traffic dynamics. In Ch. 6, we propose a traffic model sharing the same structure as the general motion model in (1.1), so that it can be tightly coupled with planning algorithms. However, there might be more features of traffic dynamics to be captured. For joint predictions of all agents, samples are drawn from marginal distributions of each agent instead of a joint distribution. As a result, the predictions might be inconsistent and lead to collisions. It remains to be addressed how to define and generate a joint distribution for all agents in the traffic. The proposed traffic model is also limited to only making predictions for detected agents. It fails in capturing undetected agents which might show up in the future and affect the ego motion at the present. The capability of the model is especially important for scenarios like blind turning where we expect the ego vehicle to creep forward and take a better observation of potential up-coming vehicles. The above mentioned two limitations are not unique to this work, but exist in almost all works in the literature. It might be important to have a consensus on traffic modeling among the community before we could further improve motion planning for self-driving cars.

Appendix A

Supplementary Proofs of POMCP++

A.1 Infinitely Often Visits of Nodes

Lemma A.1. *A belief action node $h\boldsymbol{a}$ will be visited i.o. given that its parent belief node h is visited i.o. and $0 < \epsilon_a < 1$.*

Proof. Define A_n as the event that $h\boldsymbol{a}$ is visited at the n^{th} visit of h. A_n's are independent events following the definition of the action selection function in Alg. 4. Note $P(A_n)$ is lower bounded by $1/\epsilon_a$ with $0 < \epsilon_a < 1$. Therefore,

$$\sum_{n=1}^{\infty} P(A_n) \geq \lim_{n\to\infty} \frac{n}{\epsilon_a} = \infty. \tag{A.1}$$

Define A as the event that infinitely many A_n's occur, *i.e.* $A = \limsup_{n\to\infty} A_n$. We have $P(A) = 1$ as a result of Borel-Cantelli lemmas (Grimmett and Stirzaker, 2020). □

Lemma A.2. *A belief node $h\boldsymbol{z}$ will be visited i.o. given that its parent belief action node h is visited i.o. and $\epsilon_z < 0$.*

Proof. Assume $h\boldsymbol{z}$ is the m^{th} spawned child node of h. Define Z_n as the independent events that $h\boldsymbol{a}$ is visited at the n^{th} visit of h after $h\boldsymbol{z}$ is spawned. $P(Z_n)$ with $n = 1, 2, \dots$ is a nonincreasing sequence since the number of child nodes of h is nondecreasing. Note $P(Z_n)$ is lower bounded by $(1 - (m+n)^{\epsilon_z})/(m+n-1)$, which assumes new child node is added at every visit of h. Therefore,

$$\begin{aligned}\sum_{n=1}^{\infty} P(Z_n) &\geq \sum_{n=1}^{\infty} \frac{1-(m+n)^{\epsilon_z}}{m+n-1} \\ &= \sum_{n=1}^{\infty} \frac{1}{m+n-1} - \sum_{n=1}^{\infty} \frac{1}{(m+n-1)(m+n)^{-\epsilon_z}} \\ &= \sum_{k=m}^{\infty} \frac{1}{k} - \sum_{k=m}^{\infty} \frac{1}{k(k+1)^{-\epsilon_z}} \\ &\geq \sum_{k=m}^{\infty} \frac{1}{k} - \sum_{k=m}^{\infty} \frac{1}{k^{1-\epsilon_z}}.\end{aligned} \tag{A.2}$$

Given $\epsilon_z < 0$, $\sum_{n=1}^{\infty} P(Z_n) = \infty$ following the property of p-series. We have $P(\limsup_{n\to\infty} Z_n) = 1$ as a result of Borel-Cantelli lemmas. □

Proof of Theorem 4.1. Based on Lemma A.1 and A.2, the *i.o.* visit of a node in the tree depends on the *i.o.* visit of its parent nodes. Therefore, all nodes are visited *i.o.* if the root node is visited *i.o.*. □

Proof of Theorem 4.2. Define S_n as the independent events that a new child belief node is spawned by h at its n^{th} visit. The sequence $P(S_n)$, $n = 1, 2, \dots$ is nonincreasing, with each $P(S_n)$ lower bounded by $(n+1)^{\epsilon_z}$. Therefore,

$$\sum_{n=1}^{\infty} P(S_n) \geq \sum_{n=1}^{\infty} (n+1)^{\epsilon_z}$$

A sufficient condition for $\sum_{n=1}^{\infty} P(S_n) = \infty$ is $-1 \leq \epsilon_z < 0$ ($\epsilon_z < 0$ is enforced to ensure $0 < (n+1)^{\epsilon_z} < 1$), under which $P(\limsup_{n\to\infty} S_n) = 1$, *i.e.* h spawns new child nodes *i.o.* . □

A.2 Unbiased policy evaluation

Lemma A.3. *The discounted summation of rewards of the simulation episode is an unbiased estimate of the value at the traversed belief nodes as $N \to \infty$ and $K \to \infty$.*

Proof. Represent a simulation episode from belief node $h_t \in \mathcal{T}$ as $S_t \coloneqq \boldsymbol{x}_t, \boldsymbol{a}_t, \boldsymbol{z}_t, \ldots$, with $\boldsymbol{x}_t$ sampled from $\boldsymbol{b}_t$. For the sake of notation sanity, define $X_t \coloneqq \boldsymbol{x}_{t:\infty}$, $A_t \coloneqq \boldsymbol{a}_{t:\infty}$, $Z_t \coloneqq \boldsymbol{z}_{t:\infty}$, i.e. $S_t = X_t, A_t, Z_t$. Then the value of the node h_t under the policy π is,

$$V_\pi(h_t) = \int_{S_t} R(S_t) p_\pi(S_t) dS_t. \tag{A.3}$$

Recall $R(S_t)$ is the discounted summation of reward of a simulation episode, $p_\pi(S_t)$ is the *p.d.f.* of S_t. Here and also in the following context, the subscript π in $p_\pi(\cdot)$ indicates the *p.d.f.* is related to the policy π. With some application of conditional probability and Bayes' rule, one has,

$$\begin{aligned} V_\pi(h_t) &= \int_{S_t} R(S_t) p_\pi(S_t) dS_t \\ &= \int_{S_t} R(S_t) p(X_t|A_t, Z_t) p_\pi(A_t, Z_t) dS_t \\ &= \int_{S_t} p_\pi(A_t, Z_t) R(S_t) \frac{p(Z_t|X_t, A_t) p(X_t|A_t)}{p(Z_t|A_t)} dS_t \\ &= \int_{A_t, Z_t} p_\pi(A_t, Z_t) V(h_t|A_t, Z_t) dA_t dZ_t, \end{aligned} \tag{A.4}$$

where,

$$V(h_t|A_t, Z_t) = \int_{X_t} R(S_t) \frac{p(Z_t|X_t, A_t) p(X_t|A_t)}{p(Z_t|A_t)} dX_t. \tag{A.5}$$

