Non-parametric Models for Long-term Autonomy

by

Acshi K. Haggenmiller

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Robotics)
in the University of Michigan
2022

Doctoral Committee:

Professor Edwin Olson, Chair
Professor Ella M. Atkins
Associate Professor Dmitry Berenson
Professor Odest Chadwicke Jenkins

Acshi K. Haggenmiller
acshikh@umich.edu
ORCID iD: 0000-0002-4854-5793

© Acshi K. Haggenmiller 2022

ACKNOWLEDGMENTS

Thank you to my advisor Ed Olson for giving me ideas, inspiration, support, and focused attention as I have learned what it means to be a researcher. Thank you to my other committee members (Ella Atkins, Dmitry Berenson, and Chad Jenkins) for their valuable feedback on and investment in this material. Thank you to my labmates who helped introduce me to robotics research. In particular, thank you to my labmate and collaborator Max Krogius, who provided timely and apt feedback to so many of my ideas and helped me avoid getting caught up on minor misunderstandings.

Most importantly, thank you to my wife Enid Law for the love, confidence, and support she has given to me in this journey.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE

LIST OF TABLES

TABLE

ABSTRACT

In this thesis we propose novel estimation techniques for localization and planning problems, which are key challenges in long-term autonomy. We take inspiration in our methods from non-parametric estimation and use tools such as kernel density estimation, non-linear least-squares optimization, binary masking, and random sampling. We show that these methods, by avoiding explicit parametric models, outperform existing methods that use them. Despite the seeming differences between localization and planning, we demonstrate in this thesis that the problems share core structural similarities. When real or simulation-sampled measurements are expensive, noisy, or high variance, non-parametric estimation techniques give higher-quality results in less time.

We first address two localization problems. In order to permit localization with a set of ad hoc-placed radios, we propose an ultra-wideband (UWB) graph realization system to localize the radios. Our system achieves high accuracy and robustness by using kernel density estimation for measurement probability densities, by explicitly modeling antenna delays, and by optimizing this combination with a non-linear least squares formulation. Next, in order to then support robotic navigation, we present a flexible system for simultaneous localization and mapping (SLAM) that combines elements from both traditional dense metric SLAM and topological SLAM, using a binary "masking function" to focus attention. This masking function controls which lidar scans are available for loop closures. We provide several masking functions based on approximate topological class detectors.

We then examine planning problems in the final chapter and in the appendix. In order to plan with uncertainty around multiple dynamic agents, we describe Monte-Carlo Policy-Tree Decision Making (MCPTDM), a framework for efficiently computing policies in partially-observable, stochastic, continuous problems. MCPTDM composes a sequence of simpler closed-loop policies and uses marginal action costs and particle repetition to improve cost estimates and sample efficiency by reducing variance. Finally, in the appendix we explore Learned Similarity Monte-Carlo Planning (LSMCP), where we seek to enhance the sample efficiency of partially observable Monte Carlo tree search-based planning by taking advantage of similarities in the final outcomes of similar states and actions. We train a multilayer perceptron to learn a similarity function which we then use to enhance value estimates in the planning.

Collectively, we show in this thesis that non-parametric methods promote long-term autonomy by reducing error and increasing robustness across multiple domains.

CHAPTER 1

Introduction

The central idea of this thesis is that non-parametric models can help provide the robustness necessary for long-term autonomy. The central challenge in long-term autonomy is that errors can accumulate over time, eventually leading to failures. In this thesis, we will show that non-parametric models can help to reduce the magnitude of errors across a variety of domains, including localization and planning.

In some cases, error can be modeled and reasoned about parametrically, such as with a Gaussian [1][2] or Gaussian mixture [3] model. While methods such as the extended Kalman filter (EKF) perform linearization to handle non-linear models in a Gaussian framework, linearization error in the EKF may cause the solution to diverge. To address this issue, alternative methods have been proposed, such as the unscented Kalman filter [4] and the extended finite impulse response filter [5][6]. In cases where a parametric model is more complex with interacting variables and their distributions, variational Bayesian inference can be used to estimate the latent model variables [7]. A primary benefit of parametric models is that they often afford closed-form solutions. And some problems with local minima can even be relaxed into a convex form to help with finding the global minimum. However, when their assumptions about the underlying data and error distributions are incorrect, they provide no guarantees about their solution accuracy. Closed-form solutions can be very attractive because they often have very predictable and low computational requirements, but many real world problems are very difficult or impossible to reduce into a form suitable for these solvers. Furthermore, the longer a robot is operating in the real world at a time, the more likely it is that the assumptions behind these parametric models will be violated.

By comparison, non-parametric methods can handle arbitrary distributions and do not rely on the same assumptions for their performance. For example, by extending from a fixed number to a potentially infinite number of Gaussians, the infinite Gaussian mixture model can approximate any distribution arbitrarily well [8][9]. Gaussian process regression is another non-parametric method based on distributions of functions [10], and similar to other non-parametric methods, can be used in a maximum likelihood formulation with Gauss-Newton non-linear optimization [11]. Non-parametric methods are also important when the unknown distribution is complex, highly variable,

and where we want to avoid approximation error. A key challenge with non-parametric methods is controlling their computational efficiency. Gaussian process regression has computational complexity cubic in the number of input points and non-linear optimization, needing an iterative solver, is generally much slower than linear optimization. Non-parametric methods also tend to have more local minima, which may require additional complexity to ensure they find good solutions. Despite these difficulties, by avoiding model assumptions we help address the increased uncertainty and complexity of real-world long-term autonomy.

Throughout this thesis, we show how these ideas can be applied to key problems in perception and planning. Our guiding principle is to avoid all model assumptions that are not strictly required.

1.1 Non-parametric Error Modeling for Ultra-wideband Localization Networks

We apply non-parametric estimation techniques to localization in chapter 2 [12], showing that because real-world ultra-wideband (UWB) radios can produce high-variance non-Gaussian measurements, these techniques are needed to enable accurate network localization in highly adversarial environments. Our non-parametric method solves the network localization problem for a set of stationary UWB radios by preserving the direct measurements between each pair and then using each set of measurements as a particle representation of the true measurement probability distributions using kernel density estimation. Our non-parametric model allowed us to use a non-linear measurement model with explicit coupling of each radio's position and *antenna delay*, an internal radio parameter.

We performed experiments with four real-world datasets and released both these datasets as well as our source code for reproducibility and comparison. Figure 1.1 shows that even under nearly-Gaussian conditions, this coupled non-parametric method outperformed baseline methods that make assumptions about Gaussian or Gaussian-mixture distributions. Under noisier conditions with non-Gaussian distributions, this method far outperforms the baselines.

A standard commercial practice when using UWB radios for localization is to rigidly affix each radio to a location and then precisely measure that position. By using our method to avoid this manual calibration, an autonomous system can continuously recalibrate UWB radio positions. With this sort of proprioception/self-awareness, when localization radios are moved either intentionally or accidentally, the system should be able to automatically recover, enabling longer periods of autonomy.

The contributions of chapter 2:

1. Explicit coupled modeling of node antenna delays and locations in a maximum a posteriori

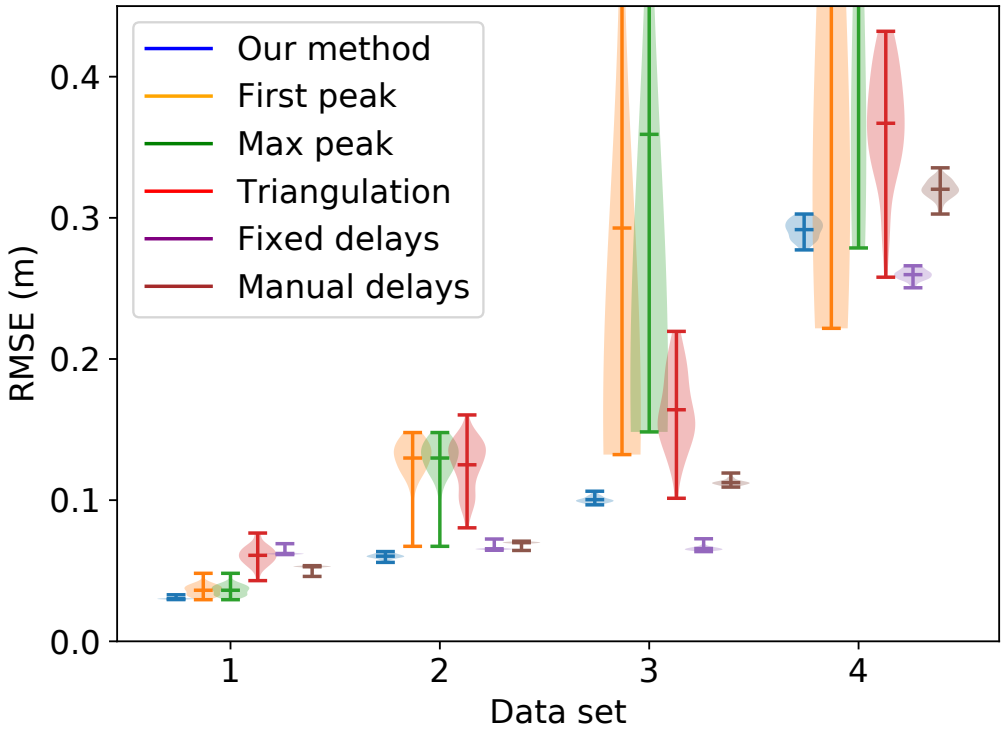

Figure 1.1: Performance comparison of our method and its variations with three baseline methods. Our non-parametric method and its fixed and manual delay variations far outperform the three baselines across increasingly non-line-of-sight datasets. High uncertainty in datasets 3 and 4 cause the full method to perform worse than the fixed and manual delay methods because the system is under-constrained.

formulation.

2. A framework for optimizing with non-parametric kernel density estimate error models for each pairwise radio link.

3. Four open-access datasets for comparisons with our method, along with the complete source code for running our method and generating figures.

1.2 The Masked Mapper: Masked Metric Mapping

We use ideas from topological mapping to solve the simultaneous localization and mapping (SLAM) problem in chapter 3 [13], without imposing the full constraints of accurate topological labels. Our flexible mapping scheme uses a masking function as an abstraction to focus the attention of a pose graph SLAM system. The masking function takes the robot's observations and returns true if the robot is in an important location. This enables us to plug-in approximate/simplified topological

class detectors (e.g. for hallway junctions, or for where the class changes) as the masking function, and focus on these areas for finding loop closures. State-of-the-art methods in SLAM generate dense metric lidar maps, creating precise maps at a high computational cost by storing lidar scans for each pose node and continually attempting to close loops. In many cases, trying to always make loop closures is unnecessary for localization and even risky because of perceptual aliasing and false positives. By masking out these less useful positions, our method can create more accurate maps despite performing far fewer scan matches.

We evaluate our system with three simple mask functions on a 2.5 km trajectory with significant angular drift. We compare the number of scan matches performed under each mask as well as the accuracy of the loop closures. Figure 1.2 shows that with very large loops, large straight hallways subject to perceptual aliasing, and large odometry drift, our method is effective at recovering an accurate map.

When a mobile robot loses its localization or "gets lost" and is unable to relocalize on its own, a human operator will need to intervene. Human environments are subject to constant change and this poses a particular challenge to a robot's ability to recognize previous locations. Taking advantage of masked mapping, a robot designer can design masking functions that prioritize features that are both distinctive and unlikely to change or that will still be recognizable even with expected changes. By masking away less reliable lidar scans, a robot will be able to operate for longer in a changing environment without losing localization or needing human intervention. In addition, computationally-efficient masked mapping leaves extra computing power for more effective autonomy in the robot's other subsystems.

The contributions of chapter 3:

1. A flexible approach to masking a high performance SLAM system to lower computational cost and minimize the chance of errors due to perceptual aliasing.

2. Two effective masking functions based on detecting openings and intersections that improve map quality while performing five times fewer scan matches than an "always true" mask function.

3. Demonstration of our system on a $2.5\,\mathrm{km}$ dataset.

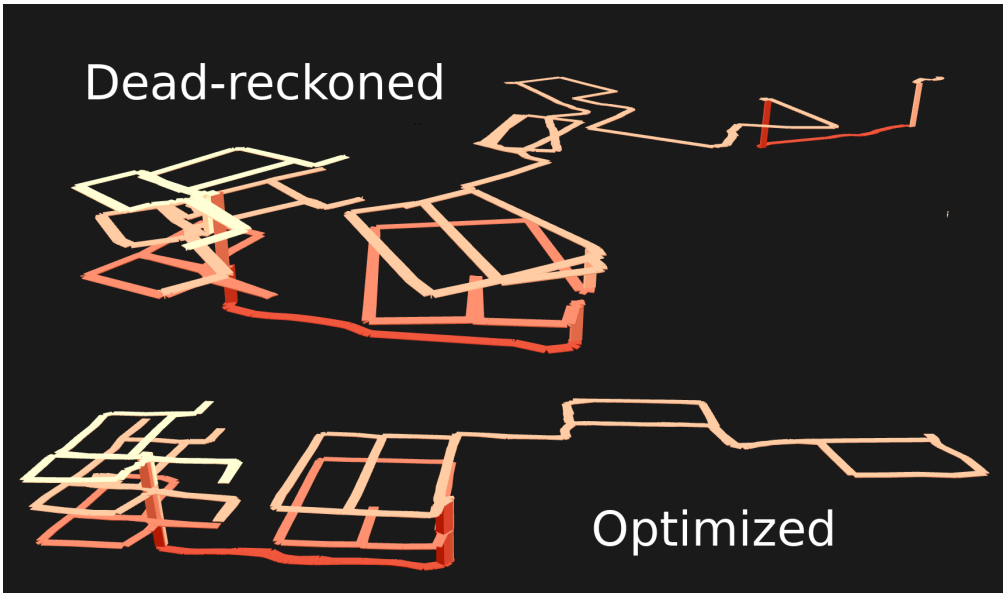

Figure 1.2: Comparison between our optimized map and the odometry-only map. We get more effective loop closures by using ideas from topological mapping. Before loop closures and optimization, given the large size of some of these loops, we had as much as 30 degrees of rotation error (top). After loop closures found with our masking function and optimization, we recover an accurate map (bottom).

1.3 Monte-Carlo Policy-Tree Decision Making (MCPTDM)

In chapter 4 [14], we apply non-parametric estimation to planning in domains such as autonomous driving, where rollouts are expensive and the costs/rewards for each action are high-variance. We present Monte-Carlo policy-tree decision making (MCPTDM), a framework for planning with uncertainty that uses Monte-Carlo tree search (MCTS) to solve the QMDP [15] simplification of a partially observable Markov decision process (POMDP). Due to the high computational cost of rollouts, we focus on sample efficiency by using KL-UCB [16] over the standard upper-confidence bound (UCB). We also introduce two innovations to MCTS, marginal action costs and fairness-based particle repetition, that also increase sample efficiency.

We perform experiments on both a simplified abstract scenario and also on a self-driving car scenario, and show that both of these innovations significantly improve performance. Figure 1.3 shows that MCPTDM outperforms a number of comparable methods on the self-driving task. We again release all of my source code for replication and comparison.

From a planning perspective, increasing the efficiency of robot planning allows for planning further into the future and for considering more possible plans and situations. Planning is in general very hard because of the curse of dimensionality in plans and belief. However much we can improve planning efficiency, more effective plans will mean that a robot is more likely to achieve

its goals without needing an intervention or human help.

The contributions of chapter 4:

1. Monte-Carlo Policy-Tree Decision Making (MCPTDM), which allows an agent to efficiently compute policies in partially-observable and stochastic problems having either discrete or continuous action and state spaces. It does this by composing its policy from a sequence of simpler policies.

2. The enhancements of marginal action cost (*MAC*) estimation, which improves UCB-based tree exploration and final action selection, and "particle repetition", which improves fairness in cost estimation by reducing the effect of unlucky or unusual initial conditions.

3. Validation of our MAC estimation and particle repetition improvements on an abstract task and evaluation and comparison with MPDM [17] and EUDM [18] on an autonomous self-driving task.

4. Openly-released source code capable of reproducing all the figures and evaluation results from this chapter for full replication of our results and further extensions.

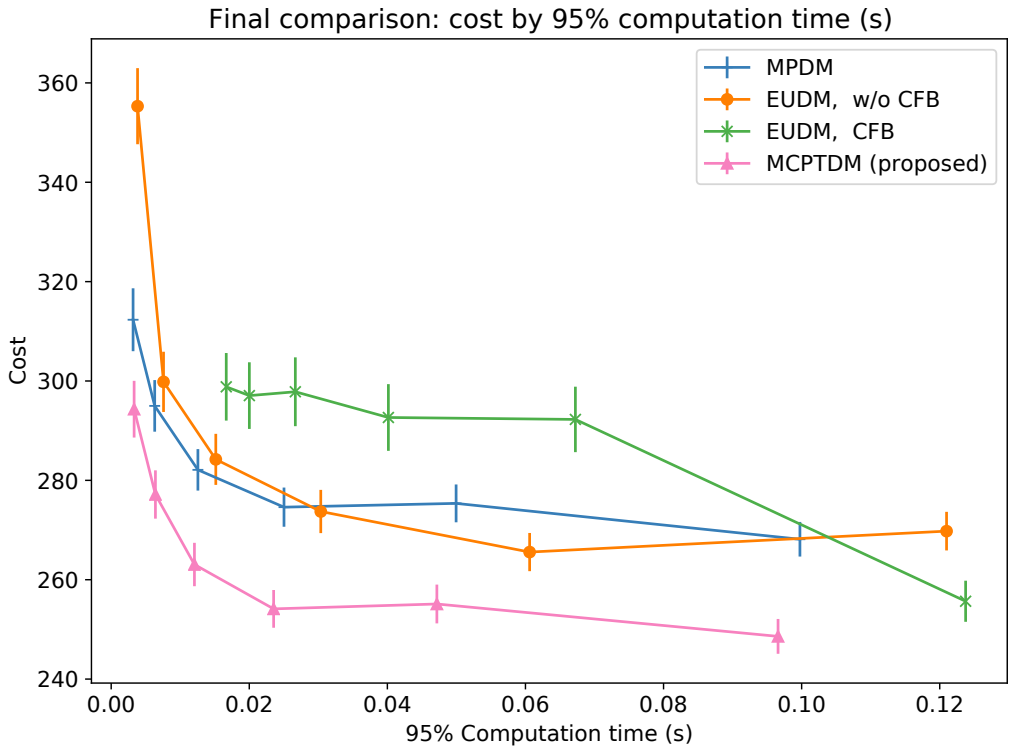

Figure 1.3: Final comparison of computational time against planning cost for MCPTDM and our baseline methods. MCPTDM achieves either significantly lower final cost or significantly lower computational time than either EUDM [18] or MPDM [17].

1.4 Learned Similarity Monte Carlo Planning (LSMCP)

In the appendix, we start a line of research on learning a similarity function with a multilayer perceptron to enhance the sample-efficiency of partially-observable Monte Carlo tree search-based planning. Planning for black-box partially observable Markov decision process models is difficult because we do not have access to domain-specific simplifications to help reduce the exponential complexity of uncertainty, state size, and planning horizons. Improvements to general POMDP planning would be broadly applicable to many robotics problem domains.

Learned similarity Monte Carlo planning (LSMCP) starts by gathering training data containing the ground-truth rollout values for many different action choices for the same states. LSMCP then learns a mapping from action-state tuples to a Euclidean space for easy distance calculations. Action-state tuples that are closer have higher similarity, and by combining many known values of action-state tuples together we can estimate the value of another unknown, but similar, action-state tuple. LSMCP uses these similarity estimates to inform the value of beliefs containing multiple possible states, leading to better value estimates for the same total number of Monte Carlo trials/iterations. So far, however, we have had only mixed results with LSMCP and further work is needed.

CHAPTER 2

Non-parametric Error Modeling for Ultra-wideband Localization Networks[1]

2.1 Introduction

In this chapter, we propose an ultra-wideband-based (UWB) localization system that achieves high accuracy through non-parametric estimation of measurement probability densities and explicit modeling of antenna delays. By using kernel density estimation and non-linear least-squares optimization, we take full advantage of the noisy non-Gaussian UWB measurements that might otherwise need to be identified and discarded to fit a parametric model. While we attempt to capture the physics of UWB measurements in an accurate mathematical model, we keep with our overall guiding theme and avoid any assumptions placed on the error model of these measurements. Differences in environment, configuration, and hardware can all impact the error landscape and we do not want to unnecessarily restrict the applicability of our system by placing parametric restrictions on the measurement error.

Wireless sensor networks show promise for rapid, accurate, and inexpensive deployment of localization in new environments, especially compared to sensors such as LIDAR or motion capture. We use ultra-wideband radios because of their low cost and potential for high accuracy measurements; specifically, we use the DecaWave DWM1000. There are several key problems to overcome for high localization accuracy. First, non-line-of-sight (NLOS) conditions result in high-variance non-Gaussian measurement error. Second, each radio has a unique and potentially unknown internal UWB parameter called *antenna delay*, which is the combined time it takes a signal to leave the generating hardware and be radiated by the antenna and the similar delays on the receiver. For the DWM1000, single-measurement error improves from roughly $30\,\mathrm{cm}$ to $4\,\mathrm{cm}$ after the process of manual antenna delay calibration [19]. Third, localization accuracy of a robot or other moving UWB node depends on knowing the positions of the fixed nodes, but manually measuring these for a large network can be time consuming.

[1] Adapted from Haggenmiller, Krogius, and Olson [12]

In order to achieve high accuracy and support rapid deployment, we consider the joint problem of solving for both antenna delays and node positions, using only the direct *time of propagation* measurements (sum of the respective antenna delays and time-of-flight) between nodes in the network.

This approach is difficult for several reasons. First, the addition of antenna delays and lack of a priori known *anchors* makes the problem less constrained. Second, in non-line-of-sight conditions the signal strength for the direct path is weakened and the timed signal will more likely correspond to a non-direct reflected route, or *multipath* route, that will overestimate the true distance. Third, to compensate for the weaker signal strength of the true path signal, the UWB radio can be made more sensitive, but this increases the likelihood of mistaking noise for the path signal and underestimating the true distance [20]. The combination of the previous effects leads to a measurement error model dependent on unknown environment conditions with an unknown number of modes and possibly non-Gaussian noise. Finally, the optimization problem is non-convex, has many local minima, and can suffer from geometric flip ambiguities [21].

To solve for unknown antenna delays, we explicitly model these delays and their prior probabilities. To account for unknown error conditions, we directly approximate the measurement probability density function (PDF) in a non-parametric fashion. As the problem is non-convex, we use gradientless non-linear optimization initialized with trilateration through robust quadrilaterals [22].

By choosing to use a non-parametric model for the measurement PDFs, we avoid approximating the PDF as some other model (e.g. Gaussian or Gaussian mixture) and we also avoid classifying and discarding measurements (perhaps incorrectly) as multipath outliers. In this way, out model attempts to get as much information out of each measurement sample as possible.

The main contributions of this chapter include:

1. Explicit coupled modeling of node antenna delays and locations in a maximum a posteriori formulation.

2. A framework for incorporating non-parametric error models, where the error models are built dynamically based on empirical data for each pairwise radio link.

3. Four open-access datasets which correspond to varying levels of LOS and NLOS links, for comparisons with our method.

2.2 Related Work

The use of sensor networks for localization is well studied in literature. A variety of sensor types may be used, including those based on WiFi access points [23], Bluetooth Low Energy (BLE)

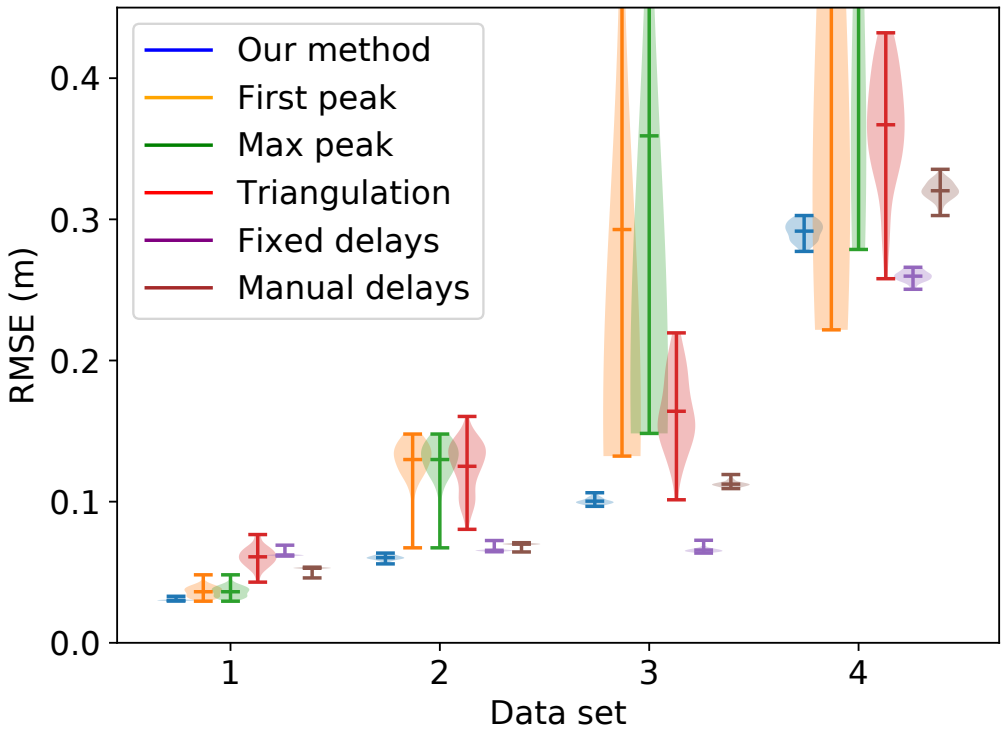

Figure 2.1: Robustness of our method to increasingly non-line-of-sight (NLOS) datasets. We show the distribution of RMSE of 7 UWB node locations evaluated on 50 bootstrap samples from each dataset. First peak and max peak methods, as predicted, are not robust to NLOS conditions (datasets 2, 3, 4). Triangulation, which does not perform non-linear optimization, has higher RMSE. Fixed nominal and manual delay methods cannot achieve as high accuracies on the low-NLOS datasets 1 and 2 as the full method, and the manual values might be over-fit to calibration conditions. High uncertainty in datasets 3 and 4 cause the full method to perform worse than the constant delay methods because the system is less constrained.

devices [24], chirp spread spectrum (CSS) radios [25], and ultra-wideband (UWB) radios [26]. While WiFi and BLE-based methods often work opportunistically with existing hardware, their dependence on received signal strength (RSS) and fingerprinting metrics leads to relatively coarse location estimates (on the order of meters). Furthermore, because these techniques do not directly use range estimates, they require a prior map of the localization environment.

When the node sensors are capable of taking range measurements, the main problem is *network localization*, or the localization of unknown node positions given the positions of a priori known anchors.