$V(h_t|A_t, Z_t)$ is the value of the node h_t given the future measurement and action sequence. In Alg. 4, $V(h_t|A_t, Z_t)$ is estimated with,

$$\tilde{V}(h_t|A_t, Z_t) = \frac{1}{\eta} \sum_{i=0}^{K-1} R(S_t^i) p(Z_t|X_t^i, A_t), \tag{A.6}$$

where X_t^i is forward sampled following the given action sequence A_t with the initial state $\boldsymbol{x}_t^i$ sampled from $\boldsymbol{b}_t$. Geweke (1989) proves the validity of estimation with importance sampling under weak assumptions[1], *i.e.*,

$$\tilde{V}(h_t|A_t, Z_t) \xrightarrow[\text{as } K\to\infty]{a.s.} V(h_t|A_t, Z_t), \tag{A.7}$$

where $\xrightarrow{a.s.}$ means almost surely convergence. Therefore,

$$\int_{A_t,Z_t} p_\pi(A_t, Z_t)\tilde{V}(h_t|A_t, Z_t)dA_t dZ_t \xrightarrow[\text{as } K\to\infty]{a.s.} V_\pi(h_t). \tag{A.8}$$

If one samples a measurement and action sequence from $p_\pi(A_t, Z_t)$, (A.8) shows $\tilde{V}(h_t|A_t, Z_t)$ is an unbiased estimate of $V_\pi(h_t)$. However, in following the tree policy, (A_t, Z_t) is sampled from the distribution $p_\mathcal{T}(A_t, Z_t)$ defined implicitly by the tree $\mathcal{T}$, instead of $p_\pi(A_t, Z_t)$. In the following, we show that if new measurements are sampled *i.o.* from each belief-action node in $\mathcal{T}$, $\tilde{V}(h_t|A_t, Z_t)$ still serves as an unbiased estimate of $V_\pi(h_t)$ even though (A_t, Z_t) is sampled based on $p_\mathcal{T}(A_t, Z_t)$.

To start, consider the first action measurement pair $(\boldsymbol{a}_t, \boldsymbol{z}_t)$ in (A_t, Z_t). Define $f(\cdot)$ to be an integrable function *w.r.t.* the corresponding probability measure (a bounded function defined on the sample space should suffice). Then,

$$\begin{aligned}
\mathop{\mathbb{E}}_{p_\mathcal{T}(\boldsymbol{a}_t,\boldsymbol{z}_t)}\{f(\boldsymbol{a}_t, \boldsymbol{z}_t)\} &= \int_{\boldsymbol{a}_t}\left(\frac{1}{n}\sum_{i=0}^{n-1} f(\boldsymbol{a}_t, \boldsymbol{z}_t^i)\right)p_\pi(\boldsymbol{a}_t)d\boldsymbol{a}_t \\
&\xrightarrow[\text{as } n\to\infty]{a.s.} \int_{\boldsymbol{a}_t}\left(\int_{\boldsymbol{z}_t} f(\boldsymbol{a}_t, \boldsymbol{z}_t)p_\pi(\boldsymbol{z}_t|\boldsymbol{a}_t)d\boldsymbol{z}_t\right)p_\pi(\boldsymbol{a}_t)d\boldsymbol{a}_t \\
&= \int_{\boldsymbol{a}_t,\boldsymbol{z}_t} f(\boldsymbol{a}_t, \boldsymbol{z}_t)p_\pi(\boldsymbol{a}_t, \boldsymbol{z}_t)d\boldsymbol{a}_t d\boldsymbol{z}_t.
\end{aligned} \tag{A.9}$$

The first equality in (A.9) is based on how actions and measurements are selected at node $h_t \in \mathcal{T}$ in Alg. 4. Here and in the following context, we assume that n measurements are

[1]Briefly speaking, the four assumptions by Geweke (1989) require the proper definition of prior density and the likelihood function, *i.i.d.* sample sequence, and an integrable *r.v. w.r.t.* posterior density. All of the assumptions are satisfied in our problem setup.

sampled for each belief-action node in the tree with n sufficiently large. With large n, we can safely ignore the probability of sampling new measurements at the belief action nodes, *i.e.* one of the n measurements will be selected. The *a.s.* convergence in (A.9) follows the strong law of large numbers.

Mathematical induction lends itself to prove the rest of the action, measurement sequence. Suppose,

$$\mathop{\mathbb{E}}_{p_{\mathcal{T}}(\boldsymbol{a}_{t:T-1},\boldsymbol{z}_{t:T-1})}\{f(\boldsymbol{a}_{t:T-1},\boldsymbol{z}_{t:T-1})\} \xrightarrow{a.s.} \mathop{\mathbb{E}}_{p_{\pi}(\boldsymbol{a}_{t:T-1},\boldsymbol{z}_{t:T-1})}\{f(\boldsymbol{a}_{t:T-1},\boldsymbol{z}_{t:T-1})\}. \tag{A.10}$$