Three existing methods for network localization are: simulated annealing, semidefinite programming, and multidimensional scaling. When using simulated annealing localization [27], the node position estimates are randomly perturbed by progressively decreasing amounts which correspond to a temperature parameter. Changes that lower the error function are more likely to remain. At each temperature, all the nodes are perturbed sufficiently for the system to enter a thermal equilibrium before the temperature is lowered. When using semidefinite programming [28], a suboptimal relaxation of the weighted maximum likelihood estimate is used. This SDP formulation is convex and provides a good initial estimate for further refinement through a gradient search method. When using multidimensional scaling [29], the objective function is the least squares error between measured and expected ranges. A gradient estimate is found that is exact when the error is zero, which is then used with gradient descent. Unlike our method, each of these uses known anchors and was evaluated only in simulation, not taking into account hardware complexities like antenna delay.

Graph realization is a subproblem of network localization where all node positions are solved for with only range measurements. Known anchors are not used. This subproblem is less common in the literature. Di Franco et al. [30] propose an MDS-based method with Gaussian mixtures and expectation-maximization to solve the graph realization. They evaluate their method with the SurePoint [31] UWB hardware, which is also based on the DecaWave DW1000 UWB chip. They do not consider antenna delay characterization of their hardware.

Even with the use of UWB radios, there are multiple ways to estimate range between devices. In one case, by using time difference of arrival and wired synchronized clocks, a UWB system achieved sub-millimeter accuracy [26]; however, this need for wired time synchronization could make the system infeasible in certain scenarios. In a more typical example of synchronization-free time-of-flight (TOF) ranging, a UWB system also based on the DWM1000 module achieved single measurement error in the range of about $12\,\mathrm{cm}$ to $36\,\mathrm{cm}$ [32].

Once the graph realization is complete, the fixed UWB nodes can be used for real-time tracking of dynamic objects. Prior works in this area generally perform sensor fusion to mitigate multipath effects and achieve higher update rates. Hol et al. fuse inertial measurement unit (IMU) readings

with UWB time-of-arrival measurements from six synchronized nodes [33]. Zwirello et al. assume a pedestrian model and also fuse step length estimates in addition to IMU [34].

The area of graph realization for UWB networks has not yet been extensively explored. Our key differences compared to prior work include modeling of antenna delays and optimization on an empirical error model.

2.3 Approach

This chapter discusses use of UWB-based nodes using standard double-sided two-way ranging (DS-TWR) time-of-flight (TOF) measurements with the inexpensive DecaWave DWM1000 UWB module. We also constrain the problem to two dimensions because of a large dependence we found on measurement error due to the varying angle of incidence between antennas (Fig. 6 of [32]) and antenna radiation patterns (Fig. 7 of [35]). By restricting to two dimensions, we can orient the radios such that the antenna radiation patterns have equal energy over angle of incidence.

2.3.1 Problem statement

We start with a set of UWB nodes that are scattered through an environment. The size of this environment and what kinds of LOS or NLOS conditions to expect are unknown. We can attempt to take as many range measurements between any two nodes as we like, but depending on the distance between nodes and environmental conditions, it might not be possible to take measurements between every pair. Our goal is to find the coordinates and antenna delays of all the nodes given the set of measurements that we are able to make.

We pose this as a maximum a posteriori problem, expanding the prior on the parameters to include the probability of being able to measure a certain distance as well as that of encountering a certain antenna delay time:

$$\begin{aligned}
\theta_{\text{opt}} &= \arg\max_{\theta} p(\theta|M) && (2.1)\\
&= \arg\max_{\theta} p(M|\theta)p(\theta) && (2.2)\\
&= \arg\max_{\theta} \prod_{ij\in E} p(M_{ij}|\theta)\times \\
&\qquad \prod_{ij\in E} p(||X_i - X_j||) \times \prod_i p(\text{delay}_i) && (2.3)
\end{aligned}$$

Optimization parameter θ includes both coordinates X and antenna delays; M includes all

Table 2.1: Symbols and definitions used in this chapter

c	speed of light, in $\mathrm{m}/\mu\mathrm{s}$
M	set of all measurements t for all edges ij
E	set of all measured edges ij in the network
X	set of all estimated coordinates (2 or 3-D) X_i
f	Gaussian probability density function
delay	set of all antenna delays $delay_i$
μ_{delay}	nominal antenna delay value
σ_{delay}	nominal antenna delay standard deviation
θ	optimization parameters $\{X, \text{delay}\}$
tof_{ij}	time of flight for edge ij
prop_{ij}	propagation time estimate for edge ij
d_{ij}	distance estimate for edge ij
$G(M_{ij})$	convolution-estimated PDF for edge ij
pdf_{ij}	table of estimated PDF for edge ij

measurements taken on each edge in E (see symbol key in Table 2.1).

We face a variety of problems making this formulation tractable. First, the individual measurements of any edge are not independent because they are conditioned on an unknown environment. For example, while LOS conditions generally result in a unimodal Gaussian distribution, NLOS conditions may lead to an arbitrary number of arbitrarily spaced Gaussians that may not even include one centered on the true distance, depending on the environment.

Second, the need to perform computations on individual measurements may pose computational problems when there are many measurements. Third, finding the correct prior probabilities for distances that can be measured (since the hardware devices have limits to their range) and for the antenna delays also increases the complexity of the problem.

Without further constraints on the coordinates and network, the maximal solution will not be unique. At the very least, the coordinate system requires a fixed origin and coordinate axes. This leaves a solution that should differ from the ground truth by at most one translation, rotation, and mirroring operation. However, especially in NLOS conditions, node connectivity might be low enough and uncertainty high enough that it is possible for a network topology to emerge with flip ambiguities that result in multiple optimal solutions [22] [21]. A *flip ambiguity* is when a network has multiple unique exact solutions. The simplest example occurs when a node has connectivity of only two and its position could be on either side of the two connected nodes. Even if the solution is unique, conditions close to a flip ambiguity may contribute to various local minima, so a good initial estimate will be necessary to find the best solution.

One of the key ideas in this chapter is to build a probabilistic model of range measurements in terms of a density estimation problem, viewing each of these individual measurements as data that

informs the model. Our approach is to essentially convolve each measurement with a Gaussian corresponding to the intra-modal error (which we characterize under controlled testing conditions before the fact) and then compute the sum over the Gaussians. For simplicity, we also disregard the prior probability for distances on measured edges, $p(||X_i - X_j||)$, as it is likely uniform in the basin of convergence of the optimal solution.

We take $p(M_{ij}|\theta)$ to be the sum of the measurement probabilities, with f the Gaussian PDF, M_{ij}^t the measurement t of edge ij, $\text{prop}_{ij}(\theta)$ the propagation time estimated from parameters θ including coordinates X and antenna delays, and σ^2 the nominal intra-modal measurement variance:

$$\begin{aligned} p(M_{ij}|\theta) &\approx \sum_t f(M_{ij}^t; \text{prop}_{ij}(\theta), \sigma^2) \\ &:= G(M_{ij}, \text{prop}_{ij}(\theta)) \end{aligned} \tag{2.4}$$

We recognize that this is essentially a convolution between M_{ij} and a Gaussian with $prop_{ij}(\theta)$ specifying the location of the result in that convolution. This means we only need to calculate that convolution once in the method's initialization. We treat G as an estimate of the probability density function (PDF) of M_{ij}. This operation is effectively the same as calculating the kernel density estimate with a Gaussian kernel.

This leaves us with our final form of the optimization problem:

$$\theta_{\text{opt}} = \arg\max_\theta F(\theta) \tag{2.5}$$

$$F(\theta) = \prod_{ij \in E} G(M_{ij}, \text{prop}_{ij}(\theta)) \prod_i p(\text{delay}_i) \tag{2.6}$$

To solve the problem of the arbitrary origin, rotation, and mirroring, we make use of the fact that our intended application is robot localization. Our algorithm can first assign these arbitrarily during triangulation. Later, we can have the robot drive in a small loop, redefining the origin as the robot's initial location and the x-axis as the direction of its first motion. We can then resolve the mirroring ambiguity by coercing the winding order of the robot's estimated trajectory to match the known winding order of the path.

To solve both the problem of flip ambiguities and of a good initial estimate, we make use of trilateration with robust triangles [22]. This method will also identify whether the network is such that flip ambiguities cannot be distinguished. We run the robust trilateration with a series of different initial measurement values drawn from the distribution G of each edge. This gives us a variety of initial conditions across different possible modes of the data. We choose the trilateration with the lowest residual to further optimize.

We also repeat the entire process of choosing an initial estimate and further optimizing it several

times to choose the best solution (see Algorithm 1).

Algorithm 1: Our non-parametric graph realization method

Input: number of total attempts A
number of random initializations L
measurements M
edges E
Output: best solution θ_{opt}
for $ij \in E$ **do**
  $\text{pdf}_{ij} \leftarrow \text{EstimatePDF}(M_{ij})$
end
$\text{delay}^0 \leftarrow \{\mu_{\text{delay}}, \mu_{\text{delay}}, \ldots\}$
for $a \leftarrow 1$ **to** A **do**
  for $l \leftarrow 1$ **to** L **do**
    $\text{prop} \leftarrow \text{DrawFromPDFs}(\text{pdf})$
    $d \leftarrow \text{EstimateDistances}(\text{prop}, \text{delay}^0)$
    $X^{0l} \leftarrow \text{RobustTrilateration}(d)$
  end
  $X^0 \leftarrow \arg\min_l \text{Residual}(X^{0l}, \text{delay}^0)$
  $\theta_a \leftarrow \arg\min_\theta F(\theta)$
    $\theta^0 = \{X^0, \text{delay}^0\}$
end
$\theta_{\text{opt}} = \arg\min_a \text{Residual}(\theta_a)$

2.3.2 Estimating probability density functions

We approximate the PDF of each link in the network by convolving the measurements with a Gaussian of nominal standard deviation $\sigma = 1.3 \times 10^{-4}\,\mu\text{s}$ (see Figure 2.2). The convolution is stored in a table over a range from $\min_j p_j - 6\sigma$ to $\max_j p_j + 6\sigma$ with a spacing of $2.5 \times 10^{-6}\,\mu\text{s}$, or approximately $0.075\,\text{cm}$.

We estimate this nominal standard deviation empirically as the standard deviation of measurements between a pair of our nodes under ideal unimodal LOS conditions.

We use quadratic interpolation to read out probabilities from each table. For queries beyond the 6σ range, we approximate the result with a Gaussian centered at the mean of the distribution with a standard deviation of $\sigma = 1.3 \times 10^{-4}\,\mu\text{s}$ again.

2.3.3 Robust quadrilateral trilateration

We find our initial localization estimate by assuming a fixed antenna delay value (μ_{delay}) for every node and fixed propagation times drawn from the measurement PDFs.

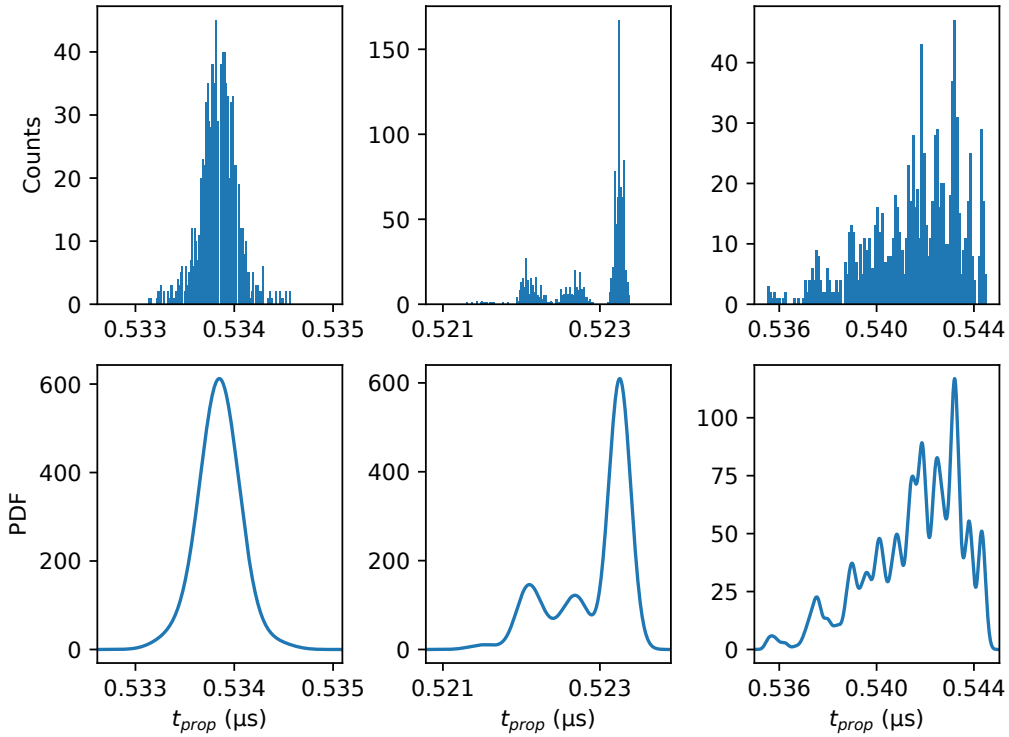

Figure 2.2: Estimated probability distributions (*bottom*) from 1024 UWB node-to-node measurements (*top*) with varying line-of-sight conditions. A unimodal line-of-sight link (*left*) could be easily approximated by a single Gaussian and the multimodal measurements (*center*) could be fairly well represented by 3 or 4 Gaussians, but the heavily non-line-of-sight measurements (*right*) are not Gaussian. Our non-parametric error modal preserves the distribution in all three cases.

With these assumptions, we have a single distance measurement for each link to use in the trilateration:

$$\text{tof}_{ij} = \text{prop}_{ij} - 0.5(\text{delay}_i + \text{delay}_j) \tag{2.7}$$

$$= \text{prop}_{ij} - \mu_{\text{delay}} \tag{2.8}$$

$$d_{ij} = c \times \text{tof}_{ij} - \text{bias}(c \times \text{tof}_{ij}) \tag{2.9}$$

The bias function is a range bias characterized by the manufacturer that we have smoothed with quadratic interpolation [36]. From this point, the method is performed as described by Moore et. al. [22].

2.3.4 Minimization of the non-parametric objective function

In order to capture the full complexity of our measurements, we are building non-parametric error models of each link at run time. As this precludes use of analytic gradients, we optimize $\log F(\theta)$ with the generalized Gauss-Newton method and calculate both the gradient and Hessian numerically.

We use optimization parameter $\theta = \{X, \text{delay}\}$, where X includes both x and y coordinates. Extension to higher dimensions is trivial at this point.

$$\log F(\theta) = \sum_{ij \in E} -\log G(M_{ij}, \text{prop}_{ij}(\theta)) + \sum_i -\log p(\text{delay}_i) \tag{2.10}$$

$$\text{prop}_{ij}(\theta) = \text{tof}_{ij}(\theta) + 0.5(\text{delay}_i + \text{delay}_j) \tag{2.11}$$

$$\text{tof}_{ij}(\theta) = \big(d_{ij} + \text{bias}(d_{ij})\big)/c \tag{2.12}$$

$$d_{ij} = ||X_i - X_j|| \tag{2.13}$$

$$p(\text{delay}_i) = f(\text{delay}_i; \mu_{\text{delay}}, \sigma^2_{\text{delay}}) \tag{2.14}$$

We use $\sigma_{\text{delay}} = 3.3 \times 10^{-4}\,\mu\text{s}$, which is derived from the manufacturer's antenna delay calibration note citing a 3-sigma variation of $30\,\text{cm}$[19]. Unfortunately, the manufacturer did not include a mean antenna delay, so we performed the manual calibration described in that document for 8 of our UWB nodes, and took $\mu_{\text{delay}} = 0.516\,\mu\text{s}$ as the mean.

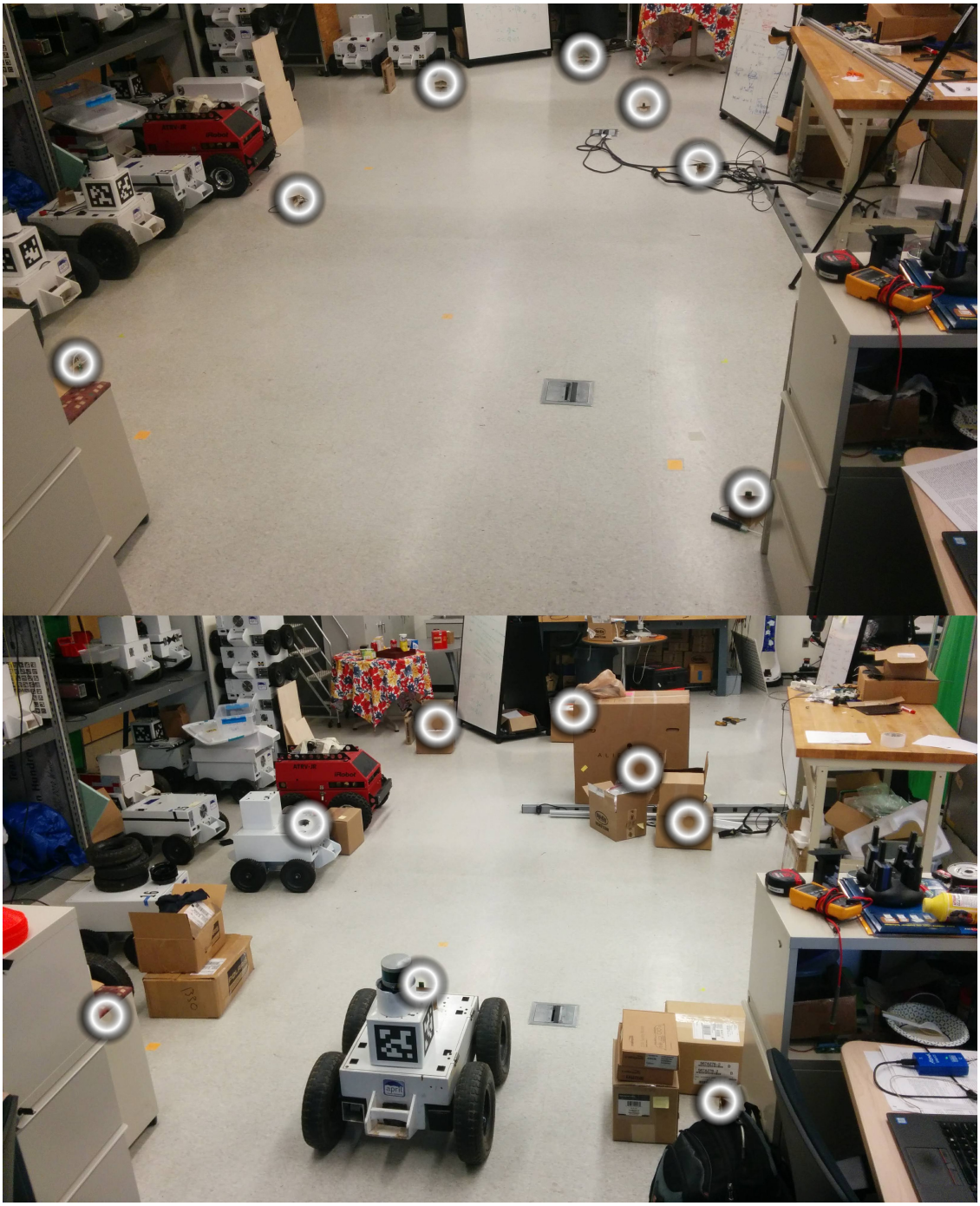

Figure 2.3: Layout of our 7 fixed-location UWB radios, in line-of-sight conditions dataset 1 (*top*), and non-line-of-sight dataset 3 (*bottom*). Nodes are circled, but are only approximate in the bottom image. We made our datasets progressively non-line-of-sight by surrounding the nodes with boxes and other clutter.

2.4 Evaluation

In all of our experiments, we configured our DWM1000 module to transmit on channel 1 at 850kbps with a 64MHz pulse repetition frequency (PRF), 512 symbol preamble, the DecaWave-optimized start-of-frame delimiter (SFD), and with smart transmit power control, according to the DW1000 user manual [37]. We used NLOS-optimized threshold settings, LDE_CFG1: 0x67, LDE_CFG2: 0x0004 [20].

For each dataset, we placed eight UWB nodes in our lab space as in Figure 2.3, with the antennas oriented vertically. As we intended to later localize the robot by use of the seven fixed nodes, we did not collect a ground truth location for the robot. However, measurements between the robot and the other seven nodes were still included in the graph realization. Dataset 1 was taken as shown in Figure 2.3 (*Top*), so that LOS would exist between all nodes. Dataset 3 was taken as shown in 2.3 (*Bottom*). Datasets 2 and 4 had fewer and greater obstructions than dataset 3, respectively.

While we perform these experiments with known Z coordinates (mostly zero), the method should in principle extend to higher dimensions. However, a full 3D solution is inherently more sensitive to the geometry of the network and the number of measurements.

To take data for evaluation of our method, we attempted to collect 1024 measurements for each of the 28 possible links. As some NLOS links might be completely blocked, we would give up if we could not collect that many measurements in under 10 seconds. Links with LOS conditions typically collected 1024 measurements in about 7 seconds. For all of our evaluations, we first process the data slightly to remove any measurements of less than $0.5\,\mu$s as obviously representing invalid negative ranges, and any measurements more than 4 standard deviations from the mean. This prevents the PDF tables from becoming needlessly large in an attempt to include outlier noise. In most of our evaluations, we do not use the full number of measurements taken, and so we take a random bootstrap sample of the desired number of measurements. We evaluate each condition with the same 50 distinct random seeds, so that each configuration is compared against the same 50 random subsets of measurements. We report the root mean square error between solved positions of the 7 fixed non-robot nodes and ground truth after finding the translation, rotation, and mirroring that minimizes this error. Ground truth was determined by placing nodes only at the corners of 1ft-by-1ft floor tiles.

Unless otherwise specified, we performed all of our evaluations with total method attempts $A = 20$, random trilateration initializations $L = 250$, $\mu_{\text{delay}} = 0.516\,\mu$s, $\sigma_{\text{delay}} = 3.3 \times 10^{-4}\,\mu$s, and a maximum of 256 measurements per link.

We did not further investigate A or L because we found those values to be relatively insensitive, and performance did not seem to increase for larger values. We performed parameter sweeps over

the variables describing the prior on antenna delay due to their significance to the method, and over the number of measurements because taking measurements is the dominant factor in the total time taken for graph realization. The results are in Figures 2.4, 2.6, and 2.7.

Finally, we evaluate the performance of our method by ablation analysis. In the *first peak* version of the algorithm, we construct the PDF with a single Gaussian of mean μ centered on the lowest peak (local maxima) of the full PDF G, with standard deviation fit to minimize the least-squares error. This method follows the intuition that because incorrect multipath measurements are too long by definition, the first peak most likely represents the direct path. The *max peak* version is similar, but uses the highest amplitude peak of the full PDF, following the intuition that multipath or noise-based peaks will be lower in strength. The rest of the algorithm in both methods is unchanged. In the *triangulation* version, the algorithm skips the non-linear minimization of F (antenna delays remain at nominal values). In the *fixed delays* version, the non-linear minimization is only over the coordinates, leaving the antenna delay times constant at the prior mean value. The *manual delays* version is similar, but we manually calibrate the values in LOS conditions with LOS optimized thresholds [20]. We place the nodes three at a time in an equilateral triangle with 10-foot long sides and take 1000 measurements between each pair. We use the mean values and compute the least squares solution for the antenna delays [19]. The results of the total ablation analysis are in Figure 2.1.

2.5 Discussion

We use the ablation analysis in Figure 2.1 to examine the performance of our method under the increasing NLOS conditions from datasets 1 through 4. We find that under the most LOS conditions, the more naive first/max peaks methods have equivalent performance, but the disparity grows rapidly as the conditions become increasingly NLOS. We also find that the benefit of modeling antenna delays seems to fall off under NLOS conditions. We recognize that with only 8 nodes, our complete problem is only barely fully constrained, having $8 \times 7/2 = 28$ equations and $8 \times 3 = 24$ unknowns. As the uncertainty in the NLOS links increases, the overall system would need to be more over-determined to accurately solve for all the unknowns. As a result, in high NLOS cases we cannot solve for the antenna delays to a higher accuracy than the $0.516\,\mu\mathrm{s}$ prior. However, on the most well constrained LOS case, modeling of the antenna delays cuts RMSE from $6.2\,\mathrm{cm}$ to $3.0\,\mathrm{cm}$.

Examining the sensitivity of our method to the prior on the antenna delays, we find from Figure 2.4 that the prior mean μ_{delay} is significant to the performance of our method, but looking at Figure 2.5 we see that our method is able to return antenna delay values that are many standard deviations more accurate than the values provided by the incorrect priors. Looking at the prior on standard

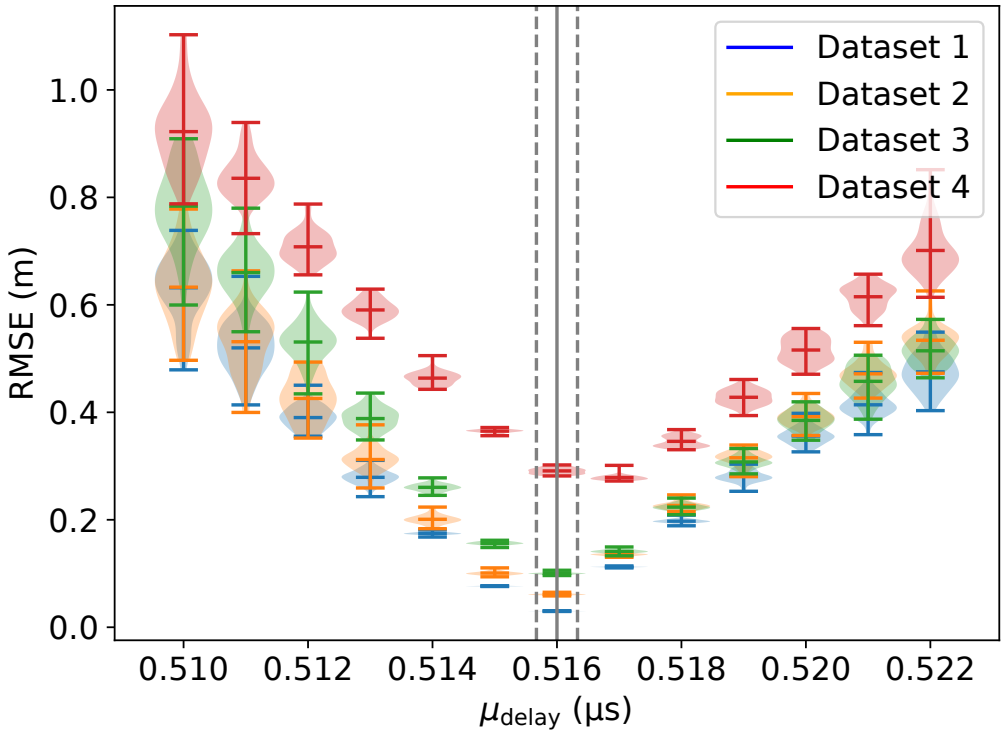

Figure 2.4: Graceful degradation of our method's accuracy when supplied with an incorrect antenna delay prior mean μ_{delay}. Each additional nanosecond difference from the empirical mean of $0.516\,\mu\text{s}$ (*vertical line*) is a $30\,\text{cm}$ biased error on each measurement. The DWM1000 manufacturer specifies an antenna delay standard deviation of $3.3 \times 10^{-4}\,\mu\text{s}$ (*dashed lines*) [19].

deviation σ_{delay}, we find the method to be far less sensitive. In the most NLOS case where the antenna delays are least constrained, we find that the too-large standard deviations are particularly harmful. Values in the range of about $1 \times 10^{-4}\,\mu\text{s}$ to $5 \times 10^{-4}\,\mu\text{s}$ seem to work best. Smaller values effectively prevent the antenna delays from moving away from the mean, reducing the algorithm to the fixed delay simplification, while larger values leave the choice of antenna delay too unconstrained.