Note,

$$\begin{aligned}
&\mathop{\mathbb{E}}_{p_{\mathcal{T}}(\boldsymbol{a}_{t:T},\boldsymbol{z}_{t:T})}\{f(\boldsymbol{a}_{t:T},\boldsymbol{z}_{t:T})\} \\
=&\mathop{\mathbb{E}}_{p_{\mathcal{T}}(\boldsymbol{a}_{t:T-1},\boldsymbol{z}_{t:T-1})}\left\{\mathop{\mathbb{E}}_{p_{\mathcal{T}}(\boldsymbol{a}_{T},\boldsymbol{z}_{T})}\{f(\boldsymbol{a}_{T},\boldsymbol{z}_{T},\boldsymbol{a}_{t:T-1},\boldsymbol{z}_{t:T-1})\}\right\} \\
\xrightarrow{a.s.}&\mathop{\mathbb{E}}_{p_{\pi}(\boldsymbol{a}_{t:T-1},\boldsymbol{z}_{t:T-1})}\left\{\mathop{\mathbb{E}}_{p_{\mathcal{T}}(\boldsymbol{a}_{T},\boldsymbol{z}_{T})}\{f(\boldsymbol{a}_{T},\boldsymbol{z}_{T},\boldsymbol{a}_{t:T-1},\boldsymbol{z}_{t:T-1})\}\right\}.
\end{aligned} \tag{A.11}$$

The *a.s.* convergence in (A.11) follows the assumption in (A.10). It is worth mentioning that both $\mathop{\mathbb{E}}_{p_{\mathcal{T}}(\boldsymbol{a}_{t:T},\boldsymbol{z}_{t:T})}\{f(\boldsymbol{a}_{t:T},\boldsymbol{z}_{t:T})\}$ and $\mathop{\mathbb{E}}_{p_{\pi}(\boldsymbol{a}_{t:T},\boldsymbol{z}_{t:T})}\{f(\boldsymbol{a}_{t:T},\boldsymbol{z}_{t:T})\}$ are also bounded functions on $(\boldsymbol{a}_{t:T-1},\boldsymbol{z}_{t:T-1})$. Therefore, in order to show,

$$\mathop{\mathbb{E}}_{p_{\mathcal{T}}(\boldsymbol{a}_{t:T},\boldsymbol{z}_{t:T})}\{f(\boldsymbol{a}_{t:T},\boldsymbol{z}_{t:T})\} \xrightarrow{a.s.} \mathop{\mathbb{E}}_{p_{\pi}(\boldsymbol{a}_{t:T},\boldsymbol{z}_{t:T})}\{f(\boldsymbol{a}_{t:T},\boldsymbol{z}_{t:T})\}, \tag{A.12}$$

(A.11) suggests it is suffice to have,

$$\mathop{\mathbb{E}}_{p_{\mathcal{T}}(\boldsymbol{a}_{T},\boldsymbol{z}_{T})}\{f(\boldsymbol{a}_{T},\boldsymbol{z}_{T},\boldsymbol{a}_{t:T-1},\boldsymbol{z}_{t:T-1})\} \xrightarrow{a.s.} \mathop{\mathbb{E}}_{p_{\pi}(\boldsymbol{a}_{T},\boldsymbol{z}_{T})}\{f(\boldsymbol{a}_{T},\boldsymbol{z}_{T},\boldsymbol{a}_{t:T-1},\boldsymbol{z}_{t:T-1})\}. \tag{A.13}$$

The same argument as (A.9) can be applied again to show the validity of (A.13). Recall that the belief action nodes in $\mathcal{T}$ are visited *i.o.* and spawn new measurements *i.o.* as $N \to \infty$, proved in Theorem 4.2. Therefore, $\tilde{V}(h_t|A_t, Z_t)$, a bounded function, is an unbiased estimate

of $V_\pi(h_t)$ with (A_t, Z_t) sampled from $p_{\mathcal{T}}(A_t, Z_t)$, *i.e.*,

$$\begin{aligned}
&\mathop{\mathbb{E}}_{p_{\mathcal{T}}(A_t,Z_t)}\Big\{\tilde{V}(h_t|A_t,Z_t)\Big\} \\
\xrightarrow[\text{as } N\to\infty]{a.s.} &\mathop{\mathbb{E}}_{p_\pi(A_t,Z_t)}\Big\{\tilde{V}(h_t|A_t,Z_t)\Big\}, \qquad \text{(A.12)}, \\
\xrightarrow[\text{as } K\to\infty]{a.s.} &\mathop{\mathbb{E}}_{p_\pi(A_t,Z_t)}\{V(h_t|A_t,Z_t)\}, \qquad \text{(A.7)}, \\
&= V_\pi(h_t), \qquad \text{(A.4)},
\end{aligned} \tag{A.14}$$

which completes the proof. □

Proof of Theorem 4.3. Given the results of Lemma A.3, it is left to show that the summation of discounted rewards is also an unbiased estimation at the traversed belief action nodes. Recall (4.6),

$$\tilde{V}(h_t\boldsymbol{a}_t|A_{t+1},Z_t) = \frac{1}{\eta}\sum_{i=0}^{K-1} p\big(Z_t|X_t^i,A_t\big)R(S_t). \tag{A.15}$$

Applying the theory of importance sampling again,

$$\begin{aligned}
\tilde{V}(h_t\boldsymbol{a}_t|A_{t+1},Z_t) &\xrightarrow{a.s.} \int_{X_t} p(X_t|A_t,Z_t)R(S_t)dX_t \\
&= \int_{X_t} p(X_t|A_t,Z_t)(r(\boldsymbol{x}_t,\boldsymbol{a}_t,\boldsymbol{x}_{t+1}) + \gamma R(S_{t+1}))dX_t,
\end{aligned} \tag{A.16}$$

shows the summation of stage reward and reward-to-go. The stage reward can be simplified to,

$$\begin{aligned}
&\int_{X_t} p(X_t|A_t,Z_t)r(\boldsymbol{x}_t,\boldsymbol{a}_t,\boldsymbol{x}_{t+1})dX_t \\
=&\int_{\boldsymbol{x}_t,\boldsymbol{x}_{t+1}} r(\boldsymbol{x}_t,\boldsymbol{a}_t,\boldsymbol{x}_{t+1})p(\boldsymbol{x}_t,\boldsymbol{x}_{t+1}|A_t,Z_t)d\boldsymbol{x}_t d\boldsymbol{x}_{t+1} \\
=&r(\boldsymbol{b}_t,\boldsymbol{a}_t,\boldsymbol{b}_{t+1}).
\end{aligned} \tag{A.17}$$

With some abuse of notation, $r(\cdot)$ denotes both the stage reward for $\boldsymbol{x}_t$, $\boldsymbol{x}_{t+1}$, and its expectation *w.r.t.* $p(\boldsymbol{x}_t,\boldsymbol{x}_{t+1}|A_t,Z_t)$. Meanwhile, the reward-to-go is,

$$\int_{X_t} p(X_t|A_t,Z_t)\cdot\gamma R(S_{t+1})dX_t = \gamma\tilde{V}(h_t\boldsymbol{a}_t\boldsymbol{z}_t|A_{t+1},Z_{t+1}). \tag{A.18}$$

Therefore, (A.15) can be rewritten as,

$$\tilde{V}(h_t\boldsymbol{a}_t|A_{t+1},Z_t) = r(\boldsymbol{b}_t,\boldsymbol{a}_t,\boldsymbol{b}_{t+1}) + \gamma\tilde{V}(h_t\boldsymbol{a}_t\boldsymbol{z}_t|A_{t+1},Z_{t+1}). \tag{A.19}$$

One should be reminded that (A_{t+1},Z_t) is sampled based on $p_{\mathcal{T}}(A_{t+1},Z_t)$. Then, the expectation of $\tilde{V}(h_t\boldsymbol{a}_t|A_{t+1},Z_t)$ is,

$$\begin{aligned}
\mathop{\mathbb{E}}_{p_{\mathcal{T}}(A_{t+1},Z_t)}\Big\{\tilde{V}(h_t\boldsymbol{a}_t|A_{t+1},Z_t)\Big\} &= r(\boldsymbol{b}_t,\boldsymbol{a}_t,\boldsymbol{b}_{t+1}) + \gamma\sum_{i=0}^{n-1}\mathop{\mathbb{E}}_{p_{\mathcal{T}}(A_{t+1},Z_{t+1})}\Big\{\tilde{V}(h_t\boldsymbol{a}_t\boldsymbol{z}_t^i|A_{t+1},Z_{t+1})\Big\} \\
&\xrightarrow{a.s.} r(\boldsymbol{b}_t,\boldsymbol{a}_t,\boldsymbol{b}_{t+1}) + \gamma\sum_{i=0}^{n-1}V_\pi\big(h_t\boldsymbol{a}_t\boldsymbol{z}_t^i\big) \\
&\xrightarrow{a.s.} r(\boldsymbol{b}_t,\boldsymbol{a}_t,\boldsymbol{b}_{t+1}) + \gamma\int_{\boldsymbol{z}_t}V_\pi\big(h_t\boldsymbol{a}_t\boldsymbol{z}_t^i\big)p(\boldsymbol{z}_t|\boldsymbol{a}_t)d\boldsymbol{z}_t \\
&= V_\pi(h_t\boldsymbol{a}_t)
\end{aligned} \tag{A.20}$$

The first *a.s.* convergence is based on Lemma A.3, while the second is, again, from the strong law of large numbers. Therefore, $\tilde{V}(h_t\boldsymbol{a}_t|A_{t+1},Z_t)$ is an unbiased estimate of $V_\pi(h_t\boldsymbol{a}_t)$. □

Comparing (A.6) and (A.15), one might find it striking that $\tilde{V}(h_t\boldsymbol{a}_t|A_{t+1},Z_t)$ is the same as $\tilde{V}(h_t|A_t,Z_t)$. The results should not be over-interpreted leading to the conclusion that $V_\pi(h_t\boldsymbol{a}_t) = V_\pi(h_t)$. The correct interpretation should be $\frac{1}{\eta}\sum_i^{K-1}P(Z_t|X_t^i,A_t)R(S_t^i)$ serves as an unbiased estimate for both $V_\pi(h_t\boldsymbol{a}_t)$ and $V_\pi(h_t)$. The discomfort should be resolved by considering that $V_\pi(h_t)$ is updated (implicitly) when any of its child belief-action nodes is visited, while updating $V_\pi(h_t\boldsymbol{a}_t)$ requires $h_t\boldsymbol{a}_t$, one child of h_t, to be visited.

Appendix B

Implementation Details of the Data-driven Traffic Model

B.1 Graph Neural Network (GNN) Setup

The implementation of RTGNN is setup with PyTorch[1]. Specially, the implementation of the GNN is based on the framework provided by PyTorch Geometric[2] (Fey and Lenssen, 2019).

As introduced in Sec. 6.1.1, we consider a discretized control space for the vehicles. The discretized acceleration space consists of 21 accelerations equally spaced from -8m/s^2 to 8m/s^2, while the discretized angular velocities consist of 21 angular velocities equally spaced from -0.5rad/s to 0.5rad/s. The fixed duration of the motion primitives is set to 0.5s, which agrees with the labeling frequency of the nuScenes dataset.

To define the graph, we consider agents within a square of 50m around the ego. More precisely, the boundaries of the square is set to be 40m ahead, 10m behind, and 25m on both sides of the ego vehicle. Edges of the graph are introduced based on spatial proximity with a radius of 25m. The max iterations of message passing, K in Alg. 5, is set to 2.

[1] `https://pytorch.org/`

[2] `https://pytorch-geometric.readthedocs.io/en/latest/`

In Alg. 5, $\texttt{CNN}_{\text{m}}$ contains three layers encoding a local map of size 100×100 into a vector of size 32. The three layers include 4, 8 and 16 filters with kernel size 5, 5, and 3 and stride 1. Each convolution layer is followed by relu activation and max pooling. $\texttt{CNN}_{\text{w}}$ consists of two layers encoding the possible future states and intention of an agent of size $21 \times 21 \times 6$ to a vector of size 32. Each layer uses 16 and 32 filters with kernel size 5 and stride 1. In $\texttt{CNN}_{\text{w}}$, a convolution layer is followed by leaky relu activation and max pooling. $\texttt{MLP}_{\text{m}}$ consists of 3 layers encoding the message from a vector of dimension 70 to 16. The number of hidden units in the first two layers of $\texttt{MLP}_{\text{m}}$ are 64 and 32. $\texttt{MLP}_{\text{q}}$ decodes the node feature and aggregated message from a vector fo size 68 to the intention which is of dimension 441. $\texttt{MLP}_{\text{q}}$ consists of 3 layers, the first two of which have 64 and 128 hidden units. In total, the network consists of 94,105 parameters.