The total time necessary for the graph realization and calibration in a new environment is dominated by the time needed to take measurements across all possible edges. With our evaluation setup, taking 128 measurements per link would take a maximum of 35 seconds for the 8 nodes. Execution of the complete algorithm took half a second on the author's laptop in the worst case, leading to less than one minute for the complete localization process.

2.6 Conclusion

In this chapter, we proposed a non-parametric error model that makes UWB network localization robust to NLOS conditions. We found that our modeling and simultaneous calibration of antenna

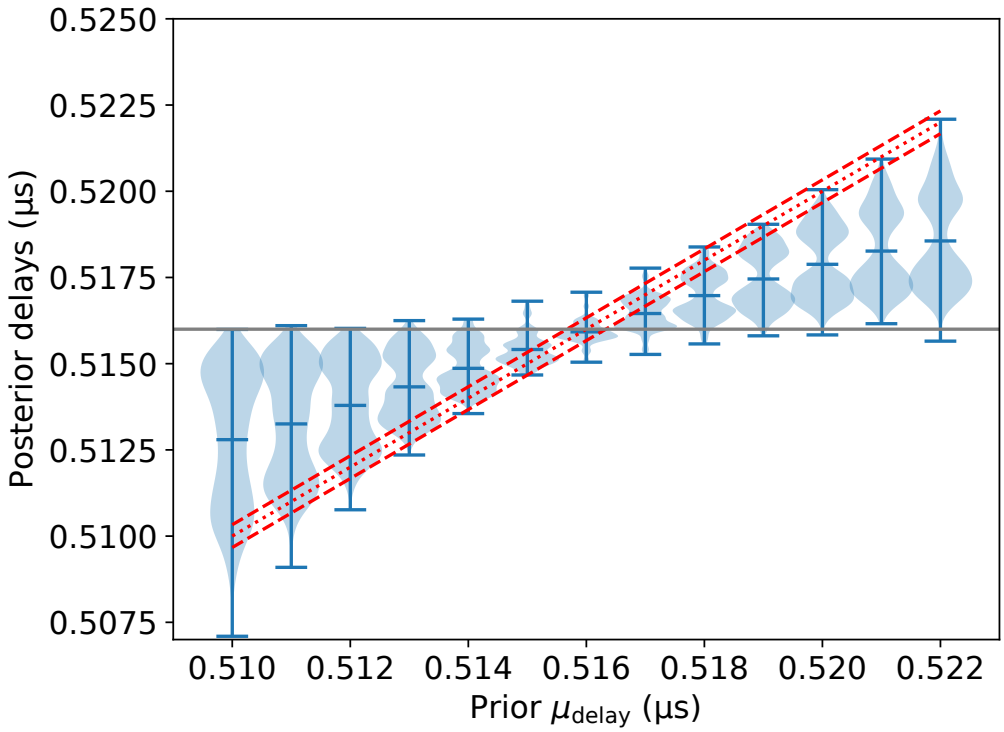

Figure 2.5: Robustness of our method's ability to solve for antenna delays with an incorrect prior mean (aggregated over all UWB nodes and datasets). If our method did nothing to improve on the given prior, each violin plot would have its posterior mean centered on the given prior (*dotted red line*). If our method were perfect, each plot would have the posterior centered on the known true mean (*solid horizontal line*). Because the prior standard deviation is small (*dashed red lines*), the optimization faces a high cost to move the posteriors away from the given prior. We see that under all conditions, our method produces posterior antenna delays significantly better than the supplied prior.

delays lead to very high accuracy is LOS conditions. We found that only 16 to 128 measurements were necessary per link and that a network of 8 UWB nodes required less than a minute for both the collection of measurements and execution of the algorithm. On this network we achieved accuracies of $3\,\text{cm}$ RMSE in the LOS dataset 1, and $30\,\text{cm}$ in the NLOS dataset 4. Our datasets and source code files can be found at `https://osf.io/pkbq4/`.

In the next chapter, we propose a non-parametric planning framework for domains such as autonomous driving, where rollouts are expensive and the costs/rewards for each action are high-variance, similar to how UWB measurements are high-variance under NLOS conditions.

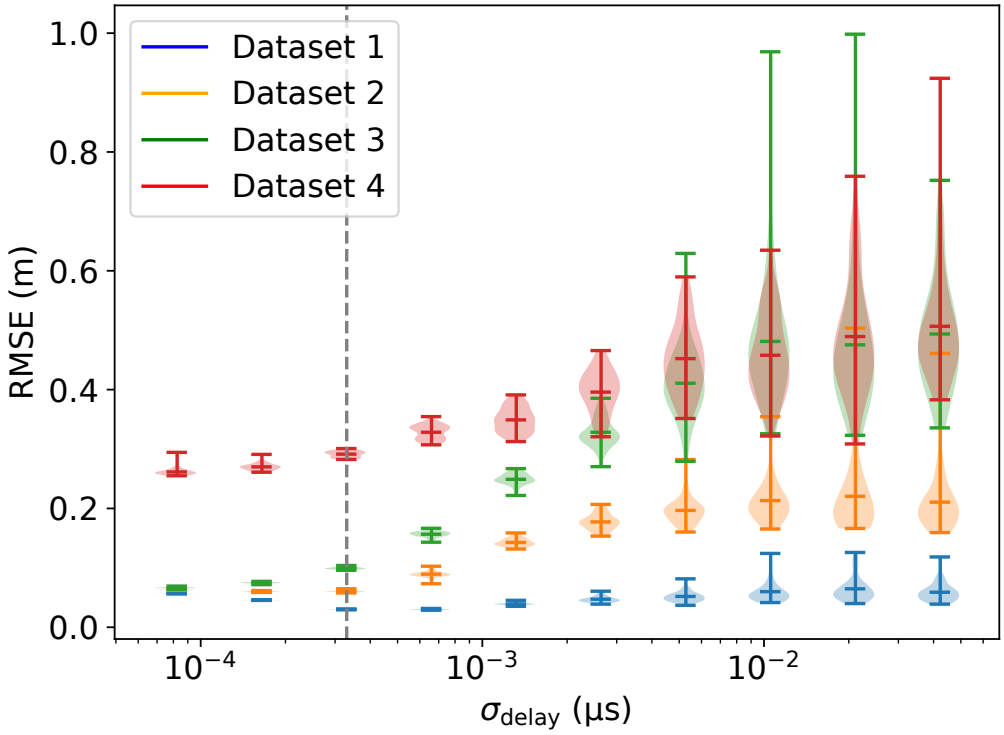

Figure 2.6: Sensitivity analysis of our method to the antenna delay standard deviation prior σ_{delay}. Dataset 1 has low uncertainty and so is well constrained, making it insensitive to choice of σ_{delay}. Datasets 2, 3, and 4 have increasing uncertainty from non-line-of-sight conditions, making them more sensitive. Values below the nominal of 3.3×10^{-4} µs (*dashed vertical line*) essentially make the method perform as if the antenna delays are fixed. Datasets 3 and 4 are not quite constrained enough to solve for antenna delays well, so their RMSE grows the most for larger σ_{delay} (compare the full method with the fixed antenna delay version in Figure 2.1).

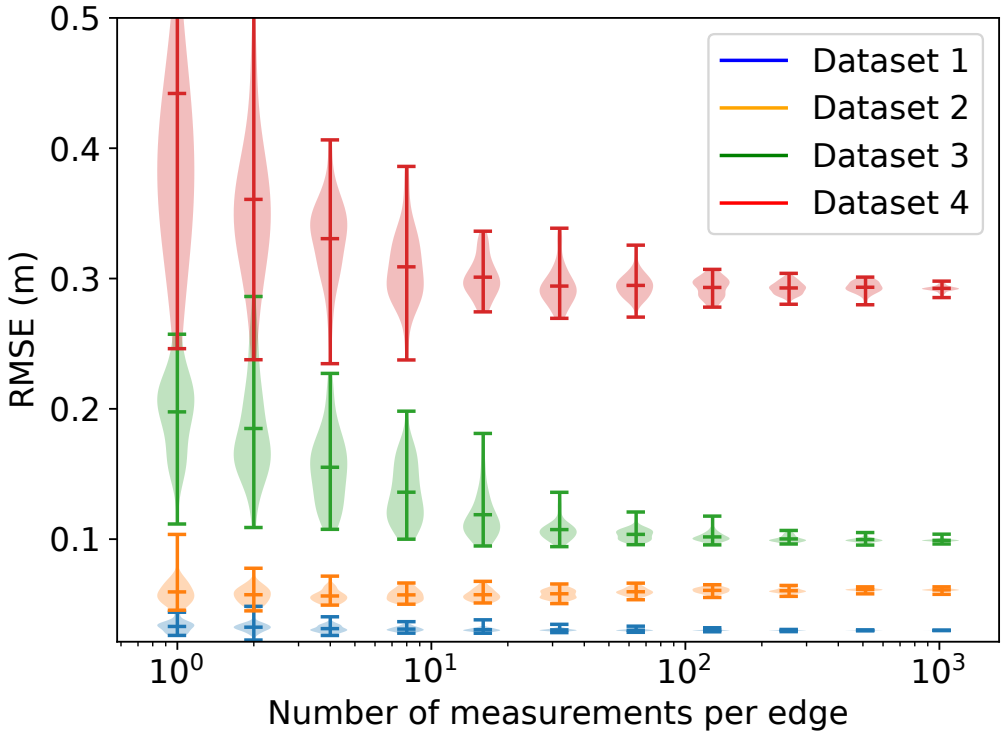

Figure 2.7: Convergence rate of our method on varying line-of-sight conditions. The fewer the measurements needed, the less time required to localize and characterize the network. In datasets 3 and 4, accuracy continues to improve until about 128 measurements per edge. In datasets 1 and 2, however, low measurement uncertainty allows even just one measurement per link to suffice.

CHAPTER 3

The Masked Mapper: Masked Metric Mapping[1]

3.1 Introduction

In this chapter, we present a flexible system for simultaneous localization and mapping (SLAM) that combines elements from traditional dense metric SLAM and from topological SLAM, using a *masking function* (*mask*) to focus attention by controlling which robot poses are available for loop closures. Under our guiding principal of avoiding explicit parametric models where possible, we take advantage of key ideas from topological mapping without imposing the full constraint of accurate topological labelling. Noticing that certain topological classes are useful for effective loop closing, we can use approximate/simplified detectors for these classes to aid in loop closing, but without needing to worry about false positives or other accuracy concerns in the detectors. In this way, we get advantages without the constraints of a strict model.

Typical robot navigation systems use dense metric map representations that are based around information-dense lidar maps for localization and route planning. In pose SLAM, a factor graph is constructed with nodes representing the pose of the robot at different points in time and edges representing odometry measurements and lidar-scan loop closures. There are efficient techniques to optimize these factor graphs as non-linear least squares problems [38]. By continually integrating odometry and lidar scans, the precise location of the robot and a detailed map of its environment can be maintained at all times. As the size of the map increases it becomes more and more difficult to maintain global consistency, as odometry drift will continue to accumulate until the robot returns to an area it has seen before. This accumulated error can be eliminated by matching the robot's current location to places it has seen before to form loop closures, but this process is computationally expensive and has the risk of incorrectly matching similar-looking regions (perceptual aliasing). The continuous loop-closing work of dense metric SLAM produces highly precise maps at a high computational cost and potential risk of false-positive loop closures.

[1] Adapted from Haggenmiller, Kabacinski, Krogius, and Olson [13]

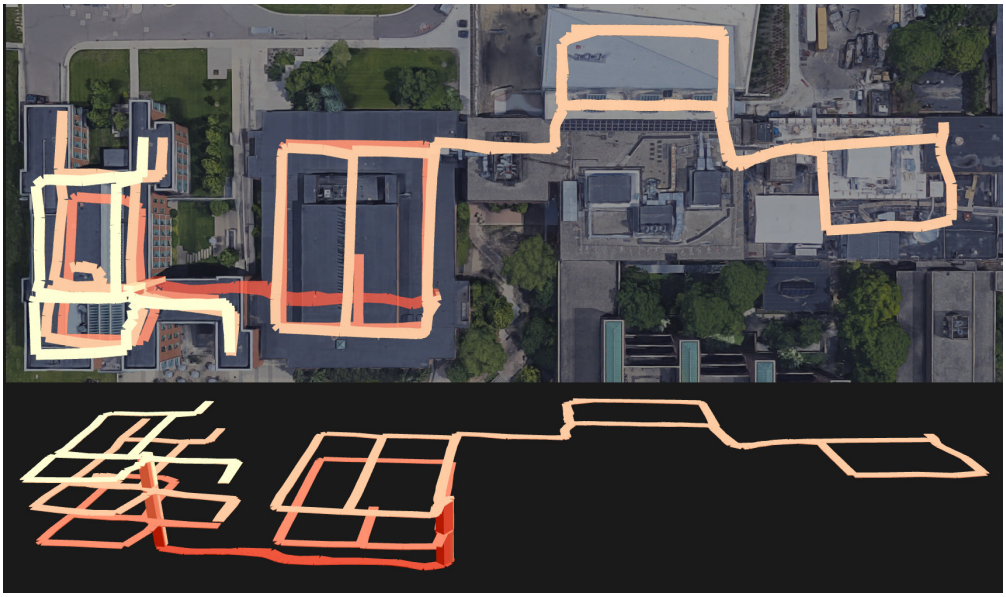

Figure 3.1: Top-down (top) and perspective (bottom) views of a masked metric map, spanning several buildings on the University of Michigan North Campus. The floors are color-coded, using a lighter color for each floor from the first to the fourth. The robot drove $2.5\,\mathrm{km}$ and the total optimized path length over all floors is $1.9\,\mathrm{km}$. Imagery from Google and Maxar Technologies.

Topological mapping methods address these problems by focusing on interpreting the semantics of the environment and using these to build models that more closely resemble the human cognitive map [39] with intersections, paths, places, regions, and so forth. These representations can be very simple and the robot will only need to close loops when it makes sense based on the topology and semantic labels. As a result, topological maps can generally define an environment with fewer nodes and loop closures, although the process of determining the correct semantic labels for the robot's location and of segmenting the environment can be complex.

The key advantage in topological mapping is that high-precision localization in hallways and similar environments is often unnecessary. A simple control law such as go-straight or wall-follow will reliably navigate these scenarios without centimeter-level localization. In addition, these hallways are the places where scan matching systems are most likely to fail from perceptual aliasing.

Our system takes advantage of the fact that some places need precise localization more than others by abstracting this distinction into a configurable masking function. A well-designed masking system can achieve both lower probability of failure and lower computational complexity, sacrificing precision in hallways and similar locations that is often unneeded.

The main contributions of this paper include:

1. A flexible approach to masking a high performance SLAM system to lower computational cost and minimize the chance of errors due to perceptual aliasing.

2. Two effective masking functions based on detecting openings and intersections that improve

map quality while performing five times fewer scan matches than an "always true" mask function.

3. Demonstration of our system on a $2.5\,\mathrm{km}$ dataset.

3.2 Related Work

In building a consistent map, mapping methods must choose when and how to represent locations, how to detect loop closures and validate them, and how to optimize the resulting map/graph. We generally refer to any specifically remembered location as a *node*.

In metric pose SLAM, nodes are generally placed at a regular spacing, with each node maintaining its measurements and object frame. Matches between nodes are used to close loops, and in an early paper, matches are searched for between all pairs of nodes with a matching score threshold for acceptance [40]. Thrun et. al. filter the node pairs to check with a series of geometric constraints before matching them, and repeat this in an iterative approach [41]. Google's Cartographer builds locally consistent submaps, and then later connects them. Whenever the system gets a new lidar scan, it attempts to match it with nearby completed submaps. Loop closures that have a high enough match score are added to the optimization with a Hilbert loss for robustness against false positives [42]. Our method follows from dense pose SLAM, focusing on how sparse node placement may make the matching and loop closure problems easier.

Just as our method is based around a selective masking function, topological methods often focus on node selection in their maps. The nodes can represent entire partitioned areas, as with how Thrun partitions an occupancy-grid map by finding the critical points and lines of the Voronoi skeleton where clearance is locally minimized [43]. The nodes can also be points in the robot's trajectory, with the trajectory segmented into straight lines and nodes placed at the "corners", fitting the segments to the original trajectory [44]. Although simple, this representation can be used for absolute self-localization and building consistent maps [45].

The Spatial Semantic Hierarchy (SSH) topological mapping framework also starts with abstract definitions and lends itself to extension. Nodes are abstractly defined as the end-state of a hill-climbing control law that should be unique for localization and consistent mapping, while the edges between nodes are also abstractly specified by a transition control law and a termination criterion [39]. The nodes are often defined to be decision points at gateways or openings in the map. They can be found in ways of varying complexity: by computing the medial axis of the free space in a local occupancy grid constructed around each node [46]; by computing the pruned Voronoi skeleton of an occupancy grid and selecting the portions of the skeleton that leave the "core" [47]; and by calculating the Voronoi skeleton, finding the isovist at each point, computing

the isovist eccentricity, and finally selecting the locations of openings with a learned classifier [48]. These methods generally take advantage of semantic place-type labels and gateway locations to filter loop closure candidates and get initial guesses for the match transformations. Another hybrid approach grounds the nodes as areas of high sensory overlap and then optimizes the robot's path, the local metric maps, and the transformations between areas as a unified Bayesian problem [49].

Feature-based SLAM methods demonstrate more ways to close loops. An expectation-maximization approach based on Markov localization simultaneously optimizes both robot poses and the map but does not achieve real-time performance [50]. A covariance-entropy ratio test can be used to both place nodes ("keyframes") and to validate loop closures, using the entropy of a reference loop closure to set dynamic thresholds [51]. Loop closure hypotheses can be made with a RANSAC-based algorithm using geometric and locality constraints and verified with a joint compatibility branch-and-bound algorithm [52]. To make this loop closure validation more robust, it can be postponed as long as there is considerable overlap between different local maps [53]. Alternatively, a learned classifier can be used to generate candidate loop closures with validation provided by RANSAC and geometric constraints [54].

3.3 Approach

3.3.1 Problem statement

We consider the process of observing odometry and lidar scans in order to map an environment for localization.

As a pose SLAM problem seeking to build a consistent map, we have to choose which robot positions to represent in our map (the nodes), how to detect loop closures and validate them, and how to optimize the resulting map/graph.

We consider the use of an arbitrary masking function to determine when a robot position should be explicitly represented in the map as a node with its corresponding lidar scan. To maintain global consistency, the system will also need to detect and validate loop closures in a way that does not depend on the choice of mask. In order to find all valid loop closures, the mask must consistently fire at least once in the area around each ground-truth junction in the map.

We define the masking function as a function that maps to a Boolean from all of the robot's observations Z until the current time T:

$$MaskF : \{Z_t\}_{t=0}^{T} \rightarrow \{0, 1\} \tag{3.1}$$

Any arbitrary masking function can be used with our system, and like any binary function, masks

can be combined together with Boolean operators. For long-term applications, it may be helpful to design a masking function to select for static environmental features. In this chapter, however, we aim to principally show the advantage of choosing a mask that selects for distinctiveness.

3.3.2 Method overview

The first step in creating the metric map is deciding when to add nodes. We defer this choice to the specified masking function with a few exceptions: we choose a minimum edge length ($1.5\,\mathrm{m}$) to prevent duplicate scan-matching nodes and a maximum edge length ($9\,\mathrm{m}$) after which we add odometry-only nodes. We also add one of these odometry-only nodes when a floor change is detected. These odometry-only nodes are necessary because a rigid-body transformation can only approximate the actual motion of the robot on an edge, and does not fully couple the rotational and translational components. Limiting the maximum edge length gives the map optimizer a more accurate representation to work with.

After adding a scan-matching node to the graph, we determine if it might close any loops in the map. We start by computing the *Dijkstra Projection* [55] from this new node to all other nodes in the trajectory graph. This provides us with the expected rigid-body transformation and associated covariance of the minimum error path from the new node to each existing node. We then compute the Mahalanobis distance of the displacement between each of those pairs. If this falls below some threshold, then the two nodes have a high likelihood of being in the same area.

Depending on the masking function, there may be either a high or low number of potential nodes to check, so we sort them by the Mahalanobis distance and only proceed with a limited number (MaxMatches; 5 in our evaluation) of the most likely nodes for further testing. We use the method described in Many-to-Many Multi-Resolution Scan Matching [56], and only keep the matches that pass a minimum scan-match similarity score.

We then use our loop validator to confirm these potential loop closures before adding them to the map for optimization. Whenever we validate a loop closure, we optimize our map factor graph by performing a batch update of the nonlinear least-squares problem.

We describe these steps more precisely in Algorithm 2.

3.3.3 Dijkstra projection

The Dijkstra Projection [55] predicts the rigid transformation and covariance of the minimum error path between a source node and all other nodes in the graph, using Dijkstra's algorithm. We could

use any scalar uncertainty metric over covariance, but choose the trace for simplicity:

$$\text{path}_{ij\text{opt}} = \underset{\text{path}ij}{\arg\min}\, \text{tr}(\Sigma_{\text{path}ij}) \tag{3.2}$$

As the shortest path is computed, we keep track of the cumulative predicted transformation and covariance, composing each successive transformation z_b onto the current total path transformation z_a.

Rigid transformations z_a and z_b can be composed like so:

$$\begin{aligned} z_{ab} = & \quad z_a \circ z_b \\ \equiv & \begin{bmatrix} z_{a_x} + \cos(z_{a_\theta}) z_{b_x} - \sin(z_{a_\theta}) z_{b_y} \\ z_{a_y} + \sin(z_{a_\theta}) z_{b_x} + \cos(z_{a_\theta}) z_{b_y} \\ z_{a_\theta} + z_{b_\theta} \end{bmatrix} \end{aligned} \tag{3.3}$$

And their combined covariance can be estimated with the Jacobians J_1 and J_2 of Eq. 3.3 (see Appendix B of Bosse et. al. for details [55]):

$$\begin{aligned} \Sigma_{z_{ab}} = & J_1(z_a, z_b) \Sigma_a J_1(z_a, z_b)^T + \\ & J_2(z_a, z_b) \Sigma_b J_2(z_a, z_b)^T \end{aligned} \tag{3.4}$$

We compute the Mahalanobis distance of a translation with an additional parameter R ($2\,\text{m}$) of translation that we want to allow between x_i and x_j. Although the Dijkstra projection also provides a transformation, we use the transformation from our current optimized map, $z_{ij} = x_i^{-1} \circ x_j$:

$$\text{Maha}_{ij} = \sqrt{z_{ij} (\Sigma_{ij\text{opt}} + \text{diag}(R^2, R^2))^{-1} z_{ij}} \tag{3.5}$$

We allow this extra translation R because two nodes do not need to coincide exactly to form a good closure, they only need to be close enough for their scan match transformation to complete the loop. The R we use then represents the distance apart at which we can reliably match two lidar scans.

Algorithm 2: Masked Mapper SLAM

$N \leftarrow \{\}$, $E \leftarrow \{\}$
$N_0 \leftarrow$ CreateNode(Pose(), LidarScan())
$t_{\text{last full}} \leftarrow 0$, $t \leftarrow 1$
Loop
  repeat
    $d \leftarrow ||x_{t-1}^{-1} \circ \text{Pose}()||$
    if $d >$ MaxEdgeLen $\vee$ FloorChanged() **then**
      $n \leftarrow$ CreateNode(Pose(), $\emptyset$)
      $e \leftarrow$ CreateEdge(N_{t-1}, n, $x_{t-1}^{-1} \circ$ Pose())
      $N_t \leftarrow n$, $E \leftarrow E \cup e$, $t \leftarrow t + 1$
    end
    $d_{\text{full}} \leftarrow ||x_{t_{\text{last full}}}^{-1} \circ \text{Pose}()||$
  until $d_{\text{full}} >$ MinEdgeLen $\wedge$ MaskF(Observations);
  $n \leftarrow$ CreateNode(Pose(), LidarScan())
  $e \leftarrow$ CreateEdge(N_{t-1}, n, $x_{t-1}^{-1} \circ$ Pose())
  $N_t \leftarrow n$, $E \leftarrow E \cup e$
  $t_{\text{last full}} \leftarrow t$, $t \leftarrow t + 1$
  dijk $\leftarrow$ DijkstraProjection(N, E, n)
  mahas$_m \leftarrow$ Maha$_{nm} \forall (m, z_{nm}, \Sigma_{nm}) \in$ dijk
  mahas $\leftarrow$ lowest MaxMatches entries in mahas
  for $m \in$ mahas $|$ mahas$_m <$ MaxMaha **do**
    if LidarSimilarity$(n, m) >$ MinScore **then**
      matches$_m \leftarrow$ LidarSimilarity(n, m)
    end
  end
  for match $\in$ matches **do**
    AddToLoopValidator(match)
  end
  for $m \in$ NewlyValidatedLoopClosures() **do**
    $e \leftarrow$ CreateEdge($n, m,$ MatchXform(n, m)))
    $E \leftarrow E \cup e$
    OptimizeFactorGraph(N, E)
  end
EndLoop

3.3.4 Masking functions (masks)

In this section, we describe several masks conforming to Eq. 3.1 that we use to evaluate our system.

3.3.4.1 Never

By always returning false, this masking function results in the worst-case baseline of dead-reckoning.

3.3.4.2 Always

Similar to a fully metric SLAM system, this mask creates scan-matching nodes at the highest allowable density, and attempts to continually scan match between these nodes. Because we are not combining scans into a global occupancy grid, this setup will be more similar to metric SLAM systems that also perform scan matching across individual lidar scans. This function serves mainly as a worse-case baseline for computational complexity in the number of scan matches.

3.3.4.3 Isovist Eccentricity

Inspired by a previous use of isovist eccentricity in a decision point detector [48], this mask calculates the eccentricity of the robot's 2D lidar scan, and applies a threshold to it. The eccentricity essentially works as a measure of the narrowness or "hallwayness" of the lidar scan. The lidar scan naturally forms an isovist by considering each scan point as a vertex connected to its two neighbors. We compute the image moments, M_{ij}, of this isovist by triangulating the polygon and calculating the moments analytically. Then we calculate the covariance matrix and eccentricity as:

$$C = \begin{bmatrix} \frac{M_{20}}{M_{00}} - \left(\frac{M_{10}}{M_{00}}\right)^2 & \frac{M_{11}}{M_{00}} - \frac{M_{10}M_{01}}{M_{00}^2} \\ \frac{M_{11}}{M_{00}} - \frac{M_{10}M_{01}}{M_{00}^2} & \frac{M_{02}}{M_{00}} - \left(\frac{M_{01}}{M_{00}}\right)^2 \end{bmatrix} \tag{3.6}$$

$$e = \sqrt{1 - \frac{\lambda_2}{\lambda_1}} \tag{3.7}$$

where λ_1 and λ_2 are the eigenvalues of the covariance matrix. We apply a threshold with hysteresis to this eccentricity value to decide when a node should be added to the graph. Unlike the previous work, we can be much simpler because we do not need to avoid false positives.