B.2 Training Setup

To define the target intention in (6.5), we set $\sigma_x = \sigma_y = 5\text{e}{-2}\text{m}$, $\sigma_\theta = 1.75\text{e}{-2}\text{rad}$ and $\sigma_v = 0.1\text{m/s}$. As introduced in Sec. 6.1.2, each training samples accounts for four seconds of traffic, *i.e.* $t = 8$ in (6.7) given that data are labeled at 2Hz in nuScenes. To train the network, we use the Adam optimizer (Kingma and Ba, 2015) implemented in PyTorch with learning rate set to $2\text{e}{-3}$. The batch size for each training iteration is 16. We set the maximum epochs to be 50. It takes about 10 hours to train on a desktop with an Intel i9-9920X CPU (12 cores at 3.5GHz), 32G RAM, and a Nvidia 2080Ti GPU. With the same hardware configuration, it takes 0.176s on average to make one four-second sample prediction. The time required varies depending on the number of agents in the scene.

To alleviate the compounding error issue of recurrent neural networks, we apply scheduled sampling as in Bengio et al. (2015). Specifically, the sampling rate, the rate of using sampled vehicle states instead of ground truth states as the input to the next step, increases linearly from 0.0 at Epoch 10 to 0.5 at Epoch 30. The sampling rate maintains constant otherwise.

Bibliography

A.-a. Agha-mohammadi, S. Chakravorty, and N. M. Amato. FIRM: Sampling-based feedback motion-planning under motion uncertainty and imperfect measurements. *The International Journal of Robotics Research*, 33(2):268–304, 2014.

A.-a. Agha-mohammadi, S. Agarwal, S.-K. Kim, S. Chakravorty, and N. M. Amato. SLAP: Simultaneous localization and planning under uncertainty via dynamic replanning in belief space. *IEEE Transactions on Robotics*, 34(5):1195–1214, 2018.

R. Alterovitz, T. Simeon, and K. Goldberg. The stochastic motion roadmap: A sampling framework for planning with markov motion uncertainty. In *Proceedings of Robotics: Science and Systems*, Atlanta, GA, USA, June 2007.

P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

A. Bachrach, S. Prentice, R. He, P. Henry, A. S. Huang, M. Krainin, D. Maturana, D. Fox, and N. Roy. Estimation, planning, and mapping for autonomous flight using an RGB-D camera in GPS-denied environments. *The International Journal of Robotics Research*, 31 (11):1320–1343, 2012.

P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu. Relational inductive biases, deep learning, and graph networks, 2018.

S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, page 1171–1179. MIT Press, 2015.

D. P. Bertsekas et al. *Dynamic programming and optimal control: Vol. 1*. Athena scientific Belmont, 2017.

S. Brechtel, T. Gindele, and R. Dillmann. Probabilistic decision-making under uncertainty for autonomous driving using continuous POMDPs. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 392–399, 2014.

M. Brossard, S. Bonnabel, and J. Condomines. Unscented Kalman filtering on Lie groups. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2485–2491, 2017.

A. Bry and N. Roy. Rapidly-exploring Random Belief Trees for motion planning under uncertainty. In *2011 IEEE International Conference on Robotics and Automation*, pages 723–730, 2011.

H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. nuScenes: A multimodal dataset for autonomous driving. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11618–11628, 2020.

S. Casas, W. Luo, and R. Urtasun. Intentnet: Learning to predict intention from raw sensor data. In *Proceedings of The 2nd Conference on Robot Learning*, volume 87, pages 947–956. PMLR, 2018.

S. Casas, C. Gulino, R. Liao, and R. Urtasun. SPAGNN: Spatially-aware graph neural networks for relational behavior forecasting from sensor data. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9491–9497, 2020.

S. Casas, C. Gulino, S. Suo, K. Luo, R. Liao, and R. Urtasun. Implicit latent variable model for scene-consistent motion forecasting. In *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, 2020.

A. Censi, D. Calisi, A. De Luca, and G. Oriolo. A bayesian framework for optimal motion planning with uncertainty. In *2008 IEEE International Conference on Robotics and Automation*, pages 1798–1805, 2008.

Y. Chai, B. Sapp, M. Bansal, and D. Anguelov. MultiPath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction. In *Proceedings of the Conference on Robot Learning*, pages 86–99, 2019.

M.-F. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan, and J. Hays. Argoverse: 3d tracking and forecasting with rich maps. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8740–8749, 2019.

P. Chaudhari, S. Karaman, D. Hsu, and E. Frazzoli. Sampling-based algorithms for continuous-time POMDPs. In *2013 American Control Conference*, pages 4604–4610, 2013.

S. M. Chaves, J. M. Walls, E. Galceran, and R. M. Eustice. Risk aversion in belief-space planning under measurement acquisition uncertainty. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2079–2086, 2015.

H. M. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, S. Thrun,

and R. C. Arkin. *Principles of robot motion: theory, algorithms, and implementation.* MIT press, 2005.

A. Couëtoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard. Continuous upper confidence trees. In *Learning and Intelligent Optimization*, pages 433–445. Springer Berlin Heidelberg, 2011.

H. Cui, V. Radosavljevic, F. Chou, T. Lin, T. Nguyen, T. Huang, J. Schneider, and N. Djuric. Multimodal trajectory predictions for autonomous driving using deep convolutional networks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2090–2096, 2019.

S. Cyrill. Robotics dataset. `http://www2.informatik.uni-freiburg.de/~stachnis/datasets.html`, 2020. [Online; accessed 02-September-2020].

W. Ding, L. Zhang, J. Chen, and S. Shen. Safe trajectory generation for complex urban environments using spatio-temporal semantic corridor. *IEEE Robotics and Automation Letters*, 4(3):2997–3004, 2019.

D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel. Practical search techniques in path planning for autonomous driving. *Ann Arbor*, 1001(48105):18–80, 2008.

N. E. Du Toit and J. W. Burdick. Robot motion planning in dynamic, uncertain environments. *IEEE Transactions on Robotics*, 28(1):101–115, 2012.

N. Evestedt, E. Ward, J. Folkesson, and D. Axehill. Interaction aware trajectory planning for merge scenarios in congested traffic situations. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 465–472, 2016.

D. Falanga, P. Foehn, P. Lu, and D. Scaramuzza. PAMPC: Perception-aware model predictive control for quadrotors. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–8, 2018.