In order to also record nodes at dead-ends in the graph, we compute a measure of "drift" to determine if the robot is at an extreme edge of the isovist. This is computed as the Mahalanobis distance from the robot to the centroid of the isovist using the covariance matrix above. We apply

a simple threshold to this drift measure. The mask returns true if either the drift threshold or the eccentricity threshold passes.

3.3.4.4 Openings

This mask is also based on the idea of placing nodes at likely intersections or decision points. We first determine which directions are currently viable for the robot to take and filter these data over a small time horizon to help reject noise. Finally, the mask returns true if one of these directions has an angular difference from the robot's current trajectory between 40 and 90 degrees, indicating that the robot has approached a newly possible direction.

We determine these directions by first splitting a 2D lidar scan such that each continuous segment only has ranges greater than 5 meters from the robot, indicating a potential opening for the robot to travel through. We then refine the end-points because we want to find the narrowest part of each segment and determine if the robot will be able to fit. Finally we apply a minimum width threshold of $0.75\,\mathrm{m}$, the approximate width of the robot, and a minimum angle threshold of 9 degrees, to help with false positives.

To refine the end-point choices, we start by defining a center line from the robot that passes through the mean of the original end-points. Then we choose the point that minimizes the sum of the opening width and the square root of the perpendicular distance to the center line, considering only points whose projection onto that line is between 0 and 5 meters and are on the same side of that line as the original end-point. We do this separately for each end-point.

To filter and select from these openings, we sort them by opening direction and only choose openings that have enough measurements over time with a low enough standard deviation in their direction measurement.

In addition, because this function will not natively mark dead-ends as useful nodes, we add the condition that if the robot turns in excess of 150 degrees away from the vector of travel since the last node, it should also return true.

3.3.5 Dijkstra Projection Loop Validation

Because the masking function is a replaceable component of our system, we need a method for validating loop closures that does not make strong assumptions about the structure of nodes available for scan matching.

Our loop validation system maintains a set of loop closure hypotheses. When it finds a valid loop involving a hypothesis, it rewards that hypothesis a vote. After a threshold of votes (we use 6), that loop closure is validated.

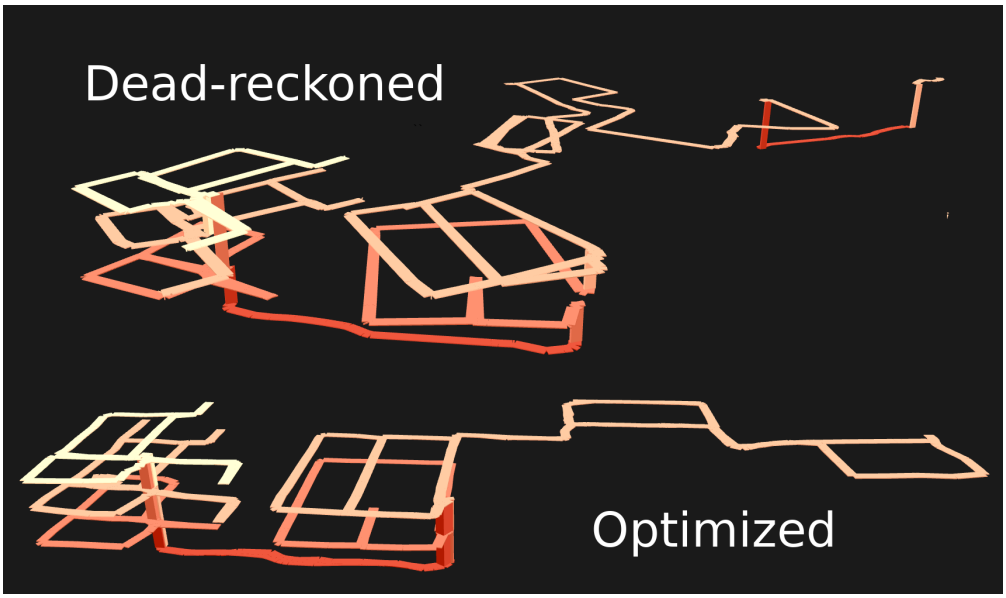

Figure 3.2: Our map both before (top) and after loop closures and optimization (bottom). Our gyroscope had an uncorrected bias that increases the angular error over time unless corrected. Given the large size of some of these loops, the loop closures may have to correct as much as 30 degrees of rotation in our map at a time. The optimized map used a Mahalanobis threshold of 5.0, minimum scan-matching score of 42.5%, and the openings mask.

The process of finding loops is based on the Dijkstra Projection [55], applied to finding the lowest error paths between the two nodes proposed for loop closure. In finding a path, we allow the search to use both validated and unvalidated loop closures in addition to odometry, always choosing validated over unvalidated loop closures when possible. From this minimum-error path, we form a complete loop by adding our loop closure edge. The resultant transformation should be unity. We judge the quality/error of the loop closure by the Mahalanobis distance between that final transformation and unity. If the final error is low enough (below 2), we give our new provisional loop closure and any other unvalidated loop closures on this path a single vote towards validation. Regardless of whether we accepted that prior loop, we repeat the process, first removing whichever edge in that last path was most redundant, or reachable from the greatest set of nodes in the search, indicating the potential presence of an alternative path. We continue this process until we either have enough votes to validate our new loop closure, or until a path can no longer be found to make another loop. In that case, the loop closure still has a chance to be validated later the next time we attempt to validate a new loop closure.

Table 3.1: The number of scan matches that each masking function ends up making over the course of map creation, the number that pass the minimum score threshold, the number that are validated as loop closures, and the total time used for map creation. The always mask uses five times as many scan matches. The openings mask is more economical than eccentricity, getting more matches validated even with fewer attempts. These numbers used a minimum scan-match score of 42.5% and maximum Mahalanobis threshold of 5.0.

Masking func.	Total	Matched	Validated	Time (s)
Always	5353	4437	3833	195
Eccentricity	1172	860	627	50
Openings	974	760	637	46

3.4 Evaluation

To evaluate our system, we drove our robot through three buildings and four floors on the University of Michigan North Campus. The robot navigated a total of about $2.5\,\mathrm{km}$ to collect the data over 2.5 hours. The resultant map has a total path length of about $1.9\,\mathrm{km}$ and a total bounding area across four floors of about $34\,000\,\mathrm{m}^2$. The third floor has the largest individual bounding area of about 264 by 82 meters.

Our robot's fiber-optic gyro has an angular bias which we did not correct, so that the performance would be more similar to a less expensive sensor. This angular bias can be seen in the unoptimized dead-reckoned map in Figure 3.2.

From these data, we construct and evaluate our masked metric maps to examine the trade-offs between the number of scan matches used, the accuracy of the final maps, and the sensitivity of our method to key parameters.

We evaluate the performance of our system by counting the number of scan-matching attempts, the number of successful matches, the number of matches that are validated as loop closures, and the computation time, shown in Table 3.1. The ideal mask function would have low computational requirements, using only the minimum number of scan-matching attempts in order to form its loop closures. It would also only choose scan matching locations that are not prone to perceptual aliasing.

We evaluate the accuracy of the final maps by checking for important loop closures. With all the correct loops closed, every path we find in the map's graph should be as short as physically possible. If loop closures are missing, some paths will be longer. If incorrect loop closures are present, some paths will be shorter. To measure this, we calculate the lengths of paths in a randomly selected set of 999 starting and ending position pairs. We use the same set for all cases, and define the positions by their timestamp in the log file. This lets our comparison be fair because while each map may choose to place nodes in different places, the timestamps can still uniquely define

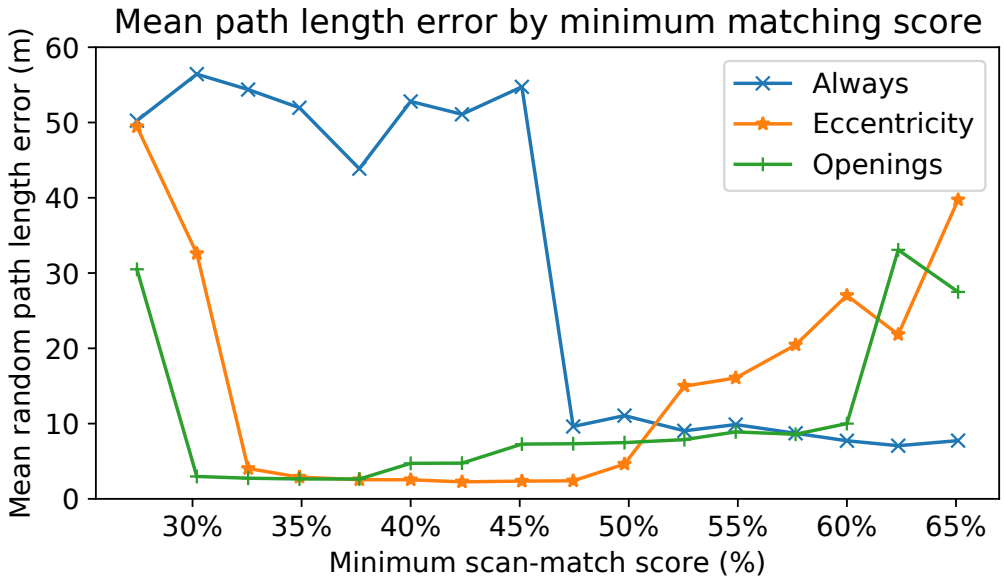

Figure 3.3: Dependence of each masking function on the minimum matching score. A score of 100% would mean that the scans perfectly overlap. An error of close to $0\,\mathrm{m}$ indicates that all the loop closures are valid. Starting with more attempts, the always mask benefits from stricter thresholds to help manage false-positives, whereas the openings and eccentricity masks benefit from low match scores because the nodes are already less prone to perceptual aliasing. The Mahalanobis threshold is 5.0 for these runs.

locations. For each generated map, we interpolate the positions of the timestamps based on the two nodes in the map that precede and follow chronologically, and we calculate the shortest paths between these interpolated positions. We manually verify several maps to have no false-positive loop closures, although they may be missing loop closures, and take the minimum length of each path across them to use as ground truth. From these ground-truth lengths, we can calculate the mean path error of any map.

3.5 Discussion

By manual inspection we were able to find that both the openings and eccentricity masks were able to generate maps without apparently missing or incorrect loop closures. An example of this is the map shown on the first page in Figure 3.1. On a standard laptop with an i7-6820HQ processor, this map took 46 seconds to generate from a 2.5 hour log file using the openings mask function, far exceeding real-time.

Based on the path errors shown in Figures 3.3 and 3.4, we see that the always mask often performs worse than the others. It was generally the case that the always mask would create false-positive loop closures due to perceptual aliasing between parts of the same hallway, and if the parameters were tuned to be more restrictive, it would fail to close a final loop at the end of the

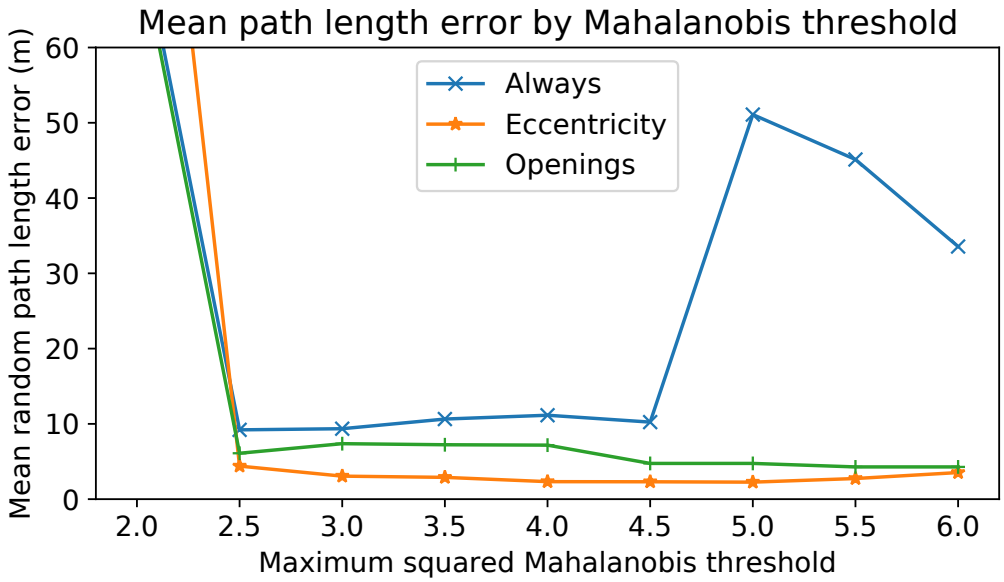

Figure 3.4: Effects of the threshold that determines which nodes are close enough to scan match against for potential loop closures. An error of close to $0\,\mathrm{m}$ indicates that all the loop closures are valid. While the openings mask benefits from lenient thresholds allowing more matching with farther-away nodes, the always mask needs a smaller threshold to help limit perceptual aliasing. The minimum scan-matching score is 42.5% for these runs.

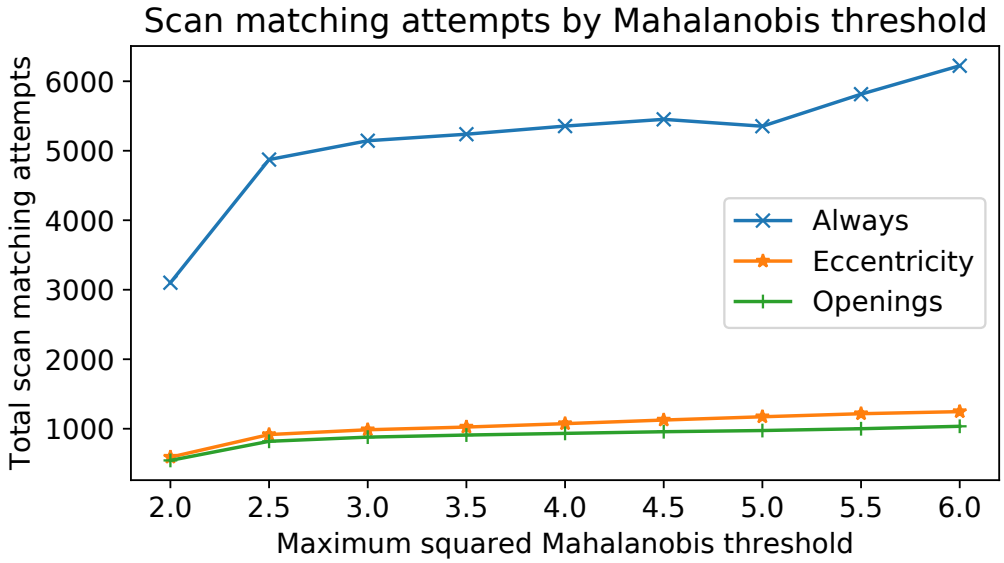

Figure 3.5: Scan matching attempts by Mahalanobis threshold. As the threshold increases/loosens, more scan matching candidates are identified and attempted. In our system, scan matching uses more computational processing than anything else, making it a proxy for the total computational resources needed. Furthermore, every attempt poses a risk for aliased nodes to incorrectly form a loop closure. Regardless of loop closing system characteristics, additional attempts can only increase the probability of false-positives. Our masks reduce the number of scan matches by five times.

trajectory when the odometry's angular error is high. These conflicting situations made the always mask hard to tune.

Increasing the scan-matching score threshold reduces the availability of loop closure candidates to all the masks. In Figure 3.3, we see that because masking reduces this availability even more, only the always mask benefits from this reduction, whereas the other masks benefit from using these lower-score matches. In Figure 3.4, we see that good performance can be reached as long as the Mahalanobis threshold permits attempting the right loop closures. Because the system always prefers to match against nodes with lower Mahalanobis distances, it is relatively unaffected by larger thresholds, although they lead to an increase in the number of total scan matches, as we see in Figure 3.5. For reference against these figures, the never/dead-reckoned map has an average path length error of $625\,\mathrm{m}$, because with no loops closed the map is treated as a one-dimensional chain.

Overall, we find that for our difficult map the always mask has the worst loop-closing performance while also being five times as computationally expensive. The openings and eccentricity masks both generate maps with low loop closure error, and that we can not visually distinguish from the the map presented in Figure 3.1.

3.6 Conclusion

Continually scan matching comes at a cost, requiring more computational resources and increasing the likelihood of failure through perceptual aliasing. In this paper we presented a flexible mapping scheme for creating sparse metric maps with a variable masking function that selects locations of interest for closing loops. Precise localization is rarely needed when driving down a hallway. This system is computationally efficient because it minimizes scan matching and optimizes a sparser map. We showed that our system can create high quality maps even in large environments with long loops and poor odometry. We compared two simple masking functions based on openings and isovist eccentricity with an "always" function that emulates a traditional dense SLAM system, and found that our masking functions produced maps with more true-positive and fewer false-positive loop closures, using only one-fifth as many scan matches.

3.6.1 Acknowledgements

This work was supported by grants from the NSF (1830615) and ARC (W56HZV-19-2-0001). Distribution A. Approved for public release; distribution unlimited. (OPSEC 3923).

Disclosure: Advisor Edwin Olson has a financial interest in a company that may have rights to foreground or background technology described in this paper.

CHAPTER 4

Monte-Carlo Policy-Tree Decision Making (MCPTDM)[1]

4.1 Introduction

In this chapter, we describe Monte-Carlo Policy-Tree Decision Making (MCPTDM), an uncertainty-aware framework for high-variance planning problems with multiple dynamic agents, such as autonomous-driving. We start by taking the existing Multi-Policy Decision Making (MPDM) framework and incorporating non-parametric advantages through random sampling and Monte Carlo tree search. Random sampling is an effective technique for exploring large possibility spaces that do not conform to simple models. This fits with our overall theme of using flexible and expressive techniques that avoid strict parametric models.

Planning with uncertainty is difficult because uncertainty compounds and marginalizing over each source of uncertainty is exponential in the number of dimensions. First, the space of possible action sequences increases exponentially with the length of the planning horizon and the number of dynamic agents. Second, uncertainty about the future world necessarily increases the further into the future we plan. The first difficulty poses computational challenges, while the second means that continuous re-planning is necessary to take advantage of new information.

Planning under uncertainty is often modeled as a partially observable Markov decision process (POMDP) [57][58], with a discrete set of states, actions, and observations, and with probabilistic state transition, observation, and reward functions. Exactly solving a real-world POMDP is intractable because of the exponential nature of the probabilistic belief space. Tools that approximately solve the exact POMDP are still only tractable for small discrete problems. More realistically, this computational cost can be made tractable through use of heuristics [59], sampling approaches [60][61], or domain-specific modeling simplifications [62]. Even so, the number of future scenarios to consider is still an exponential function of the number of possible actions

[1] Adapted from Haggenmiller and Olson [14]

(branching factor) and the length of the horizon (search depth). A brute-force tree search over all possible plans will only be possible when the action space is both discrete and small, the horizon is short, and the time discretization is coarse.

The Multi-Policy Decision-Making [17] (MPDM) framework is helpful for planning problems like these because computation time is linear in both the number of policies and the length of the planning horizon. Instead of planning in action space and directly considering each possible control input, MPDM plans in policy space and only considers selecting from high-level closed-loop policies that encode domain-specific behaviors. MPDM handles uncertainty in other dynamic agents by sampling their states and assuming that they are also following policies, again limiting the computational complexity. By having policies that encode the breadth of reasonable behaviors for both the ego (the agent that we are planning for) and other agents, MPDM takes advantage of prior domain knowledge to avoid searching extremely unlikely and unrealistic portions of the complete search tree. Policies can also be used for both discrete and continuous action spaces.

On the other hand, MPDM imposes some severe constraints to get these advantages. For each Monte Carlo sample of the belief, the ego and other agents must be modeled by a single closed-loop policy for the duration of the planning horizon, precluding the possibility of switching policies within that horizon. This makes certain larger-scale behaviors, such as an autonomous vehicle passing another vehicle and then returning to its original lane, much more awkward to handle. Each variation on such a behavior would need to be modeled by a separate closed-loop policy. If the number of variations starts to become exponential, then we no longer have linearity in computation time for the number of closed-loop policy primitives. This is the core compromise of MPDM, that policies are static within the planning horizon.

Efficient Uncertainty-aware Decision-Making (EUDM) [18] extends MPDM to help get around this limitation by using a tree search to allow up to one policy change at some future point in the planning horizon and also by using heuristics to identify situations with the obstacle agents that may lead to dangerous situations. This helps EUDM more effectively marginalize over uncertainty in the initial states and plans of the other agents. Even with policies, however, the number of possible initial belief states is still exponential in the number of obstacle vehicles to plan around.

Our goal in this chapter is to allow flexible policy switching for the ego agent in the planning horizon, where MPDM cannot switch policies at all and EUDM allows a single change. We accomplish this by combining insights from both MPDM and Monte Carlo Tree Search (MCTS) along with additional novel modifications that take advantage of the unique cost-structure and focus on safety in autonomous driving and other similarly structured tasks.

These novel modifications allow us to extract more information from a computationally-limited number of samples. The set of samples that visit a tree node form a non-parametric representation of the corresponding expected marginal cost, or immediate reward over the represented time

interval, for that node. The goal of MCPTDM is to use as few samples as possible to get the most accurate view of the expected cost probability density function and thereby choose the best path through the tree.

The principle contributions of this chapter include:

1. Monte-Carlo Policy-Tree Decision Making (MCPTDM), which allows an agent to efficiently compute policies in partially-observable and stochastic problems having either discrete or continuous action and state spaces. It does this by composing its policy from a sequence of simpler policies.

2. Two techniques: marginal action cost (*MAC*) estimation, which improves UCB-based tree exploration and final action selection; and "particle repetition", which reduces variance in cost estimation by reducing the effect of unlucky or unusual initial conditions.

3. Validation of our MAC estimation and particle repetition improvements on an abstract task and evaluation and comparison with MPDM [17] and EUDM [18] on an autonomous self-driving task.

4. Openly-released source code capable of reproducing all the figures and evaluation results from this chapter for full replication of our results and further extensions.

4.2 Background

In this section, we show how increasing generalization builds from all the way from the discrete Markov process to the continuous partially observable Markov decision process. See the Venn diagram in Figure 4.1 for how the generalizations build on each other. We also review Monte Carlo tree search.

4.2.1 Markov process

A Markov process, or Markov chain, is a stochastic model for describing states and how they change over time. For example, a Markov process could be used to model changes in weather with states for sunny, cloudy, and rainy days and with transition probabilities between each of the states. The goal of a Markov process is to predict the probability of future states given a current state, such as the probability of a sunny day on Saturday given that Monday is rainy. More formally, a Markov chain is a tuple (S, T) where S is the set of possible states and $T(s'|s)$ is the transition probability from state s to the next state s'. Because the transition probability to any new state only takes into consideration the single previous state, the Markov process is described as "memoryless"; this is called the Markov property.

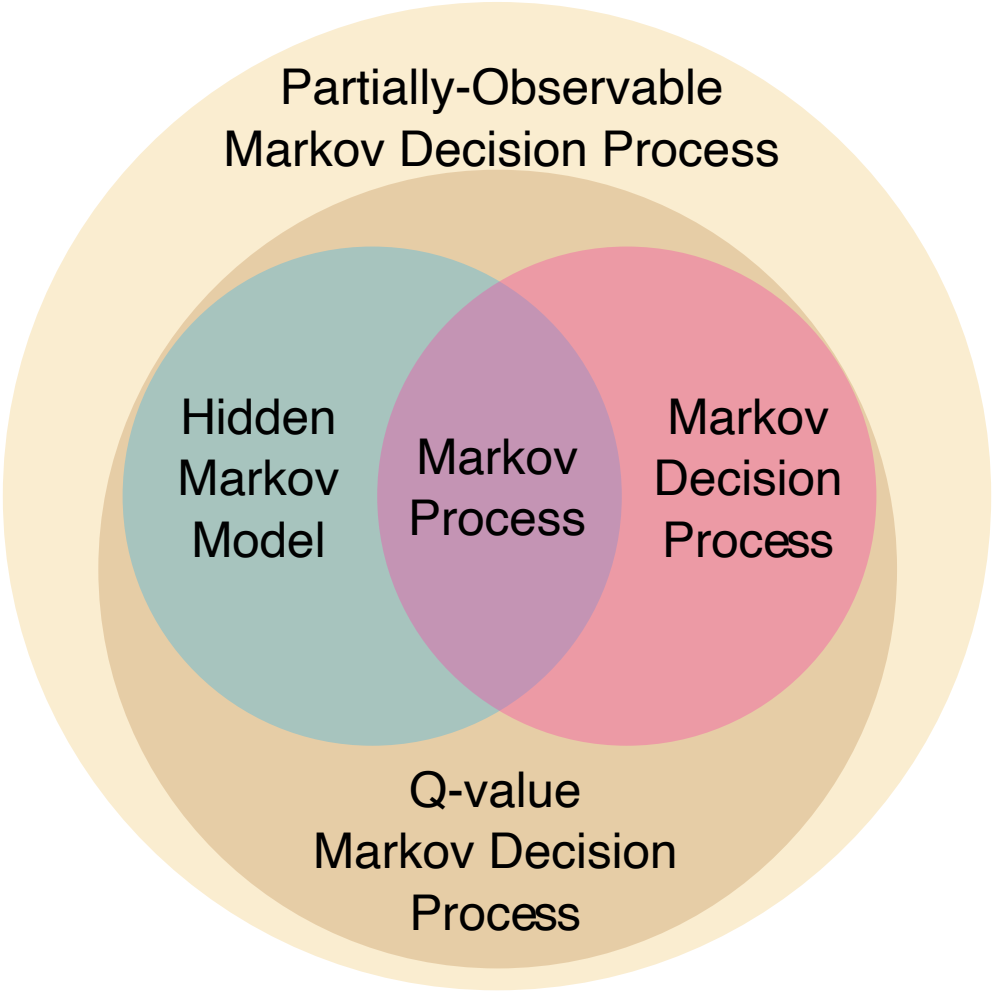

Figure 4.1: Venn diagram showing the generalizations that compose partially-observable Markov decision processes (POMDPs). The most basic abstraction is the Markov process. The hidden Markov model (HMM) and Markov decision process (MDP) are both extensions to this. The POMDP abstraction includes both of these extensions, and the Q-value MDP (QMDP) is a specific form of POMDPs.

4.2.2 Hidden Markov model

One extension to the Markov process is the hidden Markov model (HMM). In this model, states can not be directly observed, but there are stochastic observations that provide some information about the true states. The goal of an HMM is to infer the true latent states from only the observations. For example, an HMM could be used to predict the weather based only on spending trends, since certain purchases are more or less likely depending on the weather, such as ice cream on sunny days and umbrellas on rainy days. More formally, an HMM is a tuple (S, T, O, Z), where O is the set of possible observations and $Z(o|s)$ is the probability of observing o when the true state is s. S

and T are defined the same as in a Markov process.