M. Fey and J. E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

E. Galceran, A. G. Cunningham, R. M. Eustice, and E. Olson. Multipolicy decision-making for autonomous driving via changepoint-based behavior prediction: Theory and experiment. *Autonomous Robots*, 41(6):1367–1382, 2017.

J. Gao, C. Sun, H. Zhao, Y. Shen, D. Anguelov, C. Li, and C. Schmid. VectorNet: Encoding HD maps and agent dynamics from vectorized representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

J. Geweke. Bayesian inference in econometric models using Monte Carlo integration. *Econometrica: Journal of the Econometric Society*, pages 1317–1339, 1989.

G. Grimmett and D. Stirzaker. *Probability and random processes*. Oxford university press, 2020.

G. Grisetti, C. Stachniss, W. Burgard, et al. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE transactions on Robotics*, 23(1):34, 2007.

W. S. L. Hanna Kurniawati, David Hsu. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proceedings of Robotics: Science and Systems IV*, Zurich, Switzerland, June 2008.

P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2): 100–107, 1968.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997.

J. Hoey and P. Poupart. Solving POMDPs with continuous or large discrete observation spaces. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, page 1332–1338, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.

J. Houston, G. Zuidhof, L. Bergamini, Y. Ye, L. Chen, A. Jain, S. Omari, V. Iglovikov, and P. Ondruska. One thousand and one hours: Self-driving motion prediction dataset, 2020.

C. Hubmann, J. Schulz, M. Becker, D. Althoff, and C. Stiller. Automated driving in uncertain environments: Planning with interaction and uncertain maneuver prediction. *IEEE Transactions on Intelligent Vehicles*, 3(1):5–17, 2018a.

C. Hubmann, J. Schulz, G. Xu, D. Althoff, and C. Stiller. A belief state planner for interactive merge maneuvers in congested traffic. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 1617–1624, 2018b.

C. Hubmann, N. Quetschlich, J. Schulz, J. Bernhard, D. Althoff, and C. Stiller. A POMDP maneuver planner for occlusions in urban scenarios. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 2172–2179, 2019.

E. E. Hunter, E. W. Brink, E. B. Steager, and V. Kumar. Toward soft micro bio robots for cellular and chemical delivery. *IEEE Robotics and Automation Letters*, 3(3):1592–1599, 2018.

V. Indelman, L. Carlone, and F. Dellaert. Planning in the continuous domain: A generalized belief space approach for autonomous navigation in unknown environments. *The International Journal of Robotics Research*, 34(7):849–882, 2015.

L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99–134, 1998. ISSN 0004-3702.

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

M. Kennedy, K. Schmeckpeper, D. Thakur, C. Jiang, V. Kumar, and K. Daniilidis. Autonomous precision pouring from unknown containers. *IEEE Robotics and Automation Letters*, 4(3):2317–2324, 2019.

A. Kesting, M. Treiber, and D. Helbing. General lane-changing model MOBIL for car-following models. *Transportation Research Record*, 1999(1):86–94, 2007.

S. Khandelwal, W. Qi, J. Singh, A. Hartnett, and D. Ramanan. What-If motion prediction for autonomous driving, 2020.

D. P. Kingma and J. L. Ba. Adam: A method for stochastic gradient descent. In *ICLR: International Conference on Learning Representations*, pages 1–15, 2015.

L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

D. Kopitkov and V. Indelman. General-purpose incremental covariance update and efficient belief space planning via a factor-graph propagation action tree. *The International Journal of Robotics Research*, 38(14):1644–1673, 2019.

V. Kuleshov and D. Precup. Algorithms for multi-armed bandit problems. *arXiv preprint arXiv:1402.6028*, 2014.

H. Kurniawati and V. Yadav. *An Online POMDP Solver for Uncertainty Planning in Dynamic Environment*, pages 611–629. Springer International Publishing, 2016.

H. Kurniawati, Y. Du, D. Hsu, and W. S. Lee. Motion planning under uncertainty for robotic tasks with long time horizons. *The International Journal of Robotics Research*, 30(3):308–323, 2011.

M. Lauri and R. Ritala. Planning for robotic exploration based on forward simulation. *Robotics and Autonomous Systems*, 83:15–31, 2016.

S. M. LaValle. *Planning algorithms*. Cambridge university press, 2006.

W. Li and E. Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *2004 International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, pages 222–229, 2004.

Y. Li, Z. Littlefield, and K. E. Bekris. Asymptotically optimal sampling-based kinodynamic planning. *The International Journal of Robotics Research*, 35(5):528–564, 2016.

M. Liang, B. Yang, R. Hu, Y. Chen, R. Liao, S. Feng, and R. Urtasun. Learning lane graph representations for motion forecasting. In *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, 2020.

M. Likhachev, G. J. Gordon, and S. Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in neural information processing systems*, pages 767–774, 2004.

S. Liu, N. Atanasov, K. Mohta, and V. Kumar. Search-based motion planning for quadrotors using linear quadratic minimum time control. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2872–2879, 2017a.

S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar. Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments. *IEEE Robotics and Automation Letters*, 2(3):1688–1695, 2017b.

W. Liu, S.-W. Kim, S. Pendleton, and M. H. Ang. Situation-aware decision making for autonomous driving on urban road using online POMDP. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 1126–1133, 2015.

X. Liu, S. W. Chen, C. Liu, S. S. Shivakumar, J. Das, C. J. Taylor, J. Underwood, and V. Kumar. Monocular camera based fruit counting and mapping with semantic data association. *IEEE Robotics and Automation Letters*, 4(3):2296–2303, 2019.

A. Martinelli. Vision and imu data fusion: Closed-form solutions for attitude, speed, absolute scale, and bias determination. *IEEE Transactions on Robotics*, 28(1):44–60, 2012.

P. S. Maybeck. *Stochastic models, estimation, and control*. Academic press, 1982.

D. Mayne and H. Michalska. Receding horizon control of nonlinear systems. *IEEE Transactions on Automatic Control*, 35(7):814–824, 1990.

M. McNaughton, C. Urmson, J. M. Dolan, and J.-W. Lee. Motion planning for autonomous driving with a conformal spatiotemporal lattice. In *2011 IEEE International Conference on Robotics and Automation*, pages 4889–4895, 2011.