4.2.3 Markov decision process

A different extension to the Markov process is the Markov decision process (MDP). In this model, there are action choices to be made in order to maximize expected rewards. For example, controlling a robot arm to move from one pose to another could be modeled as an MDP with a reward for achieving the final pose, with costs (negative rewards) for each action/time taken, and with a transition function that takes into account uncertainty in the robot's actuation. An MDP is also parameterized by a discount factor that balances immediate and future rewards. More formally, an MDP is a tuple (S, A, T, R, γ) where A is the set of actions, $T(s'|s, a)$ is the transition probability from state s with action a to the next state s', $R : S \times A \times S \to \mathbb{R}$ is the reward function, which may be a function of the state, action, and resulting new state, and γ is the discount factor. Set of states S is defined the same as in a Markov process.

The objective of an MDP is to find an optimal policy $\pi^* : S \to A$ that maximizes the expected discounted reward: $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})]$.

4.2.4 The Partially Observable Markov Decision Process (POMDP)

When we include both the extensions of the hidden Markov model and the Markov decision process, we get the partially observable Markov decision process (POMDP) [57][58]. In this model, action choices are made in order to maximize expected rewards, but the true latent state can only be inferred through the stochastic observations. More formally, a POMDP is a tuple $(S, A, T, R, O, Z, \gamma)$ with: set of states S; set of actions A; transition probability $T(s'|s, a)$; reward function $R : S \times A \times S \to \mathbb{R}$ of state s, action a, and next state s'; set of observations O; observation probability $Z(o|s, a, s')$; and discount factor γ.

As an example, a self-driving vehicle scenario could be modeled as a POMDP. While direct observation of close-by vehicles can be used to determine their current poses, their intentions (latent state) can only be inferred from their behavior. As all the vehicles execute their actions, there may be noise attributable to the environment (road surface, wind, etc.) or the vehicles themselves (limited precision engine control and steering). The ego-vehicle has a goal to achieve for a positive reward and desires to achieve that goal safely and efficiently with negative rewards for time and unsafe conditions.

In the classic formal definitions, the POMDP and the other models above are all defined with discrete state, action, and observation spaces. In practice, while continuous action spaces can be discretized, better performance can be achieved by considering continuous versions. In this thesis, we will generally refer to the continuous POMDP model as simply the POMDP.

Sampling-based POMDP solvers often do not require explicit probability distributions for the transition and observation functions. A *generative model* of a POMDP implicitly defines the transition and observation distributions, only providing samples $s' \sim T(s'|s,a)$ and $o \sim Z(o|s,a,s')$.

Littman's POMDP tutorial [63] is a friendly in-depth introduction to POMDPs for the curious reader wishing to know more.

4.2.5 QMDPs and the Q_{MDP} solution to a POMDP

The QMDP [15] is a simplified/approximate version of a POMDP where state uncertainty is assumed to only exist at the beginning of the problem in the root belief. If all the uncertainty in a POMDP can be factored out to the root belief, then that problem is a QMDP. Navigating a world with moving obstacles is a QMDP if those obstacles have predetermined but unknown trajectories, because that unknown information is all determined at the beginning of the problem. Even if the trajectories are not predetermined but have continuous random changes, as long as the ego-agent has no way to incur a cost for information to better predict their trajectories, we can still factor out the uncertainty and consider it to all have been predetermined. If the choice for more information is free, we just reformulate the problem to always get this information, and then it can also be considered to have been predetermined. Only when we have to decide to pay a cost for information do we leave the QMDP abstraction and our problem is a full POMDP, because planning with different amounts of information in the future is incompatible with the assumption of uncertainty only existing in the root belief.

The notation $Q_{MDP}(s,a)$ denotes the expected value of a state and action in an MDP, and if we are solving a POMDP and then suddenly have full knowledge/observability of the state, the POMDP is effectively reduced to just the MDP part that it contains. The notation $Q_{MDP}(b,a)$ denotes the expected value of a belief and action in a POMDP if after taking action a, the belief becomes fully observable and the remainder of the problem can be solved as an MDP.

Any POMDP where it actually holds that all the state uncertainty can be factored out to the root belief is therefor a QMDP because the true value of $Q(b,a) = Q_{MDP}(b,a)$. And if a POMDP is close enough to a QMDP it may also be much easier and more efficient to solve as a QMDP, because a QMDP solver does not need to maintain uncertainty anywhere except for the root of the search tree. Whether this approximation is worthwhile depends on the degree to which taking costly actions for additional information needs to be prioritized in the planning.

4.2.6 Monte Carlo tree search (MCTS)

Monte Carlo tree search is a planning/search algorithm that enables efficient and approximate solutions to problems that would otherwise be intractable. As with any tree search algorithm, we

start with an initial state from which the algorithm needs to choose one action to take. The tree represents actions as edges and states as nodes. A node can also be understood as corresponding to the single action/edge that leads to it. The full tree itself can be arbitrarily deep depending on the state (like reaching check-mate in chess), or have a fixed depth (like driving with a 10-second horizon). Non-random tree search algorithms work by either exhaustively exploring all possible chains of actions and outcomes, or by eliminating branches and subtrees as provably suboptimal. Domain-specific properties can also improve computational feasibility, for example by restricting the possible actions or by guiding how the tree is explored.

The general principal of Monte Carlo tree search, considering only the dominant form based on or inspired by Upper-Confidence bound applied to Trees (UCT) [64], is to randomly explore the most promising areas of the full search tree and to progressively refine the quality of the estimated solution. Because we do not exhaustively explore the whole tree, we have the flexibility to stop at any point and take our current approximate solution. In addition, because the full search tree may be arbitrarily large, we do not ever represent this full tree in memory. MCTS starts with just the root node as a leaf node and continues from there representing as little of the full tree as possible.

A general abstract model of MCTS [65] is shown in Algorithm 3. First, the *tree policy* takes a path from the root node n of the tree to a potentially-new leaf node n'. This path is generally made by taking the child with the highest upper confidence bound at each node [64], using random selection to break ties, but other policies are possible. The tree policy is generally allowed to add new leaf-node children to a single existing leaf node each time it runs, so paths that it chooses repeatedly will gradually get deeper and deeper. Next, the *default policy* estimates the value of node n', for example by simulating random actions until a terminal state is reached, or just using a heuristic. Finally, this value is propagated back up the tree and the estimated values for each node along the path taken are updated. These three steps are repeated until the computational budget is met and finally the action corresponding to the "best" child of the root node is returned. This is generally either the highest-value child or the most visited child (e.g. the child that most consistently had the highest upper confidence bound on its value and so was chosen the most).

Algorithm 3: General abstract MCTS

function CHOOSEACTION(s_0)
1 $n \leftarrow$ CREATENODE(s_0)
2 **while** time remains **do**
3 $n' \leftarrow$ TREEPOLICY(n)
4 $v \leftarrow$ DEFAULTPOLICY(n')
5 BACKUP(n', v)
end
6 **return** action of best child of n

While this general model gets across the basic form of MCTS, it also leaves a lot of important implementation details unspecified. In Algorithm 4 we give a fuller picture of a practical MCTS implementation. Instead of having the tree policy, default policy, and backup operations proceeding sequentially, they are embedded in a single recursive function. In this *recurse* function, our tree policy runs *expand* to make sure that node n has a child for each possible action. Then, it calculates the upper-confidence bound (UCB) metric for each child and chooses the child n' with the best UCB score, using randomness to break ties, according to the upper-confidence bound for trees (UCT) algorithm [64]. If child n' is new, meaning it has never been chosen before, then we forward simulate the state of n with the action of n' to compute an immediate reward for n' and the state of n', and the tree policy portion of the search is complete. Here we are making the assumption that the immediate reward and next state are both deterministic. The default policy then performs a random action rollout to estimate the value of node n'. If n' has already been explored before, then we simply recurse and continue with the tree policy one level deeper in the search tree. Finally, for the backup operation, we run the *update statistics* method. Here we keep track of both count N and sum V to conveniently compute the mean of all values that have been found for the children of n during the search. The full value of node n is then computed as the deterministic immediate reward earned by n plus the discounted mean of values of the children of n. By default we take the mean over all these values from the children and not simply the maximum child value because when the number of visits to some of the children is small, this would result in choosing not the best child, but the luckiest child [66].

As given, Algorithm 4 makes the assumption that the transition and reward functions are deterministic and only computes them once as needed on line 10, which is appropriate for applications such as Chess or other board games important to the history of MCTS. Alternatively, we could support stochastic transition and reward functions by having UCB on line 8 select an action instead of a child node, unconditionally computing forward-simulate, and using states to directly represent nodes in the tree, instead of assuming that an action node with a given parent state defines a unique reward and child state. As a second alternative, we could eliminate the direct correspondence between tree nodes and states and instead have each node represent a history of actions. This is an approximation because different states with the same history may require different action choices, which this history model would not support. However, this history model may be necessary to support a continuous state space where exact states will never be revisited.

Many other variations are also possible: the expand function can be altered to support progressive widening; UCB can be replaced by an alternative; update-statistics can compute the value differently. In many domains, intermediate rewards are not possible, and so the reward and discounting are omitted.

Algorithm 4: A fully expanded MCTS algorithm
For node n, $N(n)$ is the visit count, $V(n)$ is the value sum, and $Q(n)$ is the expected value. n' is a child node of n.

function CHOOSEACTION(s_0)
1 $n \leftarrow$ CREATENODE(s_0)
2 **while** time remains **do**
3 RECURSE(n)
end
4 **return** action of best/highest-Q child of n
function RECURSE(n)
5 **if** state of n terminal **then**
6 **return** UPDATESTATISTICS($n, 0$)
end
7 EXPAND(n) // Add child for each action
8 $n' \leftarrow \mathrm{UCB}(n)$ // Choose a child
9 **if** child n' is new **then**
10 FORWARDSIMULATE(n') // Compute n' state and reward
11 $v' \leftarrow$ ROLLOUT(n')
12 $v \leftarrow$ UPDATESTATISTICS(n', v')
else
13 $v \leftarrow$ RECURSE(n')
end
14 **return** UPDATESTATISTICS(n, v)
function UCB(n)
15 **if** n has any unexplored children **then**
16 **return** a random unexplored child of n
end
// UCB constant C controls exploration vs. exploitation
17 **return** $\arg\max_{n'} Q(n') + C\sqrt{\frac{\ln N(n)}{N(n')}}$
function UPDATESTATISTICS(n, v)
18 $\mathrm{N}(n) \leftarrow \mathrm{N}(n) + 1$
19 $\mathrm{V}(n) \leftarrow \mathrm{V}(n) + v$
20 $\mathrm{Q}(n) \leftarrow$ REWARD(n) $+ \gamma\ \mathrm{V}(n)/\ \mathrm{N}(n)$ // discount factor γ
21 **return** $\mathrm{Q}(n)$

4.3 Related Work

In this section, we categorize previous work into several families of methods: direct, simplifying, Gaussian, sampling, and neural network, and multi-policy (MPDM-like) methods.

Along with many other real-world problems, planning for self-driving vehicles can be modeled as a POMDP. Even when a POMDP model is not used explicitly, it can be a helpful framework for

comparing and discussing methods.

4.3.1 Directly solving the POMDP

Early work, like the seminal paper from Åström [57], focused on solving for the optimal belief-space regions and the actions they map to, as well as extending from these finite horizon problems to infinite horizon problems, which sometimes required finding approximate or ϵ-optimal solutions [67]. Even with small problems, these approaches were not very efficient, with one algorithm solving a machine replacement problem with two states and two actions in 110 seconds [68].

A more recent work on directly solving a POMDP, "Grasping POMDPs" [62] from Hsiao et al., constructs reduced-size abstract state spaces for simple robot manipulation tasks by taking advantage of constrained "compliant" motions (such as sliding along a surface), and the "guarded" motions that make or break such contact. For a two-dimensional grasping task, the authors formed a 408-state POMDP and solved it in less than 10 minutes with a general solver.

Where possible, direct solutions to a POMDP are theoretically robust, but few practical problems are simple enough to be solved like this.

4.3.2 Simplification through macro-actions

A macro-action is a closed-loop policy with a termination condition that can be selected by an agent as an option in addition to its primitive actions, and were initially introduced as a tool to speed up reinforcement learning [69].

Theocharous and Kaelbling [70] employ macro-actions to approximately solve a POMDP. They discretize the belief space into a sparse dynamic multi-resolution grid and use hand-crafted macro-actions, like go-to-end-of-corridor, to avoid producing and evaluating non-productive intermediate states. Macro-actions have also been generated online, such as in PUMA (Planning under Uncertainty with Macro-Actions) [71], which generates macro-actions based on sub-goals of reward and information gain and uses anytime refinement to progressively increase the resolution of their macro-actions.

Macro-actions share many similarities with the policies used in MPDM in that both are larger closed-loop abstractions over primitive actions, reducing the effective size of the search space in planning algorithms. The primary difference is that macro-actions have termination conditions and must still be composed, sometimes even with primitive actions. In MPDM, policies do not necessarily have a termination condition nor need to be composed together during planning. In this sense, these policies are less flexible than macro-actions, but allow for faster planning in exchange for this lower flexibility.

4.3.3 Analytical uncertainty propagation with Gaussians

In some cases, a POMDP can be linearized in such a way that uncertainty can be represented and propagated as simple Gaussian distributions. In these cases, planning decisions can be made directly with respect to the analytical likelihood of acceptable outcomes.

For example, in Linear-Quadratic Gaussian Motion Planning (LQG-MP) [2], the authors linearize and jointly model motion planning, control, and state estimation with Gaussian uncertainty. This enables the planner to precompute the robot's covariance and potential deviation from its nominal path and choose the path most likely to succeed from some set of candidates. Similarly, FIRM (Feedback controller-based Information-state Road Map) [72] precomputes a probabilistic roadmap "FIRM graph", where each directed edge is a linear quadratic Gaussian (LQG) feedback controller designed to drive the belief into the neighborhood of the next node. So too the Rapidly-exploring Random Belief Tree (RRBT) [73], which iteratively builds and refines a tree of possible trajectories with local LQG feedback control to propagate state covariance, maintain chance-constraints, and avoid collisions.

By taking advantage of the composability of Gaussians with LQG control, these methods significantly factor out uncertainty by handling it analytically without an exponential increase in computation. While this is a big advantage, modeling all uncertainty as Gaussian is not suitable for highly non-linear problems like autonomous driving.

4.3.4 Sampling-based approaches

In contrast to the computational gains possible when propagating Gaussians, sampling-based approaches recognize that arbitrary probability distributions can be approximated with enough random samples. The true value of some uncertain process can be determined by using sampling to marginalize over each source of uncertainty involved.

An early reinforcement learning approach extended the use of Monte Carlo sampling from MDPs to POMDPs [60], alternating between estimating Q-values for the current policy and then improving the policy accordingly.

Because even representing the whole state space to store probabilities or Q-values can be intractable in larger problems, most approaches only represent parts of the space that they have explicitly explored. The Infinite POMDP (iPOMDP) algorithm [74] assumes that the target POMDP has an unbounded discrete state space and then extends an infinite hidden Markov model (iHMM) [75] to also handle actions and rewards, estimates belief with an approach based on beam-sampling [76], and uses a stochastic forward search for action selection. Partially Observable Monte Carlo Planning (POMCP) [61] approximately solves a POMDP given only a black-box simulator with Monte Carlo sampling, an unweighted particle filter to approximately represent the belief state, and MCTS

to explore the action space.

For multi-agent scenarios, factored-value POMCP [77] extends POMCP to multi-agent problems by keeping separate trees for each factor (which may contain 1 or more agents) and applying the variable elimination algorithm [78] with their "mixture of experts optimization" to choose actions. Alternatively, decentralized Monte Carlo tree search (Dec-MCTS) [79] performs multi-robot planning by having each robot alternate between stages of growing/deepening a Monte Carlo search tree and updating the probability distributions of other robots.

Sampling is an effective way to explore high-dimensional spaces and to marginalize over uncertainty. Effective planning often only needs a representative sample of situations. In MCPTDM, we also use sampling to marginalize over uncertainty in our belief.

4.3.5 Neural-network learning

To solve a POMDP, a neural network may be trained with inputs corresponding to the ego agent's state and beliefs and with a loss function based on the cost or reward. By training a neural network to solve a POMDP, the network can hopefully encode expert rules and heuristics without the algorithm itself needing to specify this domain knowledge. This allows the algorithm to maintain more generality by only needing domain-specific knowledge for high-level objectives through a reward or cost function.

Deep Mind's Alpha Zero [80] plays Go at an expert-level by combining MCTS with deep learning networks for evaluating board positions and selecting moves. As a fully-observable game with a discrete action space, Go is not a perfect analog for robotics applications, but its large search space and difficulty evaluating moves and states make it very interesting. A0C [81] extends Alpha Zero to work in continuous action spaces by using progressive widening, by sampling and selecting promising new with the policy network, and by training the policy network to produce a valid probability distribution from the relative MCTS results by using the loss to enforce that it integrates to 1.

When actually applied to partially observable scenarios, most methods use deep learning for only part of the method. Ding et al. [82] use a recurrent neural network to examine multi-agent interactions and predict behaviors before they can be directly observed. Paxton et al. [83] use reinforcement learning to learn both low-level control policies as well as high-level goal-directed "option" policies to search through with MCTS; they use extracted feature inputs and a single hidden layer. Mukadam et al. [84] use deep Q-learning to find a high-level tactical lane-changing strategy that determines when to elect lane changing or acceleration actions to be performed by a low-level controller.

Because many neural network approaches focus on learning policies, they could be integrated

into an MPDM-like framework. In this chapter, however, we focus more how to use MCTS to compose simple policies, rather than how to come up with these policies ourselves. A neural network approach could be an alternative to hand-written closed-loop policies.

4.3.6 Multi-policy approaches

In Multi-Policy Decision-Making (MPDM) [17], the ego agent elects one policy from a small set of high-level closed-loop policies and also models the other dynamic agents with these same policies. Monte Carlo sampling is used to marginalize over uncertainty in the state and future policies of the obstacle agents. In each Monte Carlo sample, all the agents are forward simulated together to capture closed-loop interactions [85] and the best ego agent policy is elected from the cumulative results. Further work in the MPDM framework focuses on discovering risky configurations through stochastic gradient descent of a heuristic cost function [86] and back-propagation techniques [87].

Other approaches incorporation multi-policy ideas into a larger framework. Zhou et al. [88] use a POMDP solver to determine optimal acceleration/deceleration actions for each obstacle agent policy and a particle filter to estimate the probability of each policy-agent pair and then combine this with model predictive control for the ego agent. In Efficient Uncertainty-aware Decision Making (EUDM) [18], MPDM is extended with a limited tree search and critical-scenario heuristics, as mentioned in the introduction.

Our proposed method is similar to EUDM because we also extend MPDM by modifying the policy search process and permitting policy changes in the planning horizon. However, we go farther in searching over arbitrary sequences of policies, rather than allowing only a single change in policies. To make the larger search space tractable requires reformulating the search into an MCTS-like tree search, where instead of evaluating actions at each decision point, we evaluate closed-loop policies for a certain interval of time. This means that each node in the tree will no longer correspond to a single state, but to the belief of states that exist for this history.

4.4 Synthetic abstract scenario

We first examine Monte-Carlo Policy-Tree Decision Making (MCPTDM) with a synthetic scenario and experiments that model an abstract form of the self-driving scenario we will examine later in this chapter. After describing this scenario in detail, we examine the effects of changing the expected-cost rule used by UCB for balancing the exploration-exploitation tradeoff and for selecting the final best action. We also examine the effects of various improvements to UCB. Finally, we explore the idea of *fairness with particle repetition*, helping to mitigate the effects of "unlucky" initial conditions, where poor outcomes are more attributable to the initial conditions than the spe-

cific plan being evaluated. In a self-driving situation, for example, an "unlucky" particle might include nearby vehicles having intentions that box the ego vehicle in while another vehicle performs a dangerous lane-change; boxed in like this, it doesn't matter which policy the ego vehicle chooses, even though this random coordination is very unlikely. Of course, the important property here is not so much whether a particle is "lucky" or "unlucky" so much as that we reduce the variance error between our value estimates by using the same sampled initial condition particles.

4.4.1 Problem statement

We consider an abstract version of an autonomous driving task with five policy (or action) choices, a time horizon split into four segments, and costs related to avoiding crashes and close calls and making forward progress.

While costs associated with making forward progress are likely to be relatively smooth, costs around safety and potentially crashing are more discontinuous: while a near-crash may have a high cost, an actual crash will have essentially infinite cost because crashing is a binary phenomenon. To model a more complex cost distribution, we use a mixture of two Gaussians. This allows for samples to sometimes come from the lower-cost Gaussian and sometimes from the higher-cost Gaussian, without any samples necessarily being between the two. Gaussian mixtures have prior use in compactly approximating real-world events [89][90].

In addition, in a self-driving scenario, a lot of the uncertainty is in the initial belief about the behavior and intentions of other vehicles. Specific initial conditions that might be dangerous for one ego-agent policy are likely to be dangerous for the other policies as well. To model this "risky situation" correlation, we store the initial conditions in what we call *belief particles*. If the same belief particles are propagated through different paths in the tree, we should see correlated responses.

In order to determine the empirical performance of our method, we perform a synthetic experiment. We construct a tree (of depth 4 and branching factor 5) and assign each node a random cost probability distribution that is a mixture of two Gaussian distributions with random mean μ_i, standard deviation σ_i, and mixture weight w_i, where the mixture weights sum to one. Because our model includes only positive costs, we saturate these Gaussian costs to be in the range $[0, 2\mu_i]$ so that the mean is not changed by the saturation.

When running a trial, we first sample a "belief particle" from the "initial conditions" so that all costs using this belief particle will be correlated. We need this property, where every use of the same particle results in correlated costs, in order to simulate the idea of initial conditions. We do this by sampling z-scores from the standard Gaussian distribution (a z-score is a normalized Gaussian sample, where $z = (x - \mu)/\sigma$). We also sample a weight threshold t from 0 to 1 that we

will use to select between the Gaussian mixture components. With these z-scores and threshold z_1, z_2, t from the belief particle known, the cost c for all node distributions would be both correlated and deterministic:

$$c = \begin{cases} \text{clamp}(\mu_1 + z_1\sigma_1, 0, 2\mu_1) & t \leq w_1 \\ \text{clamp}(\mu_2 + z_2\sigma_2, 0, 2\mu_2) & t > w_1 \end{cases} \tag{4.1}$$

$$\text{clamp}(x, l, h) = \begin{cases} l & x < l \\ h & x > h \\ x & \text{otherwise} \end{cases} \tag{4.2}$$

For each trial, after sampling a situation, we use UCB (or one of its variants) with the selected expected-cost rule to run that situation through the tree. After running all trials in our computational budget, we choose the lowest expected-cost top-level action as our result. We calculate the true expected costs of both the best path through the tree and also the best path starting from our chosen action. Our *regret* for this choice is then the chosen-action best-path expected cost minus the overall best-path expected cost.

When sampling node distributions, Gaussian means and standard deviations are all sampled uniformly and independently from 0 to 100.

The true marginal expected cost of any node i, which we will use later to evaluate regret, is then:

$$\mathbb{E}(\bar{c}_i) = w_{i,1}\mu_{i,1} + (1 - w_{i,1})\mu_{i,2} \tag{4.3}$$

4.4.2 Expected-cost estimation with marginal action costs (MAC)

In this section, we motivate and describe a focus on *marginal action costs* (MACs), which allow us to make a more informed exploration of the search tree by better estimating node value as compared to just using terminal rollout costs. We will also describe several more in-between formulations for estimating node value and then in the later evaluation section we will compare them all to each other.

To put this idea in context, a common application of MCTS is for board games such as Go [80] which involve many turns, a large branching factor, determinism, and a reward/cost assigned only when a terminating condition is reached (e.g. win/loss). By comparison, in a non-deterministic self-driving car scenario, while there may be a long-term goal (e.g. a destination to reach), the most important goal of the planner is to maintain safety at all times. To this end, MACs allow the search to distinguish, for example, between a collision at depth 1 and depth 4, and due to the high

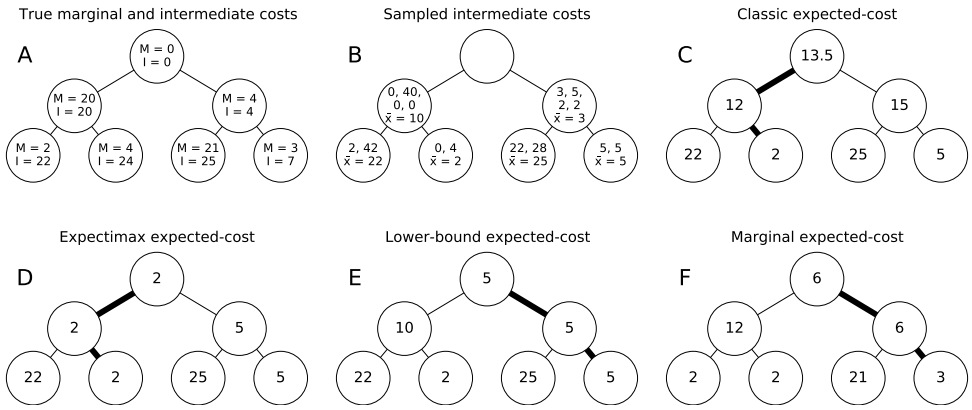

Figure 4.2: Illustration and comparison of four different rules for estimating the expected-cost of an MCTS-style node. Each rule's selected best path is in bold. For this example, only the "lower-bound" and "marginal action cost" (MAC) rules choose the optimal path. **A**: Each action choice and node has some cost distribution. We label the mean of each node's distribution as the "true marginal cost" M of that node. We also label the "true intermediate cost" I of that node, which is the sum of marginal costs leading to that node. The optimal choice in this example would be to choose actions right-right, resulting in a total expected cost of only 7. **B**: We run 2 trials for each leaf node, recording the intermediate costs of each trial after each action. In a more classical MCTS formulation, costs (or rewards) would only be known at the leaf nodes. We also label the mean intermediate cost at each node. **C**: Under the "classic" expected-cost rule, we use the mean value of every trial under a node, ignoring the intermediate costs of non-leaf nodes. **D**: An optimistic alternative "expectimax" is to set the expected-cost of a node to be equal to the best expected-cost of its children. This makes many actions appear much better than before. **E**: The "expectimax" rule is overly optimistic when a trial gets lucky and avoids a large intermediate cost in a parent node. We can use the mean parent intermediate cost as a "lower bound" on the expected-cost. **F**: Alternatively, we can also attempt to directly estimate the original "marginal action cost" of reaching each node from the intermediate costs. Then we set the estimated-cost to the marginal cost plus the best estimated-cost of a child.

cost of collisions, the entire sub-tree below a collision can be effectively pruned away. This is only possible when the collision can be attributed to a specific node.

The expected cost of MCTS nodes may be used at two different points in MCTS. In the first case, they are inputs to the UCB selection algorithm for guiding the exploration-exploitation trade-off in search. In the second case, after the computational budget for MCTS trials is met, they may be used for *final action selection*, to select the final top-level action to execute. Besides using the expected-cost, a common alternative is to choose the most-visited action as the final choice [66]. A slight improvement may be found by continuing to search until both these measures agree [91] and so we use that combined "max-robust child" variation in this chapter, putting a limit of 20% on the number of additional trials we might run. This explicit limit is only necessary because some of the parameter sweeps we perform include degenerate cases that do not converge.

4.4.2.1 Traditional MCTS expected cost estimation

The expected cost of an MCTS node is normally the mean of each trial that has passed through it [65]:

$$\bar{c}_i = \frac{1}{N_i} \sum_{k=1}^{N_i} c_{i,k} \tag{4.4}$$

where $\bar{c}_i$ is the expected cost of node i, N_i is the total number of trials that have passed through node i, and $c_{i,k}$ is the kth final trial cost that passed through node i. We call this rule "classic". (See Figure 4.2 C.)

Instead of just taking the mean of all terminal costs from all child nodes, we can optimistically take the best (lowest) child cost at each controlled choice, only averaging over the multiple stochastic costs at leaf nodes. The expected costs then bubble up from the bottom of the tree:

$$\bar{c}_i = \begin{cases} \min_{j \in \text{Children}(i)} \bar{c}_j & i \text{ is a branch} \\ \frac{1}{N_i} \sum_{k=1}^{N_i} c_{i,k} & i \text{ is a leaf} \end{cases} \tag{4.5}$$

This rule is called "expectimax" [92] (although we are minimizing costs here instead of maximizing rewards) and was originally devised in the context of game tree search where the opponent plays stochastically. It is considered a generalization of minimax search, where the opponent is assumed to play optimally [65]. (See Figure 4.2 D.)

4.4.2.2 Taking advantage of known intermediate costs

We observe that the above rule can be optimistic in some cases. For example, imagine a node following a risky action that has a 50% chance of observing a high cost, and that has two child nodes, neither of which impose any costs. If we run a trial for each of these children and get a high cost for one and zero for the other, we would optimistically choose the best option and assign an expected cost of zero to the parent node, even though the true expected cost is high. However, when we have intermediate costs for each node, we know that the high cost is attributable to the parent node and not to either of the children. We should not be deceived into believing that a low cost path exists for one of the children since the parent cost applies to both. We devise a new rule that takes the cost of the best child and applies a lower bound of the mean partial/intermediate cost of the parent node:

$$\bar{c}_i = \max \left(\bar{p}_i, \begin{cases} \min_{j \in \text{Children}(i)} \bar{c}_j & i \text{ is a branch} \\ 0 & i \text{ is a leaf} \end{cases} \right) \tag{4.6}$$

where $\bar{p}_i$ is the mean partial/intermediate cost of node i. We call this rule "lower bound". (See Figure 4.2 E.)

4.4.2.3 Marginal action costs (MAC)

Taking this idea of using additional information from intermediate costs even further, we can also directly calculate and assign marginal costs to each node and use the sum of mean marginal costs along the best path through the tree as the total expected cost:

$$\bar{c}_i = \bar{m}_i + \begin{cases} \min_{j \in \text{Children}(i)} \bar{c}_j & i \text{ is a branch} \\ 0 & i \text{ is a leaf} \end{cases} \tag{4.7}$$

where $\bar{m}_i$ is the mean marginal cost of node i, equal to the mean intermediate cost of node i minus the mean intermediate cost of the parent, if any. We notice a similarity to Q-learning in that the total expected value combines an immediate value with the best child node/successor state expected value. We call this rule "marginal" or MAC. (See Figure 4.2 F.)

4.4.3 Fairness and particle repetition

At the start of each Monte Carlo trial we sample a set of initial conditions. This *particle* then takes some path through the tree, influencing the final cost of the trial. Some of these particles may be either lucky or unlucky, making the actions and path they take look unfairly better or worse than they actually are. For example, if all the obstacle vehicles either simultaneously avoid or crowd the ego vehicle, such that no matter the ego vehicle's policy choice, there is either no possible crash or an inevitable crash. The idea is that with enough trials and particles, these effects will average out to be fair in the end, resulting in a good outcome. However, when the number of trials is computationally limited, it may help if we intentionally repeat particles along different paths through the tree to reduce any bias resulting from this good or bad luck.

We do this by recording all particles, the paths they take, and their terminal costs (because our initial conditions are compact, this does not add up to a significant amount of information). Then, after a top-level action has been elected for the next trial, we first check if there is a particle that has not gone down this path before. If so, we repeat this particle instead of sampling a new one. If there are multiple particles, we chose the one with the highest cost. We note that although it would be possible to perform particle repetition at any depth of the tree, we only found it worthwhile at the top level. This makes some intuitive sense because only at the top level of the tree do we actually make a decision between which action/policy to immediately take. Deeper in the tree, we do not care as much about a bias between individual branches so much as the overall bias that

bubbles up to the different top-level nodes.

Once the computational budget is large enough, repeating particles for fairness will no longer be necessary. Instead, it may be more effective to only draw new particles to better marginalize uncertainty. While for smaller numbers of Monte Carlo trials it is helpful to repeat particles as much as possible, for large numbers it may be best to not repeat at all. We find that the best number of particles to repeat appears to be inversely proportional to the total number of trials in our budget. This means that for an any-time implementation, it would be best to have an estimated budget before starting. We use a repetition constant to set the maximum number of particle repetitions to perform as the repetition constant divided by the number of trials in our budget.

4.4.4 Experiments

To evaluate and compare the different expected-cost rules, UCB variations, and particle repetition, we perform a variety of parameter sweeps. When not sweeping over Monte Carlo trials, we instead marginalize the number of trials with ten powers of two from 8 to 4,096. We perform enough complete runs of the algorithm to show significant results in the figures, where error bars indicate plus or minus one standard deviation of the mean (standard error).

4.4.4.1 On expected-cost rules

We start by sweeping the UCB constant (a free parameter) for each expected-cost rule. Figure 4.3 shows that the classic and MAC rules consistently outperform the expectimax and lower-bound rules, and that these differences persist even for large UCB constants which may encourage almost pure exploration. We reason that these differences reflect the effect of the expected-cost rules not on exploration, but on final action selection.

To tease apart how the expected-cost rules influences UCB and final action selection, we repeat the experiment, this time always using MAC for final action selection, but still varying the expected-cost rule for UCB. In Figure 4.4 we see this hypothesis confirmed, that the MAC rule is most important for final action selection and that most of the expected cost rules work just as well with UCB. We also include a uniform (pure exploration) rule for comparison.

We also compare expected-cost rules by computational budget (number of Monte Carlo trials) in Figure 4.5. Each rule uses the best UCB constant for it as found in Figure 4.4 and all use MAC for final action selection. We notice that MAC converges close to zero mean regret much faster than the other rules.

Algorithm 5: The MCPTDM algorithm. Inputs include initial state s_0, uncertainty belief b, and policy set P. We use p for policy, s for belief sample/initial conditions particle, n' for a child node of n, m for a marginal action cost, m_n for the set of marginal action costs observed by node n, and $\bar{c}_n$ for the expected cost of node n. In practice, an implementation may want to limit the number of repeated particles. A system may also want to specify a preference, in addition to UCB, to determine expansion order when multiple children are completely unexplored; we prefer to first explore the child with the same policy as the parent.

function CHOOSEPOLICY*(s_0, b, P)*
  $n \leftarrow$ CREATENODE(s_0)
  $k \leftarrow 0$
  for $k <$ trial_budget $\vee$ continue for max-robust child **do**
    RECURSE(n, DRAWBELIEFSAMPLE(b), P)
    $k \leftarrow k + 1$
  end
  return $\arg\min_p \bar{c}_{n'}$ for child n' with policy p
function RECURSE*(n, s, P)*
  if depth of $n \geq$ max depth **then**
    return
  end
  if n not expanded **then**
    // Add child nodes at depth+1 for each policy in P
    EXPAND(n, P)
  end
  $n' \leftarrow$ KL-UCB(n)
  if depth of $n = 0$ **then**
    // Save/retrieve particles *before* forward simulation
    if n' has unplayed particles **then**
      $s \leftarrow$ WORSTUNPLAYEDPARTICLE(n')
    else
      SAVEPARTICLE(n, s)
    end
  end
  $m \leftarrow$ FORWARDSIMULATE(n', s)
  $m_{n'} \leftarrow m_{n'} \cup \{m\}$ // Track marginal costs
  RECURSE(n', s, P)
  $\bar{c}_n \leftarrow$ MACEXPECTEDCOST(n, s)

4.4.4.2 On UCB variations

Acknowledging that there are many enhancements and alternatives to UCB, we also wanted to see if one of these would be able to improve performance. For this purpose we also implemented UCB-V [93], UCB(δ) [94], and KL-UCB and variation KL-UCB+ [16]. We performed parameter sweeps for each UCB variation to choose the parameters that produced the lowest regret for each, and

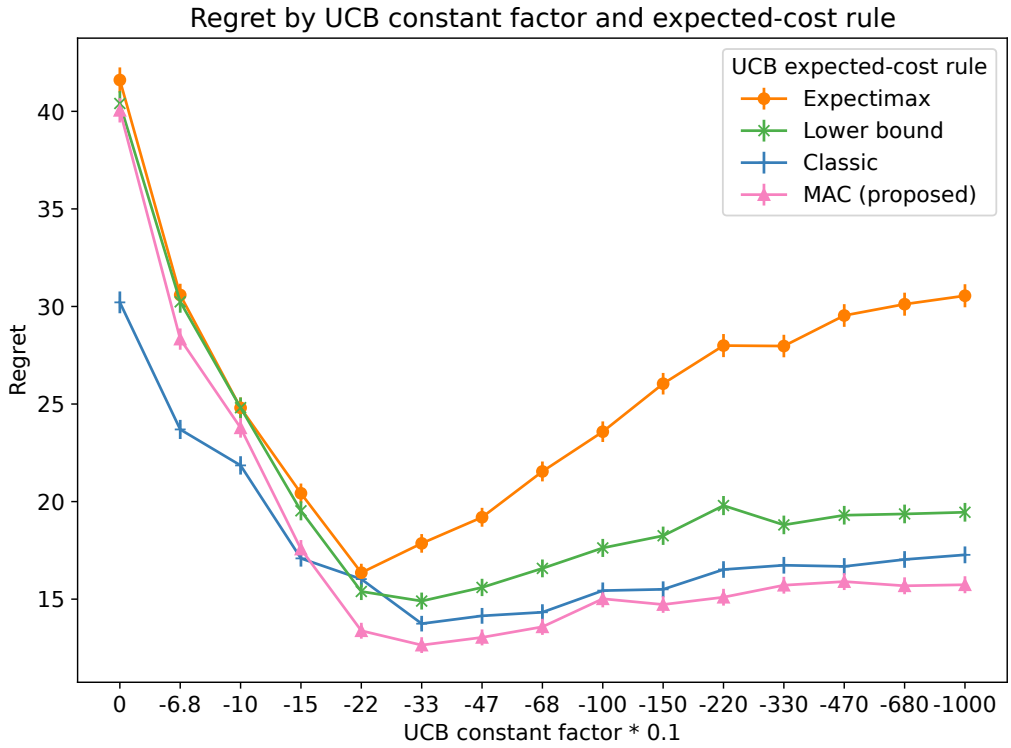

Figure 4.3: Parameter sweep of UCB constant for each expected-cost rule. Marginal action cost (MAC) and "classic" rules perform best. Lower regret is better.

then compared them. In Figure 4.6 we see that each improved variation performs fairly similarly and significantly better than UCB, although the relative differences start to widen with the highest numbers of trials. We use KL-UCB rule for the remainder of this chapter because of this advantage.

4.4.4.3 On particle repetition

In Figure 4.7 we sweep the repetition constant and show the *relative regret*, normalizing by the regret of the w/o-repetition case. For most of the cases, we see that improvements saturate when particles are being repeated as much as possible. Intuitively, we would expect that particle repetition would become less important as the number of particles increases, since the effects of individual "unlucky" particles would be mitigated by the large number of particles. We see exactly this behavior for 1,024 Monte Carlo trials in our experiment. Because we are plotting relative regret, the smaller the absolute regret, the larger the error bars. See the absolute regret trends in Figure 4.8 where we use a repetition constant of 2^{16}.

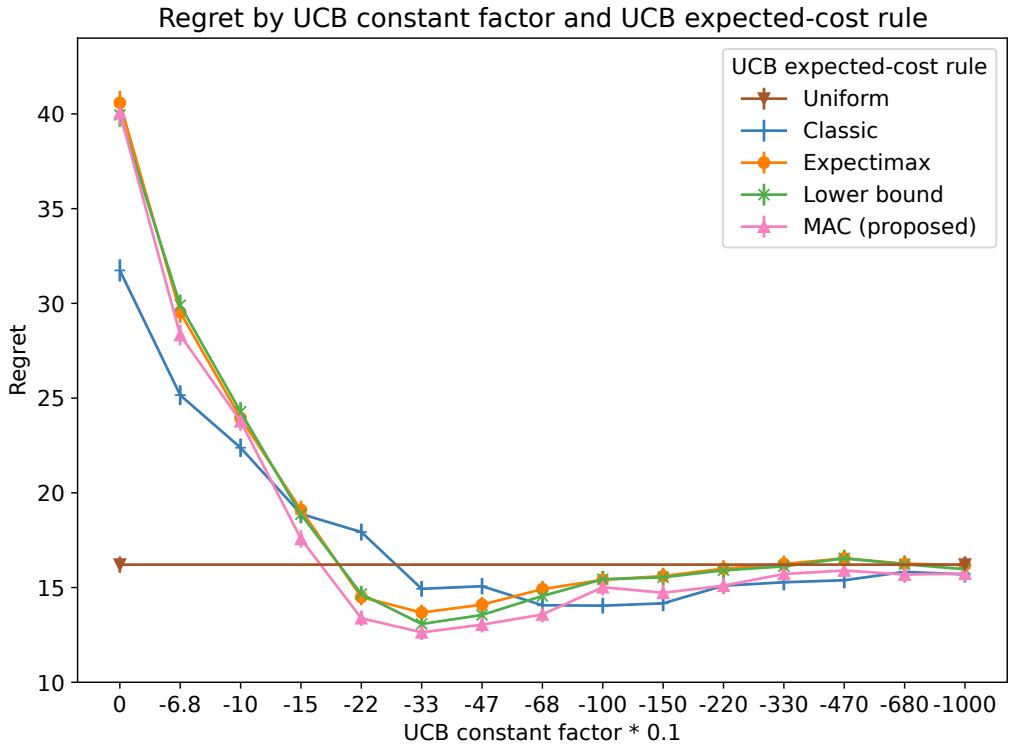

Figure 4.4: Parameter sweep of UCB constant for each UCB expected-cost rule, while using MAC for final action selection. Lower regret is better. We see that as UCB values increase, each rule's performance approaches that of uniform/pure exploration. For building up the search tree *before* final action selection, we find little difference in performance between expected-cost rules.

4.4.4.4 Cumulative improvement

Finally we perform an ablation study in Figure 4.8 to compare a traditional MCTS search with UCB and "max-robust child" final action selection to our enhanced method that adds in KL-UCB, MAC expected-cost estimation, and finally particle repetition. Each addition significantly reduces regret, although at different points along the curve.

Overall, our total combined method with MAC and particle repetition does significantly better than traditional MCTS. Our complete method is shown in Algorithm 5.

4.5 Autonomous Driving Scenario

We adopt an autonomous driving scenario very similar to that proposed by Zhang et al. for evaluating EUDM [18], but with only two-lanes going in a single direction (see Figure 4.9 and 4.10). The scenario uses a bicycle model, the intelligent driver model [95], and pure pursuit lateral control [96] for all vehicles, along with five policies: left-lane-maintain, left-lane-accelerate, right-

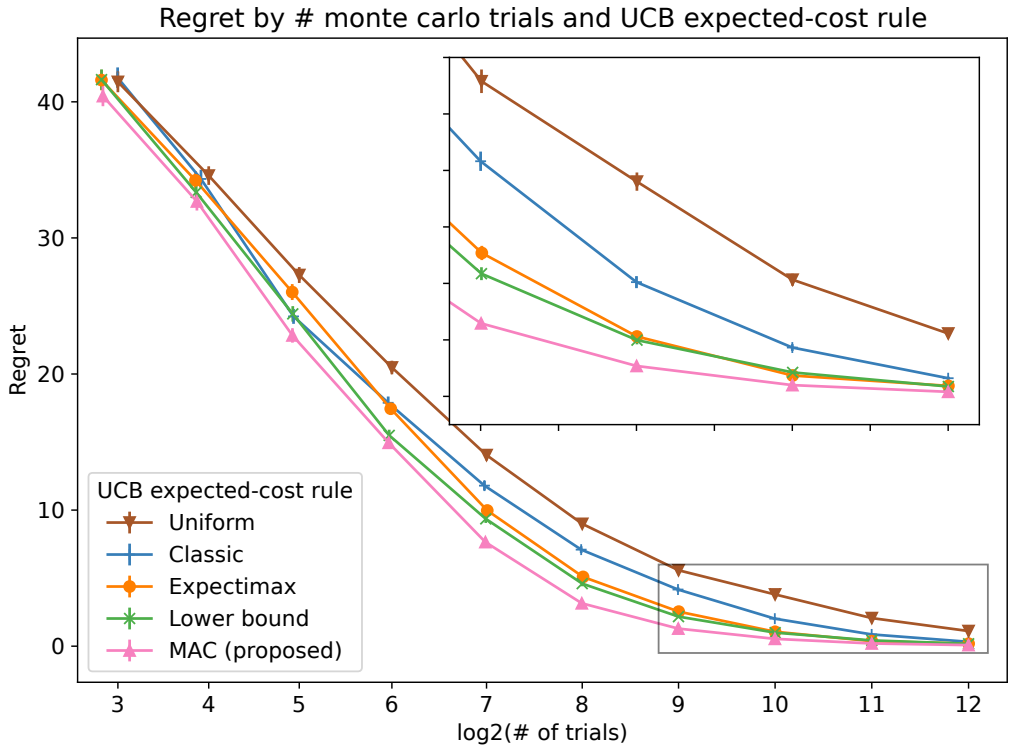

Figure 4.5: Parameter sweep of Monte Carlo trials for each UCB expected-cost rule, while using MAC for final action selection. Lower regret is better. MAC achieves a low regret faster than the other rules. Thanks to also using the "max-robust child" rule to make a final decision at a good time, uniform exploration also does surprisingly well.

lane-maintain, right-lane-accelerate, or decelerate. Electing a policy for a different lane than the current one causes a vehicle to perform a lane-change maneuver.

Besides the ego vehicle, we simulate 13 obstacle vehicles, and obstacle vehicles are removed and respawned so that we can maintain 13 vehicles within a certain distance of the ego vehicle. This number of vehicles ensures that there may be complex interactions between multiple other vehicles both in front of and behind the ego vehicle, but also keeps the environment from being too congested. Each obstacle vehicle is parameterized by a random (within some range) preferred velocity, acceleration, and follow-time, to provide some variation and uncertainty in their behaviors. Every 0.2 seconds, each obstacle vehicle has a small chance of randomly choosing a new policy (5% probability each second). Because obstacle vehicle changes occur randomly, the policies used by the obstacle vehicles will first check that the next lane is clear at least a half-vehicle's length ahead and behind before making a lane-change maneuver. The policies used by the ego vehicle do not make this check so they can be more flexible.

The ego car tries to safely and smoothly maintain a target velocity by minimizing a cost func-

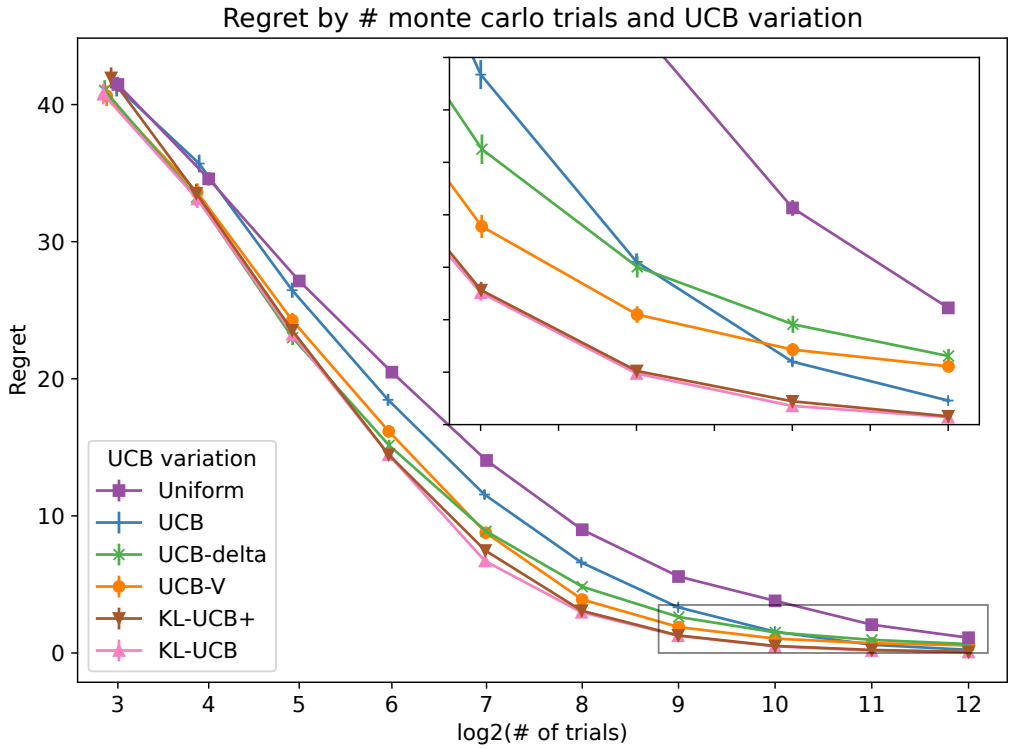

Figure 4.6: Parameter sweep of Monte Carlo trials for each UCB variation, using MAC for expected-cost and final action selection. Lower regret is better. All the improved rules outperform UCB in most cases by a significant margin. KL-UCB consistently performs best, with KL-UCB+ performing very similarly.

tion that incorporates velocity, safety, and control inputs:

$$C_{\text{vel}} = |v - v_{\text{target}}| \tag{4.8}$$

$$C_{\text{acc}} = W_{\text{acc}}\dot{v}^2 \tag{4.9}$$

$$C_{\text{steer}} = W_{\text{steer}}\dot{\theta}^2 \tag{4.10}$$

$$C_{\text{safety}} = W_{\text{safety}}(1 + e^{-k_{\text{safety}}(d_{\min} - d_{\text{safety}})})^{-1} \tag{4.11}$$

$$C = \int (C_{\text{vel}} + C_{\text{acc}} + C_{\text{steer}} + C_{\text{safety}})\alpha^t \, \mathrm{d}t \tag{4.12}$$

where v and θ are the ego vehicle's forward velocity and angle, $v_{\text{target}} = 11.2\,\text{m/s}$ is the ego vehicle's target velocity (25 MPH), W are the cost weights, $d_{\min}$ is the minimum distance between the ego vehicle and any other vehicle, k_{safety} and d_{safety} define the shape of a logistic sigmoid used for safety penalties, and $\alpha = 0.8$ is a discount factor.

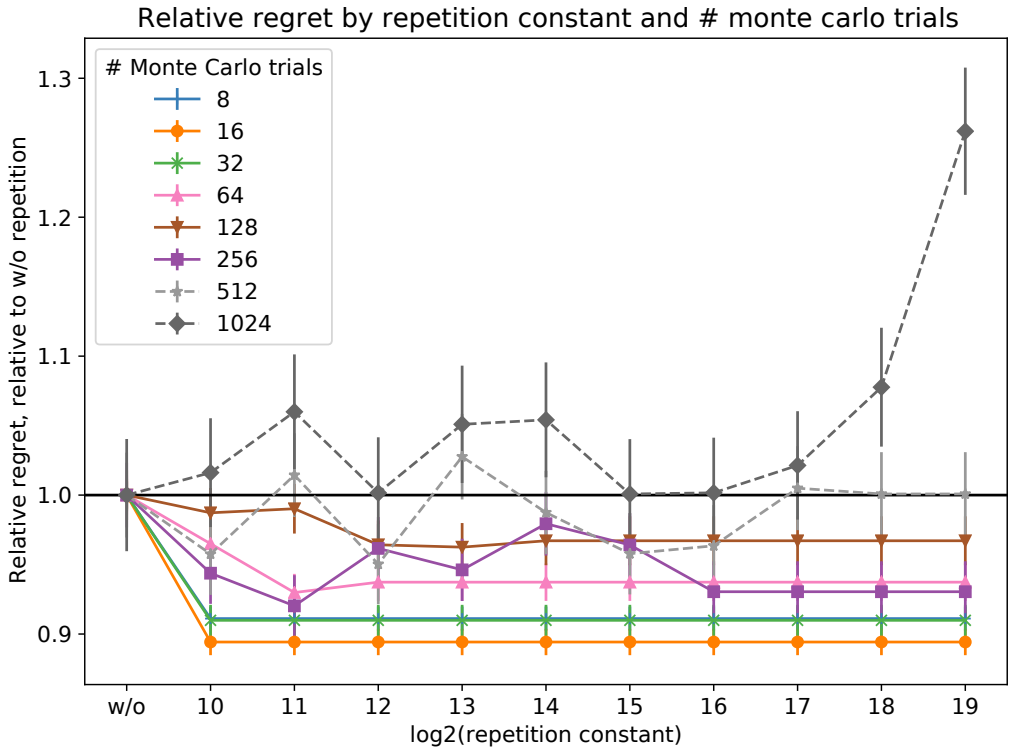

Figure 4.7: Plot of relative regret (normalized by the no-repetition case), the particle-repetition constant, and the number of trials. Particle repetition is strictly beneficial at least up to 256 trials, with up to about a 10% reduction in regret. Note that the cases with 1,024+ trials all have low absolute regret (see Figure 4.8). Lower regret is better.

4.5.1 Belief estimation

Each obstacle vehicle may or may not be intending to perform a lane-change maneuver. From the perspective of the ego agent, this is hidden state and must be estimated in order to perform a forward rollout.

For simplicity, we implement a stateless heuristic for belief estimation based on thresholds for the direction a vehicle is pointing, its position in its lane, and its velocity relative to the vehicle ahead of it. While this belief estimation leaves room for improvement, it should not affect the fairness of our method comparisons because we apply the same belief estimation for each method.