N. A. Melchior and R. Simmons. Particle RRT for path planning with uncertainty. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1617–1624, 2007.

K. Messaoud, N. Deo, M. M. Trivedi, and F. Nashashibi. Trajectory prediction for autonomous driving based on multi-head attention with joint agent-map representation, 2020.

I. D. Miller, F. Cladera, A. Cowley, S. S. Shivakumar, E. S. Lee, L. Jarin-Lipschitz, A. Bhat, N. Rodrigues, A. Zhou, A. Cohen, A. Kulkarni, J. Laney, C. J. Taylor, and V. Kumar. Mine tunnel exploration using multiple quadrupedal robots. *IEEE Robotics and Automation Letters*, 5(2):2840–2847, 2020.

K. Mohta, K. Sun, S. Liu, M. Watterson, B. Pfrommer, J. Svacha, Y. Mulgaonkar, C. J. Taylor, and V. Kumar. Experiments in fast, autonomous, GPS-denied quadrotor flight. In

2018 IEEE International Conference on Robotics and Automation (ICRA), pages 7832–7839, 2018a.

K. Mohta, M. Watterson, Y. Mulgaonkar, S. Liu, C. Qu, A. Makineni, K. Saulnier, K. Sun, A. Zhu, J. Delmerico, K. Karydis, N. Atanasov, G. Loianno, D. Scaramuzza, K. Daniilidis, C. J. Taylor, and V. Kumar. Fast, autonomous flight in GPS-denied and cluttered environments. *Journal of Field Robotics*, 35(1):101–120, 2018b.

M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, D. Johnston, S. Klumpp, D. Langer, A. Levandowski, J. Levinson, J. Marcil, D. Orenstein, J. Paefgen, I. Penny, A. Petrovskaya, M. Pflueger, G. Stanek, D. Stavens, A. Vogt, and S. Thrun. Junior: The Stanford entry in the urban challenge. *Journal of Field Robotics*, 25(9):569–597, 2008.

H. Nishimura and M. Schwager. SACBP: Belief space planning for continuous-time dynamical systems via stochastic sequential action control. In *International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, pages 267–283, 2018.

J. Nocedal and S. Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1(1):33–55, 2016.

S. Paquet, L. Tobin, and B. Chaib-Draa. Real-time decision making for large POMDPs. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 450–455. Springer, 2005.

S. Patil, Y. Duan, J. Schulman, K. Goldberg, and P. Abbeel. Gaussian belief space planning with discontinuities in sensing domains. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6483–6490, 2014.

A. A. Pereira, J. Binney, G. A. Hollinger, and G. S. Sukhatme. Risk-aware path planning for autonomous underwater vehicles using predictive ocean models. *Journal of Field Robotics*, 30(5):741–762, 2013.

K. B. Petersen and M. S. Pedersen. The matrix cookbook (version: November 15, 2012), 2012.

T. Phan-Minh, E. C. Grigore, F. A. Boulton, O. Beijbom, and E. M. Wolff. CoverNet: Multimodal behavior prediction using trajectory sets. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, page 1025–1030. Morgan Kaufmann Publishers Inc., 2003.

R. Platt, R. Tedrake, L. Kaelbling, and T. Lozano-Perez. Belief space planning assuming maximum likelihood observations. In *Proceedings of Robotics: Science and Systems*, Zaragoza, Spain, June 2010.

F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat. Comparing ICP variants on real-world data sets. *Autonomous Robots*, 34(3):133–148, 2013.

J. M. Porta, N. Vlassis, M. T. Spaan, and P. Poupart. Point-based value iteration for continuous POMDPs. *Journal of Machine Learning Resesearch*, 7:2329–2367, 2006. ISSN 1532-4435.

S. Prentice and N. Roy. The belief roadmap: Efficient planning in belief space by factoring the covariance. *The International Journal of Robotics Research*, 28(11-12):1448–1465, 2009.

M. Quigley, K. Mohta, S. S. Shivakumar, M. Watterson, Y. Mulgaonkar, M. Arguedas, K. Sun, S. Liu, B. Pfrommer, V. Kumar, and C. J. Taylor. The open vision computer: An integrated sensing and compute system for mobile robots. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 1834–1840, 2019.

M. Rafieisakhaei, S. Chakravorty, and P. R. Kumar. T-LQG: Closed-loop belief space planning via trajectory-optimized LQG. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 649–656, 2017.

N. Rhinehart, R. McAllister, K. Kitani, and S. Levine. PRECOG: Prediction conditioned on goals in visual multi-agent settings. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

S. Ross and B. Chaib-Draa. AEMS: An anytime online search algorithm for approximate policy refinement in large POMDPs. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, page 2592–2598. Morgan Kaufmann Publishers Inc., 2007.

S. Ross, J. Pineau, S. Paquet, and B. Chaib-Draa. Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research*, 32:663–704, 2008.

S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15, pages 627–635. PMLR, Apr 2011.

G. Salavasidis, A. Munafò, C. A. Harris, T. Prampart, R. Templeton, M. Smart, D. T. Roper, M. Pebody, S. D. McPhail, E. Rogers, and A. B. Phillips. Terrain-aided navigation for long-endurance and deep-rated autonomous underwater vehicles. *Journal of Field Robotics*, 36(2):447–474, 2019. doi: https://doi.org/10.1002/rob.21832.

T. Salzmann, B. Ivanovic, P. Chakravarty, and M. Pavone. Trajectron++: Dynamically-

feasible trajectory forecasting with heterogeneous data. In *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, 2020.

A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia. Graph networks as learnable physics engines for inference and control. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 4470–4479. PMLR, Jul 2018.

T. Shan and B. Englot. Belief roadmap search: Advances in optimal and efficient planning under uncertainty. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5318–5325, 2017.

G. Shani, J. Pineau, and R. Kaplow. A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems*, 27(1):1–51, 2013.

D. Silver and J. Veness. Monte-Carlo planning in large POMDPs. In *Advances in neural information processing systems*, pages 2164–2172, 2010.

T. Smith and R. Simmons. Heuristic search value iteration for POMDPs. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, page 520–527. AUAI Press, 2004.

K. Sohn, H. Lee, and X. Yan. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems*, volume 28, pages 3483–3491. Curran Associates, Inc., 2015.