4.5.2 Methods

4.5.2.1 Multi-policy decision making (MPDM)

Our classic MPDM comparison takes samples (according to the computational budget) from the belief state and closed-loop forward simulates them through each of our five policy choices for 8

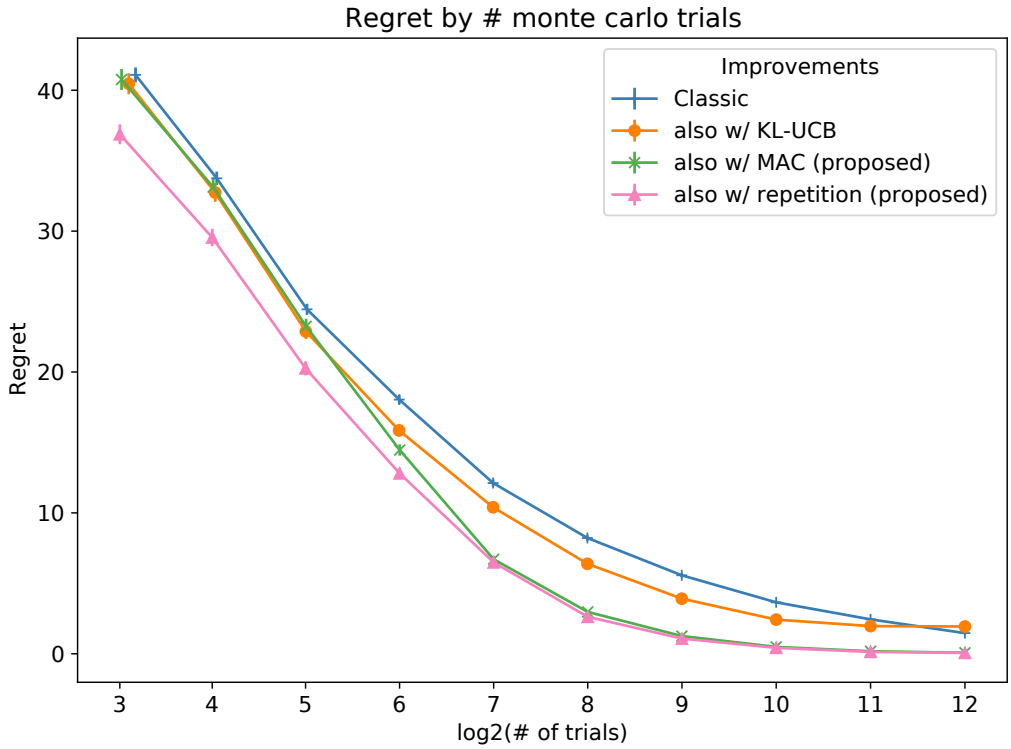

Figure 4.8: Ablation study of our method on the synthetic abstract scenario. We see the advantages of starting from traditional MCTS using UCB and "max-robust child", then adding KL-UCB, marginal action costs (MAC), and finally also particle repetition. Lower regret is better. Our full enhanced method performs better than all the ablative cases.

seconds. Finally, we elect the policy with the lowest mean cost.

4.5.2.2 Efficient uncertainty-aware decision making (EUDM)

Recall that EUDM is an extension to MPDM that includes both a specific tree search through the policies as well as a heuristic for selecting belief samples that represent the most important/risky cases.

The tree search used by EUDM, the "domain-specific closed-loop policy tree", allows for only one policy change in the planning horizon, and this change must happen below the root node of the tree. Just as in the original paper, we use a tree depth of 4 with each layer taking 2 seconds so that we have a total horizon of 8 seconds. As policy changes must happen after the root node, EUDM has a built-in hysteresis and will only actually change policies if it still wants to 2 seconds after first making that decision. The original authors use the current EUDM-selected policy as an input to a separate "spatio-temporal semantic corridor" trajectory generation module [97] which produces the actual behavior for the ego vehicle. This extra module allows their ego vehicle to react to changing circumstances in a risk-aware fashion even with the 2 seconds of policy hysteresis.

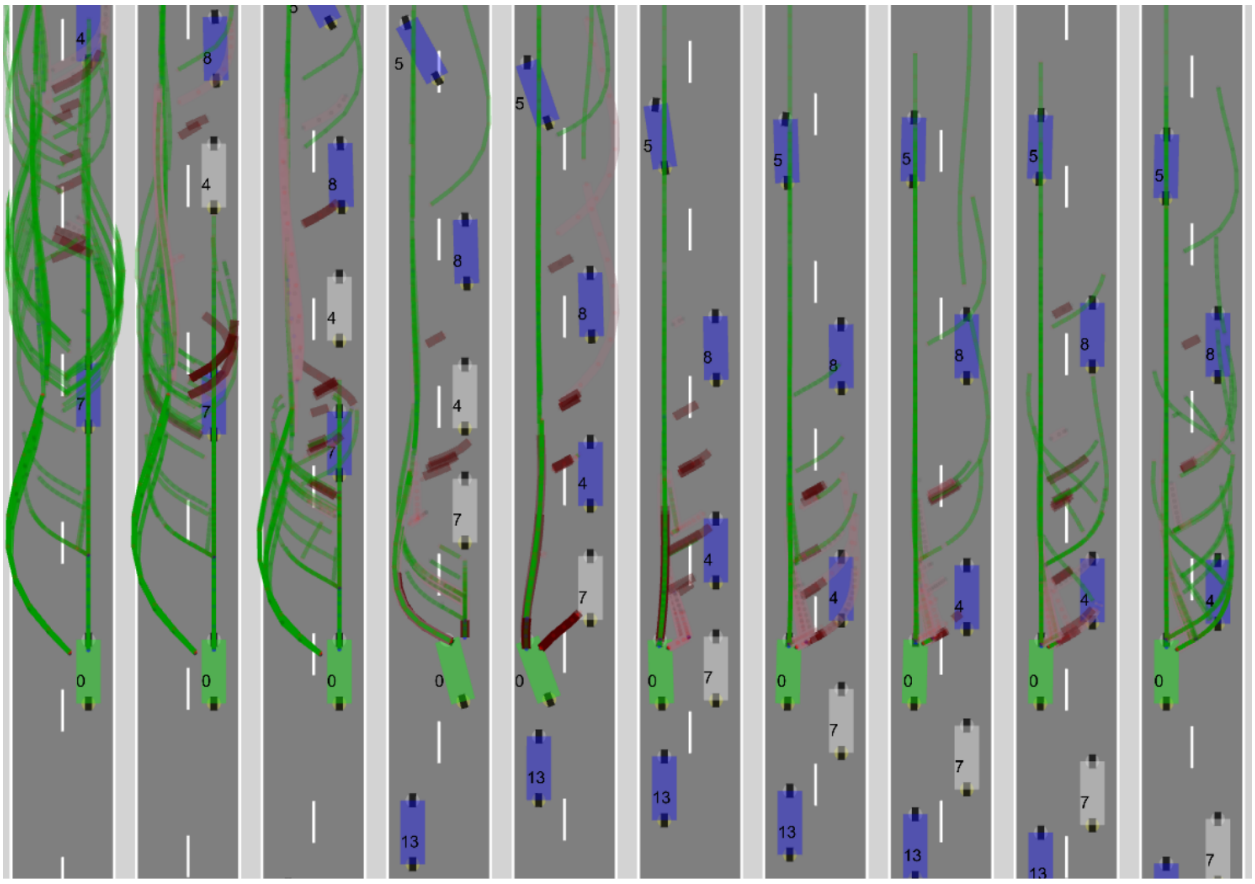

Figure 4.9: MCPTDM passing a vehicle and keeping distance from others in our simulated road environment. Vehicle 4 comes to a stop ahead of vehicle 7, causing it to stop in ahead of the ego vehicle (number 0). The ego vehicle moves into the left lane, passes vehicle 7, and then keeps a slight distance behind vehicle 4. We see how MCPTDM both performs tactical passing to make forward progress and also prefers to keep distance from vehicle 4, just in case other vehicles behave erratically. The ego vehicle is colored green, and obstacle vehicles are either blue while moving or gray while stationary. Monte Carlo trials are shown by their forward-simulated traces, which are dark red for traces leading to a crash, pink for traces that are somewhat unsafe, and green for safe traces. Frames are left-to-right in one-second increments. A video version of this sequence is provided at https://youtu.be/HYYTT3EYY1Q.

We modified EUDM to consider switching policies at any time, including immediately. This deviates from the original EUDM method, but we found it improves its performance in our comparison.

The heuristic used by EUDM, "conditional focused branching" (CFB), selects nearby obstacle vehicles, filters to just the vehicles whose policies we are uncertain about, then performs open-loop forward simulations of each belief policy, and finally makes a set of the most likely belief samples formed by the Cartesian product of vehicles and policies for each obstacle vehicle deemed risky

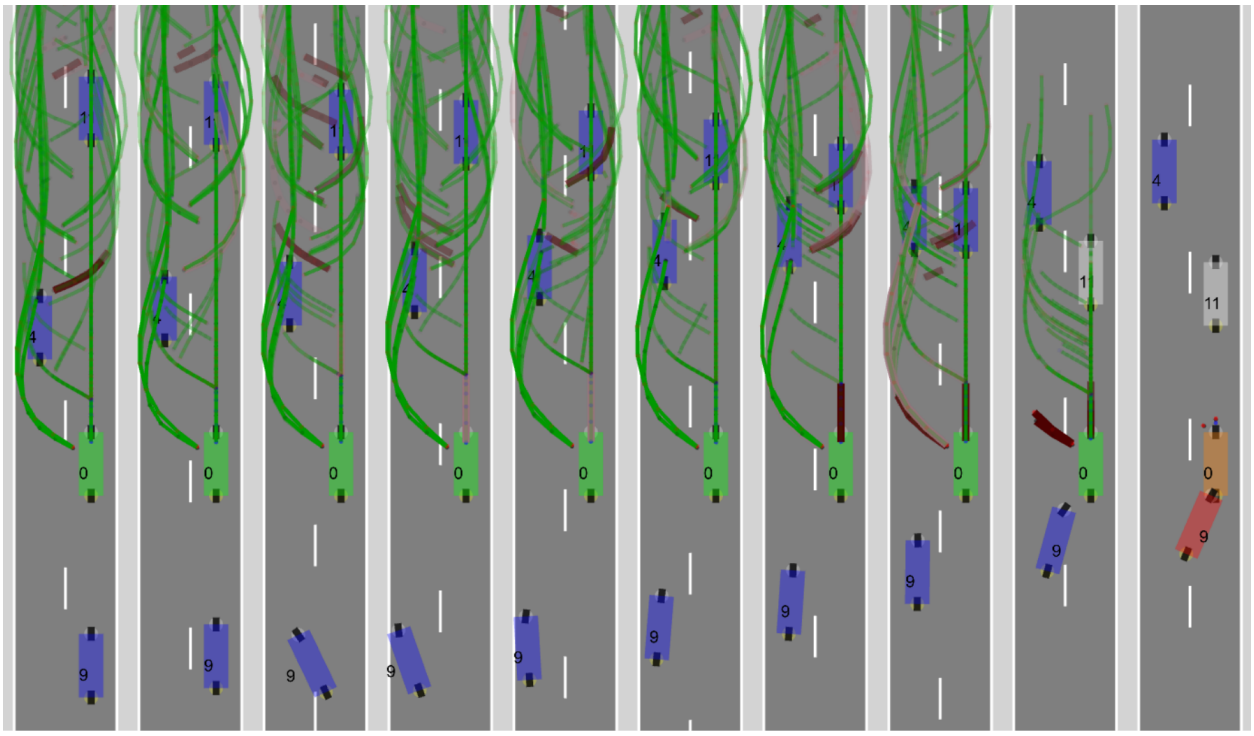

Figure 4.10: MCPTDM experiencing a crash in our simulated road environment (compare to Figure 4.9). As vehicle 11 comes to a stop in front of the ego vehicle, vehicle 9 from behind starts to make an unsafe lane-change into the right lane which the ego vehicle is unable to avoid. It is possible that the ego vehicle could avoid this crash if it were using a replanning rate of faster than 4 Hz. From the forward-simulated traces in the second-to-last frame, it appears that only scenarios with vehicle 11 accelerating first manage to avoid this crash, since the ego vehicle's intelligent driver model requires it to maintain a certain following distance. Frames are left-to-right in half-second increments. A video version of this sequence is provided at https://youtu.be/6vK-RxXwBGw.

by the open-loop simulations. All non-risky vehicles are assigned their most-likely policy.

In our implementation of EUDM, we perform open-loop forward simulations by giving only the ego and obstacle vehicle under examination dynamic policies, and simulate all the other vehicles with just a constant velocity. We use the same horizon of 8 seconds. We order obstacle vehicles according to their "risk", the difference between the minimum and maximum costs from each of the policy choices, and then we choose the 4 most risky vehicles. We form the Cartesian product of these risky obstacle vehicles and their policies and then finally select the most probable scenarios according to our belief, and weight them according to their probabilities. We take as many scenarios as allowed by our computational budget.

4.5.2.3 Monte Carlo Policy Tree Decision Making (MCPTDM)

Our final method for application to the automated driving scenario uses the improvements from the synthetic experiments described above: marginal action cost (MAC) expected-cost estimation and particle repetition. We use KL-UCB and "max-robust child" selection just as in the earlier experiments. In addition, when expanding a node in the search tree, we first explore the child with the same policy as the parent, since most of the time the ego vehicle will be maintaining its current policy.

4.5.3 Experiments

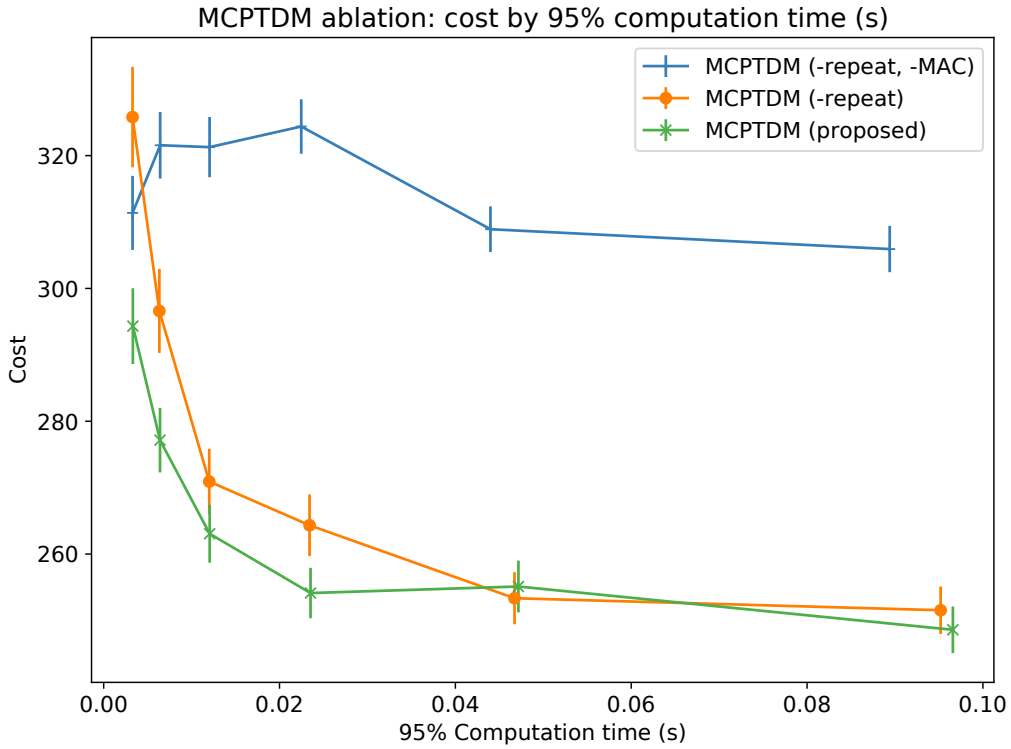

Figure 4.11: Ablation study of MCPTDM on the autonomous driving scenario. We evaluate it without particle repetition and then also with "classic" expected-cost estimation instead of marginal action costs. We see that both improvements are significant.

We perform 16,384 runs for each method in order to get significant results. To complete all these runs in a reasonable time frame, we limit each run to 30 simulated seconds and replan at 4 Hz. Runs are performed on a 2.5 GHz Intel Xeon E5-2640 with a single thread for each run so that multiple runs can be performed in parallel.

To make fair comparisons, we plot the final cost observed in each run (with no discount) against the 95% computational latency. That is, 95% of the replanning periods of a 30 second run will have

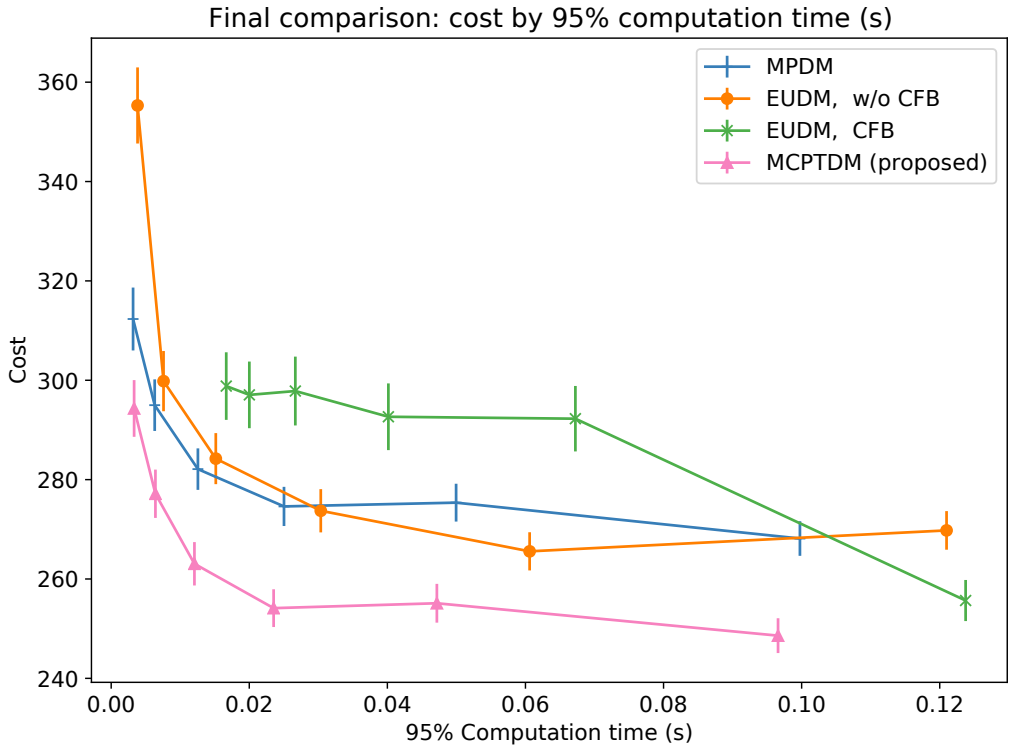

Figure 4.12: Final comparison of MCPTDM with EUDM (both with and without the CFB heuristic) and MPDM. MCPTDM achieves either significantly lower final cost or significantly lower computational time than either EUDM or MPDM.

a latency less than this. The closer a method is to the bottom left corner (closer to zero cost and zero time), the better.

Over a total of 589,824 30-second long runs to make all of these figures, there were a total of 2,573 crashes (0.44%). Because crashes are both relatively rare and also lead to much higher final costs, the error bars are relatively large in some of the figures.

First we perform an ablation to see the separate contributions of MACs and particle repetition. We performed a parameter sweep to determine reasonable constants for KL-UCB (for the full method) and for particle repetition, and found that it did best when repeating as much as possible. We find (see Figure 4.11) that both MACs and particle repetition result in significant improvements.

Finally we perform a comparison of our MCPTDM method to our baseline methods of MPDM and EUDM in Figure 4.12. MCPTDM achieves significantly lower cost for similar computational time. Our cost function is composed of a safety cost (for avoiding crashes and being too close), an efficiency cost (for being close to a target velocity), and steering and acceleration costs (for minimizing control inputs), where the safety and efficiency are the most significant. We compare just the safety and efficiency costs in figures 4.13 and 4.14, and note that the majority of the lower cost improvements of MCPTDM against MPDM and EUDM come from keeping efficiency

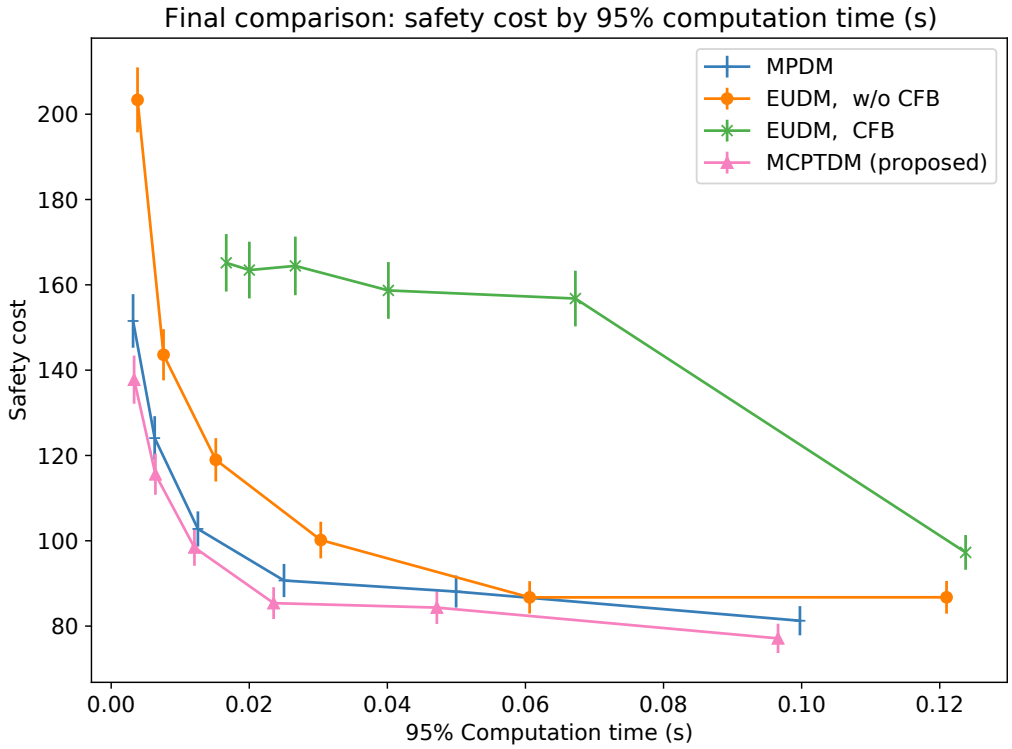

Figure 4.13: Comparison of just the final safety cost (lower is better) between each method. At all computation times, MPDM is only slightly less safe that MCPTDM. For larger computation times, EUDM is also very similar. Compare with Figure 4.14 and the plot of just the efficiency cost.

without sacrificing safety. Intuitively it makes sense that in most cases increased safety (lower safety cost) comes with worse efficiency (lower average velocity and a higher efficiency cost).

We notice that the EUDM's CFB heuristic is only an improvement in our self-driving scenario at the point of highest computational cost, even though it shares significant similarities with the scenarios used in the EUDM paper. This may be a consequence of omitting the separate trajectory optimizer from their paper, of implementing CFB in a slightly different way, or of another factor. In addition, we are surprised that vanilla MPDM is so competitive with EUDM because MPDM is not able to forward simulate policy changes like EUDM or MCPTDM.

4.6 Conclusion

In this chapter we have presented Monte-Carlo Policy-Tree Decision Making (MCPTDM), an MCTS-based search framework for problems where the marginal costs of each action or policy election are both available and important, such as autonomous vehicle planning. We have presented several novel ideas including the use of marginal action costs and fairness-based particle repetition. We have validated these improvements on both a simplified abstract task and also an

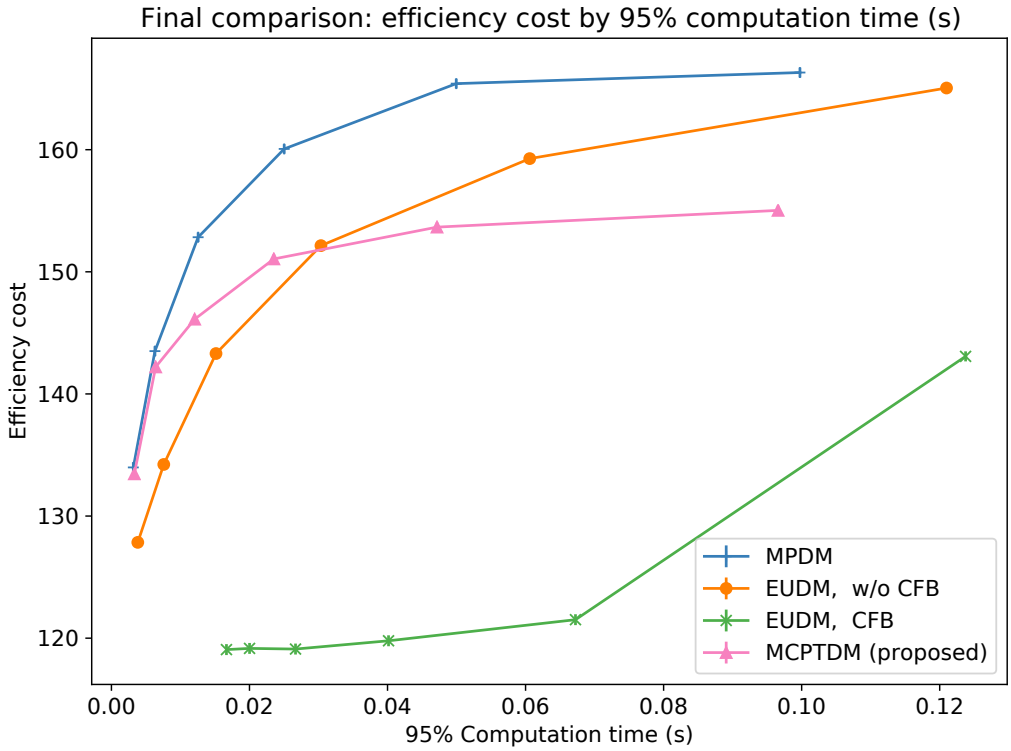

Figure 4.14: Comparison of just the final efficiency cost (lower is better) between each method. MCPTDM is quick to worsen efficiency (for better safety) and also keeps the efficiency cost relatively low as the computational budget increases. Compare with Figure 4.13 and the plot of just the safety cost.

autonomous vehicle planning task. We have demonstrated that MCPTDM has better performance than both MPDM and EUDM. The code used to generate all the results in this paper can be found at either `https://osf.io/pguhz/` or `https://github.com/acshi/MCPTDM` to aid in replicating and extending this work.

4.6.1 Acknowledgements

This work was supported by grants from the NSF (1830615).

Disclosure: Advisor Edwin Olson has a financial interest in a company that may have rights to foreground or background technology described in this chapter.

CHAPTER 5

Conclusion

In this thesis we have proposed novel estimation techniques for two key problems in long-term autonomy: how to localize and plan as uncertainty and error accumulate over time. Uncertainty is a fundamental difficulty in designing real-world robotic systems. As robotic systems need to run longer and longer, or in new and varied environments or conditions, uncertainty becomes an even bigger challenge.

While we do not claim in this thesis to have completely solved the problems of uncertainty in long-term autonomy, we hope to have shown that model assumptions have real costs in terms of how they handle more complex error distributions, and that non-parametric methods, both as a class of methods and a solution philosophy, have a significant advantage for handling real-world measurements and situations.

A model is most clearly appropriate when it is directly derived from the physical situation and is not making any limiting assumptions. In chapter 2, we model antenna delays, propagation time, time of flight, and distance together. As far as we are aware, we did not make any significant assumptions in doing this, other than fully-controllable assumptions, for example that we are in fact measuring propagation time. One issue that arose in that research is that the UWB radios we used have antennas with a preferred plane. Instead of ignoring how the error bias changes with the angle of incidence between antennas, we restricted the problem to 2D so that the antennas are always parallel. In order to leave this 2D context, however, without incorrectly assuming that angle of incidence has no effect on bias error, we would need to engage with a more complex physical model.

Sometimes models are available but do not need to be adopted fully to be useful. In chapter 3, we found that topological detectors for hallways and intersections could be used as masking functions to determine points of interest for a SLAM loop closing system. Importantly, since marking too many points of interest only results in a mild slowdown in performance, we do not need these topological class detectors to be at all perfect, greatly simplifying their design. After all, the benchmark comparison of dense metric SLAM considers *all* previous poses as loop closure candidates, so false positives are not a large concern.