E. J. Sondik. *The optimal control of partially observable Markov processes*. Stanford University, 1971.

I. Spasojevic, V. Murali, and S. Karaman. Perception-aware time optimal path parameterization for quadrotors. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3213–3219, 2020.

I. A. Şucan, M. Moll, and L. E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. `https://ompl.kavrakilab.org`.

K. Sun and V. Kumar. Belief space planning for mobile robots with range sensors using iLQG. *IEEE Robotics and Automation Letters*, 6(2):1902–1909, 2021.

K. Sun, K. Mohta, B. Pfrommer, M. Watterson, S. Liu, Y. Mulgaonkar, C. J. Taylor, and V. Kumar. Robust stereo visual inertial odometry for fast autonomous flight. *IEEE Robotics and Automation Letters*, 3(2):965–972, 2018.

K. Sun, B. Schlotfeldt, S. Chaves, P. Martin, G. Mandhyan, and V. Kumar. Feedback enhanced motion planning for autonomous vehicles. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2126–2133, 2020a.

K. Sun, B. Schlotfeldt, G. J. Pappas, and V. Kumar. Stochastic motion planning under partial observability for mobile robots with continuous range measurements. *IEEE Transactions on Robotics*, 37(3):979–995, 2021.

P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov. Scalability in perception for autonomous driving: Waymo Open Dataset. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2443–2451, 2020b.

W. Sun, J. van den Berg, and R. Alterovitz. Stochastic extended LQR for optimization-based motion planning under uncertainty. *IEEE Transactions on Automation Science and Engineering*, 13(2):437–447, 2016.

Z. N. Sunberg and M. J. Kochenderfer. Online algorithms for POMDPs with continuous state, action, and observation spaces. In *Twenty-Eighth International Conference on Automated Planning and Scheduling*, 2018.

Z. N. Sunberg, C. J. Ho, and M. J. Kochenderfer. The value of inferring the internal state of traffic participants for autonomous freeway driving. In *2017 American Control Conference (ACC)*, pages 3004–3010, 2017.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts. LQR-trees: Feedback motion planning via sums-of-squares verification. *The International Journal of Robotics Research*, 29(8):1038–1052, 2010.

D. Thakur, G. Loianno, W. Liu, and V. Kumar. Nuclear environments inspection with micro aerial vehicles: Algorithms and experiments. In *Proceedings of the 2018 International Symposium on Experimental Robotics*, pages 191–200. Springer International Publishing, 2020.

S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT press, 2005.

E. Tolstaya, R. Mahjourian, C. Downey, B. Vadarajan, B. Sapp, and D. Anguelov. Identifying driver interactions via conditional behavior prediction, 2021.

M. Treiber and A. Kesting. Traffic flow dynamics. *Traffic Flow Dynamics: Data, Models and Simulation, Springer-Verlag Berlin Heidelberg*, 2013.

J. N. Tsitsiklis. On the convergence of optimistic policy iteration. *The Journal of Machine Learning Research*, 3:59–72, 2003. doi: 10.1162/153244303768966102.

C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. N. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. M. Howard, S. Kolski, A. Kelly, M. Likhachev, M. McNaughton, N. Miller, K. Peterson, B. Pilnick, R. Rajkumar,

P. Rybski, B. Salesky, Y.-W. Seo, S. Singh, J. Snider, A. Stentz, W. R. Whittaker, Z. Wolkowicki, J. Ziglar, H. Bae, T. Brown, D. Demitrish, B. Litkouhi, J. Nickolaou, V. Sadekar, W. Zhang, J. Struble, M. Taylor, M. Darms, and D. Ferguson. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.

J. van den Berg, P. Abbeel, and K. Goldberg. LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information. *The International Journal of Robotics Research*, 30(7):895–913, 2011.

J. van den Berg, S. Patil, and R. Alterovitz. Efficient approximate value iteration for continuous Gaussian POMDPs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 26(1), 2012a.

J. van den Berg, S. Patil, and R. Alterovitz. Motion planning under uncertainty using iterative local optimization in belief space. *The International Journal of Robotics Research*, 31(11):1263–1278, 2012b.

M. P. Vitus and C. J. Tomlin. On feedback design and risk allocation in chance constrained control. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pages 734–739, 2011a.

M. P. Vitus and C. J. Tomlin. Closed-loop belief space planning for linear, Gaussian systems. In *2011 IEEE International Conference on Robotics and Automation*, pages 2152–2159, 2011b.

M. Watterson, S. Liu, K. Sun, T. Smith, and V. Kumar. Trajectory optimization on manifolds with applications to SO(3) and R3xS2. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018.

K. J. Wu and S. I. Roumeliotis. Unobservable directions of VINS under special motions. *Department of Computer Science & Engineering, University of of Minnesota: Minneapolis, MN, USA*, 2016.

N. Ye, A. Somani, D. Hsu, and W. S. Lee. DESPOT: Online pomdp planning with regularization. *Journal of Artificial Intelligence Research*, 58:231–266, 2017.

W. Zeng, W. Luo, S. Suo, A. Sadat, B. Yang, S. Casas, and R. Urtasun. End-to-end interpretable neural motion planner. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

W. Zeng, M. Liang, R. Liao, and R. Urtasun. LaneRCNN: Distributed representations for graph-centric motion forecasting, 2021.

J. Zhang, M. Kaess, and S. Singh. On degeneracy of optimization-based state estimation problems. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 809–816, 2016.

L. Zhang, W. Ding, J. Chen, and S. Shen. Efficient uncertainty-aware decision-making for automated driving using guided branching. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3291–3297, 2020.

H. Zhao, J. Gao, T. Lan, C. Sun, B. Sapp, B. Varadarajan, Y. Shen, Y. Shen, Y. Chai, C. Schmid, C. Li, and D. Anguelov. TNT: Target-driven trajectory prediction. In *Conference on Robot Learning (CoRL)*, November 2020.

B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence*, volume 3, page 1433–1438. AAAI Press, 2008.

T. Özaslan, G. Loianno, J. Keller, C. J. Taylor, V. Kumar, J. M. Wozencraft, and T. Hood. Autonomous navigation and mapping for inspection of penstocks and tunnels with mavs. *IEEE Robotics and Automation Letters*, 2(3):1740–1747, 2017.