In the cases of planning with uncertainty in chapters 4 and the appendix, general QMDP/POMDP planning prevents any obvious parametric model that can be used to simplify the handling of uncertainty and error, such as could be done with linear quadratic Gaussian (LQG) control. Instead, uncertainty is often represented by a collection of samples. Planners such as DESPOT [98] sample a set of determinized scenarios to consider and then work to find an optimal policy given those scenarios. We might consider this the most structured "model-like" approach, such that each possible policy always considers the exact same set of determinized scenarios. Monte Carlo tree search, goes even further from full guaranteed optimality and "model-like-ness" by discarding this constraint on equally considering every possible policy. In a real world problem with a huge action/policy space, it is not feasible in general to equally consider every possible policy. In MCPTDM, we also do not equally consider each scenario even for those policies that we do explore. In order to explore the most important parts of the belief and policy space, we have to give up the structure of equal and optimal exploration that would put us closer to DESPOT and LQG control. In the appendix too, with Learned Similarity Monte Carlo Planning (LSMCP), we propose that in order to infer a similarity relationship between state-action pairs, we need to use a non-parametric tool as flexible as a multilayer perceptron. Any parametric model for similarity would necessarily be too constrained to handle POMDPs of either different or greater complexity.

We have also shown specific contributions that make progress towards longer-term autonomy, enabling robotic systems to be more independent from human oversight and intervention. In chapter 2, we proposed a system for automatically finding the locations of a network of radios so they can then be used for localization. By having this as an automatic process instead of a manual calibration, a long-running autonomous system will be able to automatically recover from either intentional or accidental changes in the specific radio locations over time. In chapter 3, we proposed masked mapping for finding higher-quality loop closures with less computation. By masking off and ignoring lidar scans that are generic, non-distinctive, or more likely to change over time, a robot is more likely to maintain its localization both in the short and long terms. In chapter 4, we presented a planning framework based around combining and optimizing Multi-Policy Decision Making and Monte Carlo Tree Search. By joining both domain-specific policies and the more general foundation of efficient search, a robot can achieve more efficient planning, plan with longer time horizons, and make better decisions. All of these contributions work towards robots and autonomous systems that are more capable and less likely to need human intervention.

In total, we argue that non-parametric models and tools that make fewer assumptions are necessary to handle the accumulating error and uncertainty of real-world long-term autonomous systems. As robotic systems run for longer periods of time with less human supervision and around more dynamic agents, this is only going to be more and more important in the future.

5.1 Contributions

This dissertation included the following contributions:

- In chapter 2, we proposed an ultra-wideband-based (UWB) graph realization system to localize a set of UWB radios that achieves high accuracy through non-parametric estimation of measurement probability densities and explicit modeling of antenna delays. By using kernel density estimation and non-linear least-squares optimization, we took full advantage of the noisy non-Gaussian UWB measurements that might otherwise need to be identified and discarded to fit a parametric model. We also made four open-access datasets available for comparisons with our method, along with the complete source code for running our method and generating our figures.

- In chapter 3, we presented a flexible system for simultaneous localization and mapping (SLAM) that combines elements from traditional dense metric SLAM and from topological SLAM, using a masking function to focus attention. The abstraction of a masking function allowed us to take advantage of approximate/simplified topological class detectors to aid in finding loop closures without needing to worry about false positives or other accuracy concerns in the detectors. We presented two effective masking functions based on detecting openings and intersections that improve map quality while performing five times fewer scan matches than an "always true" mask would in dense metric SLAM. We demonstrated our system on a 2.5 kilometer dataset.

- In chapter 4, we described Monte-Carlo Policy-Tree Decision Making (MCPTDM), a framework for efficiently computing policies in partially-observable, stochastic, continuous problems. MCPTDM composes policies from a sequence of simpler policies and uses marginal action costs and particle repetition to improve cost estimates in a non-parametric error representation. We validated our improvements first on a simpler abstract problem and then compared our method to MPDM [17] and EUDM [18] on an autonomous self-driving task. We also provided openly-released source code capable of reproducing all the figures and evaluation results for full replication of our results and to enable further extensions to our work.

5.2 Future directions

For chapter 2, we would like to generalize our non-parametric network localization of UWB radios to 3D, expanding the non-parametric model in such a way as to include and account for the biasing effects of off-plane or non-parallel radio transmission angles. The biggest challenge to this is

that with introducing additional degrees of freedom into the optimization problem, it may be even harder to find a good solution with only a limited number of stationary radios. We will likely need to include dynamic movement of at least one radio in the data-gathering process in order to build the full non-parametric model of radio positions and parameters to optimize.

For chapter 3, we would like to examine the effectiveness of other masking functions that are more directly based on topological classifiers. The two masking functions we proposed and examined where both much more loosely inspired by topological mapping. By taking masking functions as directly as possible from the topological mapping literature, we can then also directly compare SLAM performance between our system and a topological mapping system, and this should make the trade-offs between these options more clear. We expect that our system, because it does not actually apply topological classes to regions of the map, would have better localization performance for the same computational cost, even when using the exact same classifiers. We would also expect that there may be straightforward ways to simplify those classifiers and reduce their computational cost without reducing the effectiveness of our system, where these same simplifications would impair the performance of the topological SLAM systems.

For chapter 4, we have already explored one future direction in the appendix, where we moved from a self-driving scenario to any general continuous-space POMDP. Alternatively, we would also like to continue examining the more specific space of QMDPs like the self-driving scenario. First, we would like to find a different QMDP problem domain to examine. Second, we would like to make the self-driving problem scenario and simulation environment more complex and realistic, potentially using industry-standard simulation tools, in order to see how MCPTDM scales with that problem complexity. We expect that MCPTDM will still have the same relative performance advantages, but this will also enable us to make direct comparisons with other self-driving planners that take entirely different approaches.

APPENDIX A

Learned Similarity Monte Carlo Planning (LSMCP)

A.1 Introduction

In this appendix, we describe Learned Similarity Monte Carlo Planning (LSMCP), a line of research that has had only mixed results so far, but that seeks to enhance the sample efficiency of partially observable Monte Carlo tree search-based planning by learning about and taking advantage of similarities (correlations) in the final outcomes of similar states and actions. In line with our principal of flexible non-parametric modeling, we train a multilayer perceptron to learn a similarity function for each problem domain and we apply these learned similarity enhancements to a planning method already based on random sampling.

An important limitation of our earlier work on MCPTDM is that it has no motivation to select actions whose primary purpose is to gather information. Some problems, including the self-driving scenario described in the last chapter, are well-suited to this subclass of POMDPs (partially observable Markov-decision processes) known as QMPDs (Q-value Markov decision processes) [15]. We are interested here in finding algorithmic improvements to the more general domain of continuous-space POMDP planning, which has the additional challenges of both continuous action and observation spaces that we managed to avoid in our work on MCPTDM.

The more demanding the planning problem, the more important it is for a Monte Carlo-based planning method to emphasize sample-efficiency. We did this in MCPTDM with 1) carefully-guided particle repeating to reduce the variance between separate action/policy choices and with 2) final selection using marginal-action costs to appropriately take into account the cost structure of the target problem domains. Considering our new goal of general continuous-space POMDP planning, point 1 is less relevant when we have a continuous action space and point 2 is highly dependent on choice of problem domain.

While researching MCPTDM we noticed that different action choices sometimes led to the same result. For example, environmental constraints such as walls, vehicles, or other obstacles can effectively reduce the total number of distinct action or policy choices. Even if the outcomes are not

identical, environmental situations around a decision point, can similarly lead to a smaller effective action space. For example, the actions of a robot facing a bifurcating path can be effectively grouped into two sets: the actions that result in taking the left path, and those that take the right path. While heuristics can be written to reduce the action space when situations like these are detected, our goal is to take advantage of this pattern, wherever it exists, for POMDPs in general. We also want to take advantage of correlations between actions even when that correlation is only partial. For example, we might have a compound action consisting of a motion part (subject to the bifurcating path-style simplifications), a manipulation part (picking up or placing target objects), and a sensing part (detecting target objects). While we would expect similarities between the same parts of the compound action, we do not expect similarity across the different parts of the compound action.

As we perform our Monte Carlo-based search, our goal is to use some generalized understanding of correlation/similarity to share and combine high-variance rollout outcomes from different parts of the search tree into lower-variance estimates that converge faster. For example, if rollout outcomes can be sorted into either "took the left path" and "took the right path", we can calculate the expected values for each outcome, getting a lower-variance estimate because we have pooled more samples together, and then assign these estimates back to the individual nodes of the tree they came from. Because we want to target general black-box POMDPs of arbitrary complexity, we will attempt to learn a representation of this similarity automatically from training data acquired by running instances of a given POMDP through non-augmented planning.

We start by building on top of a POMCPOW (partially observable Monte Carlo planning with observation widening), a planning framework that is capable of continuous-space POMDP planning [15]. In order to solve fully continuous POMDPs, POMCPOW uses progressive widening [91][99], extending the idea of double progressive widening [100] from MDPs to POMDPs. POMCPOW uses progressive widening for both the action and the observation spaces, using an explicit observation function to handle the fact that each observation may have a different probability. We give a more detailed overview of POMCPOW in the background section below.

A.2 Background

In this section, we review partially observable Monte Carlo planning with observation widening (POMCPOW), following on the description of Monte Carlo tree search given in the background section of chapter 4.

A.2.1 POMCPOW

Many state-of-the-art partially observable Markov decision process (POMDP) solvers use sampling techniques in order to make handling uncertainty tractable. We first review some of these works, and then we give greater detail on POMCPOW itself.

With the goal of finding an optimal policy, the determinized sparse partially observable tree (DESPOT) [98] algorithm considers K uniformly-sampled "determinized scenarios" that factor out all the possible uncertainty in the problem. DESPOT constructs a tree by examining every possible action sequence for each of those K samples and computes an optimal policy given that sample/scenario set. While DESPOT is primarily an offline method, it can also be run in an online fashion with branch-and-bound techniques. Overall, DESPOT is capable of handling continuous state spaces but only discrete action and observation spaces. Extensions to DESPOT include the incorporation of importance sampling [101] and using alpha vectors to share information [102]. Alpha vectors are also used by the approximately-optimal offline fully-discrete SARSOP method [103].

Setting aside optimality for an online-first approach, partially observable Monte Carlo planning (POMCP) [61] uses Monte Carlo tree search to approximately solve a POMDP given a black-box simulator and an unweighted particle filter. Similar to DESPOT, POMCP can only handle discrete POMDPs or at most continuous state spaces.

The key innovation of POMCPOW (partially observable Monte Carlo planning with observation widening) is the requirement to specify an explicit observation function $Z(o|s, a, s')$ instead of just an implicit observation function for sampling $o \sim Z(o|s, a, s')$. This enables POMCPOW to use progressive widening for observations, reusing observations when going down the search tree according to their likelihood. This is important because in a truly continuous POMDP, observation samples will always be unique.

In progressive widening [100], the number of choices (child nodes in the tree) is artificially limited to kN^α, where k sets the initial number of choices, N is the number of visits to this decision point in the tree search, and exponent $\alpha \in [0, 1]$ controls the rate of progressive widening. While for action progressive widening the action space is merely limited, next-state (in stochastic-transition MDPs) or observation (in POMDPs) progressive widening requires an approximation to be made because when the child nodes are at capacity, the unbiased next-state or observation sampled from the black-box simulator must be discarded and replaced with a next-state or observation from one of the existing limited child nodes.

Specifically, when observation progressive widening comes into play in POMCPOW, we will have already run our previous state s and selected action a through the black-box POMDP and generated the resulting next state s', observation o, and reward r. When progressive widening dictates that we repeat a previous observation instead of the just-sampled o, we have to choose an already-existing observation node. We make a random choice weighted by $Z(o|s, a, s')$, for each

already-existing observation o. This helps limit the bias introduced by not being able to use the original unbiased observation sample.

POMCPOW maintains a "root belief" about the unknown state of the POMDP. While this is typically a particle filter, the only requirement for the root belief is that it affords sampling. For each Monte Carlo trial, this root belief is sampled, and that specific state is then used for the trial. The tree itself is composed of *histories*, where each node represents some sequence of actions and observations from the root. Each Monte Carlo trial may add at most one new observation node to the tree, after which a rollout policy (e.g. random actions) is run as a black box procedure. The belief at any non-root node is implicitly represented by the collection of samples/trajectories that have passed through that node. POMCPOW uses progressive widening for both actions and observations. There is an initial limit to the number of action and observation children for any node to have, and this limit is filled up by the actions and observations that happen to occur first. As the visit count of a parent node increases, there is a schedule to progressively increase the limits on action and observation children.

Each action node maintains an estimate of its value (expected discounted reward if that node where the root). This value is the mean discounted reward calculated from the sample trajectories that pass through the node. Our goal in this work is to aggregate similar samples and states from across the search tree in order to refine these values, improving on the performance of POMCPOW.

In Algorithm 6, we show POMCPOW and encourage comparison with the description of MCTS in Algorithm 4 given in the background section of chapter 4. As MCTS is formulated to solve MDPs, each node directly represents a state. In POMCPOW, each node presents a history of actions and observations from the root node, and so specific states are more transient, sampled from the belief at the beginning of each trial and passed through the tree alongside the specific nodes. In MCTS we branch on actions, while in POMCPOW we branch on both actions and observations, but still only maintain statistics for the action nodes that we need to choose between. In addition, POMCPOW uses progressive widening because it specifically targets problems with continuous action and observation spaces, while this is generally just an option in MCTS since it is often used for discrete deterministic problems.

Algorithm 6: POMCPOW

For node n, $N(n)$ is the visit count, $V(n)$ is the value sum, and $Q(n)$ is the expected value. Node n'' is child of n' which is child of n. In function recurse, n and n'' are observation nodes and n' is an action node. Subroutines are specified in Algorithm 7.

function CHOOSEACTION(b)
- $n \leftarrow$ CREATENODE()
- **while** time remains **do**
  - // b could be a particle filter, for example
  - $s \leftarrow$ DRAWBELIEFSAMPLE(b)
  - RECURSE(n, s)
- **end**
- **return** action of best/highest-Q child of n

function RECURSE(n, s)
- **if** state of n terminal **then**
  - **return** 0
- **end**
- EXPAND(n)
- $n' \leftarrow \text{UCB}(n)$; $a \leftarrow$ action of n'
- // k_o and exponent α_o control observation progressive widening
- **if** n' has $\leq k_o N(n')^{\alpha_o}$ children **then**
  - // add new observation child node and rollout
  - $s', o, r \leftarrow$ FORWARDSIMULATE(n')
  - $n'' \leftarrow$ add new child of n' with observation o
  - add belief sample (s', r) to n'' with weight $Z(o|s, a, s')$
  - $v' \leftarrow$ ROLLOUT(s')
- **else**
  - // reuse existing observation child node and recurse deeper
  - $s', r \leftarrow$ FORWARDSIMULATE(n') // Ignoring o
  - $n'' \leftarrow$ random child of n'
  - $o \leftarrow$ observation of n''
  - add belief sample (s', r) to n'' with weight $Z(o|s, a, s')$
  - $s', r \leftarrow$ weighted sample from belief of n''
  - $v' \leftarrow$ RECURSE(n'', s')
- **end**
- $v \leftarrow r + \gamma v'$ // discount factor γ
- UPDATESTATISTICS(n', v)
- **return** v

Algorithm 7: POMCPOW subroutines

function EXPAND(n)
  // k_a and exponent α_a control action progressive widening
1 **if** n has $\leq k_a N(n)^{\alpha_a}$ children **then**
2 add new child of n with action NEXTACTION(n)
  end
function UCB(n)
3 **if** n has any unexplored children **then**
4 **return** a random unexplored child of n
  end
  // UCB constant C controls exploration vs. exploitation
5 **return** $\arg\max_{n'} Q(n') + C\sqrt{\frac{\ln N(n)}{N(n')}}$
function UPDATESTATISTICS(n, v)
6 $\mathrm{N}(n) \leftarrow \mathrm{N}(n) + 1$
7 $\mathrm{V}(n) \leftarrow \mathrm{V}(n) + v$
8 $\mathrm{Q}(n) \leftarrow \mathrm{V}(n)/\ \mathrm{N}(n)$

A.3 Other Related Work

The goal of this work is to increase sample efficiency in POMCPOW or similar MCTS-based search algorithms by reusing information across "similar" parts of the search tree, based on different actions and/or states being similar to each other. In this related work, we examine both other forms of information sharing as well as approaches that seek to more intelligently select the actions to explore in the first place.

One of the earliest techniques is the Move-Average Sampling Technique (MAST) [104][105], which keeps track of statistics for each action independent of the state, and was first used with Gibbs sampling to make action choices in the rollout (default-policy) phase. Later work showed improved performance with ϵ-greedy selection instead of Gibbs sampling [106]. MAST keeps track of these statistics across an entire game, and not just a single planning episode, and so strategies for decaying older MAST statistics have also been explored with significant improvements in the majority of cases [107].

Variations on MAST also include those that store statistics based on predicate-action pairs (in games whose state is a collection of predicates) and feature-action pairs (with designed features), or that only store statistics for actions encountered in the tree portion of the search, excluding values from actions selected in the rollout phase[105]. MAST has also been extended to sequences of actions with the N-gram selection technique (NST) [106]. One limitation of these MAST-based techniques is that they are not directly applicable to problem domains with continuous action spaces.

Rapid Action Value Estimation (RAVE) [108] is a complementary technique that operates in the selection (e.g. UCB) phase instead of the rollout phase. In RAVE, an "all moves as first" heuristic operates on the assumption that if a move will be useful later, it is also likely useful now, and so the statistics of all actions below the current search node are used to inform the selection of actions at that node. In this way, each trial/rollout provides some information about many actions instead of just one. RAVE does this by replacing Q with Q_{RAVE}, where $\tilde{Q}(s,a)$ is the mean value of all nodes with action a that are at or below the state s and where k is an equivalence parameter, setting the number of trials at which Q and Q_{AMAF} should be given equal weight:

$$Q_{RAVE}(s,a) = \beta\tilde{Q}(s,a) + (1-\beta)Q(s,a) \tag{A.1}$$

$$\beta = \sqrt{\frac{k}{3N(s,a)+k}} \tag{A.2}$$

RAVE has also been generalized to GRAVE [109] to consider not just any actions below the current node, but potentially all the actions below some parent node. When the statistics for the current node have high variance, GRAVE tries considering the RAVE statistics of the parent node,

and will continue considering further parent nodes as long as the variance is too high. A study comparing RAVE, GRAVE, and an extreme version of GRAVE that always uses the root-level statistics, called History RAVE (HRAVE), showed only small differences overall with significantly more variance on a per-game basis [110]. This study also showed the MAST and RAVE compose together well.

RAVE has been generalized to continuous spaces in cRAVE [111], where instead of only selecting values from nodes that exactly match the action, each candidate node/action is weighted according to an exponential that decays with a domain-specific distance function. In addition to incorporating this exponential decay for distance between actions, that decay can also incorporate distance between states. This work on cRAVE is very similar to the work we are trying to accomplish. The main difference is that we want to learn this distance metric instead of requiring the user to supply it. We also want to relax the "all moves as first" requirement and explore the entire tree for similar states and actions. Finally, we want to evaluate against POMDP problem domains, while the cRAVE work was evaluated against treasure hunt and energy management MDPs.

Voronoi Progressive Widening (VPW) [112] provides a specific problem-independent strategy for choosing new actions. In addition to sampling random actions, VPW only requires a distance metric on the action space. With some probability, new actions are sampled from either the total action space or from the Voronoi cell with the best value. As a specific implementation of action progressive widening, VPW in POMCPOW is called VOMCPOW, and demonstrated significantly greater performance on all of the POMDP problem domains they examined. They also showed that a specific form of VPW has global convergence guarantees for continuous POMDPs.

Bayesian Optimized Monte Carlo Planning (BOMCP) [113] also provides a problem-independent progressive widening strategy for actions. Instead of using Voronoi cells and a distance metric to decide on new actions, BOMCP uses a Gaussian process to model the action-value landscape and select the action that will maximize the expected improvement in that Gaussian process model. Unlike VPW, this requires some additional smoothness properties in the action space in order to properly construct the Gaussian process models.

These methods based on action progressive widening are essentially incorporating similarity information into the selection of which actions should be explored next. Since this information is used only at the point of the decision to add a new action node, we aim for our work to be complementary to these methods. It is also possible that if may learn a similarity function that can double as a distance function that could be used with Voronoi progressive widening as well.

A.4 Problem Statement

We consider a continuous POMDP with a generative model for the transition function and an explicit distribution for the observation function. This is the same problem class as considered by POMCPOW. In particular, we focus on problems with relatively more expensive transition functions, where the cost of the transition function is expected to be much greater than any overhead in our LSMCP planning framework.

We consider the Monte Carlo search tree that is formed in the planning process of an MCTS-based POMDP solver such as POMCPOW. The search tree branches on both continuous actions and continuous observations, and so a large number of Monte Carlo trials are necessary in order to have sufficient confidence in the value of the deeper nodes required for future planning. However, in many problems, it is extremely likely for natural similarities to arise between different parts of the search tree. If we can exploit the information present in these similarities, we can arrive at more accurate belief-action Q-values and more closely approximate the optimal policy with fewer Monte Carlo trials.

POMCPOW represents belief with a collection of state samples. We want to find a "similarity" or "equivalence weight" function that tells us how much the rollout (default policy) value of a sample state-action pair from the belief in one part of the tree can inform the value of the belief in another part of the tree. We want to find a function that maps from the state-action-rollout value tuple of a "reference" belief sample and the state-action tuple of a "query" sample to give an equivalence weight between 0 and 1:

$$\text{Sml} : S \times A \times S \times A \to \{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$$

At the extremes, a weight of 1 indicates the reference-sample rollout value is just as informative as a true query-sample rollout value and a weight of 0 indicates that the reference value is completely uninformative.

In POMCPOW, the value of an action node is the mean of $r + \gamma v'$ (see line 22 of Algorithm 6) from each time the node is visited during a Monte Carlo trial/iteration. The first time the node is visited, v' comes from a random rollout procedure. Each subsequent time the node is visited, v' comes from a call to RECURSE using a reward and next state that have been sampled from a randomly chosen observation node because of the limitations of observation progressive widening. Because of this additional complexity involved in observation widening, we focus on using similarity to better estimate the true expected value of pure rollouts.

When a rollout occurs, it runs from a single specific state. However, each action node in the tree is best represented by an action and a belief. That belief will eventually be composed of multiple states, and this collection of states will better represent the true belief of the action

node and its history as the action node is visited more and more times and the number of states increases. POMCPOW does not need to keep track of all these states, so we will need to track that ourselves to support similarity calculations. Using our similarity function Sml, we should be able to estimate the expected rollout value of a state-action pair. For an action node with a belief containing multiple states, we should then be able to combine individual state-action pair estimates into an overall estimate $\tilde{Q}$ for the whole belief of the action node.

We estimate the expected rollout value of a query action node with action a and belief b using a set of reference state-action-rollout value tuples ref, with ith reference having action a_i^{ref}, state s_i^{ref}, and rollout value v_i^{ref}. We make a separate estimate for each query state and then take the mean for the overall estimate $\tilde{Q}$ for the belief:

$$\tilde{V}(s,a) = \sum_{i \in \text{ref}} v_i^{\text{ref}} \, \mathrm{Sml}(s, a, s_i^{\text{ref}}, a_i^{\text{ref}}) \tag{A.3}$$

$$\tilde{N}(s,a) = \sum_{i \in \text{ref}} \mathrm{Sml}(s, a, s_i^{\text{ref}}, a_i^{\text{ref}}) \tag{A.4}$$

$$\tilde{Q}(s,a) = \tilde{V}(s,a) / \tilde{N}(s,a) \tag{A.5}$$

$$\tilde{Q}(b,a) = \text{mean value of } \tilde{Q}(s,a) \; \forall s \in b \tag{A.6}$$

Once we have these action node rollout value estimates $\tilde{Q}$, we will have to figure out how to use them most effectively along-side the "real" values Q obtained through regular POMCPOW to form augmented values Q'. How much weight do we give these similarity-based values? Since we are estimating rollout value, should we only use these estimates for leaf nodes? If the estimates are biased, how do we keep the bias from degrading performance?

Finally, to be useful in practice, the similarity function should have low-enough computational cost such that computing estimated values is more effective than using the same time to just run additional Monte Carlo trials.

A.5 Learning similarity

In order to support arbitrary POMDPs, LSMCP first needs to learn a multilayer perceptron (MLP) we can use to compute similarity.

However, in order to calculate $\tilde{Q}$ for every node in the search tree, we have to compute the similarity between almost every state-action sample in the search tree. The number of similarity computations would be of order $O(NM)$ where N is the number of state-action sample queries anywhere in the tree and M is the number of state-action-rollout reference samples. We note that N is strictly greater than the number of Monte Carlo trials, because each Monte Carlo trial will create a number of additional state-action samples equal to the number actions taken before the

rollout portion of the trial, while M is the total number of Monte Carlo trials, since each trial iteration ends in a rollout.

We address this potential complexity by learning the similarity function indirectly. For individual state-action samples, we learn a transformation $\mathrm{Xform} : S \times A \to \mathbb{R}^D$ into a dimension D Euclidean space "position". We then map distance into similarity:

$$d = ||\mathrm{Xform}(s_1, a_1) - \mathrm{Xform}(s_2, a_2)||_2 \tag{A.7}$$

$$\mathrm{Sml}(s_1, a_1, s_2, a_2) = e^{-d} \tag{A.8}$$

Computing similarity in this way reduces the number of MLP inferences down to just $O(N)$ and reduces the $O(NM)$ part to just computing an L2 norm and exponential. During inference time, we further reduce the number of $O(NM)$ operations by using a linear lower bound on the exponential which is much faster to compute:

$$\mathrm{Sml}(s_1, a_1, s_2, a_2) = \max(0, 1 - d) \leq e^{-d} \tag{A.9}$$

Using distance to compute similarity also opens up the possibility of using an approximate nearest neighbors (ANN) algorithm to entirely eliminate the $O(NM)$ bottleneck for problem instances with larger N.

A.5.1 Loss function

The optimal behavior for our system would have the estimates $\tilde{Q}$ directly lead to the best action being chosen by our method, LSMCP. If the best action is not chosen, then we want to minimize the regret (true value of the best action - true value of the action taken). Unfortunately, regret itself is not differentiable, since it is based on the true values of actions, and not on the estimated values. Even then, we have no way to access the absolute true values for an arbitrary POMDP. However, we can approximate the true rollout values of actions by simply running as many rollouts as needed for the mean value to converge (see training data section below). We can then use the regret as a weight on other differentiable losses that are based on the estimated values of each action choice being compared. Using regret as a weight like this encourages the learning to minimize regret even though it is not differentiable.

Since we are trying to learn a similarity function that is a function of states and not directly of beliefs, we also only consider rollout values and regret in the context of a single state. So our losses are a function of the "true" values for a state and each of its actions and the similarity-estimated values of those same actions.

Our loss function encourages learning the correct values, the correct advantages (value - mean

value of all actions), and a bias of zero, where N_a is the number of action choices:

$$\begin{aligned}\text{loss} = \text{regret} * \big[& w_1 \frac{1}{N_a} \sum_a (\tilde{v}_a - v_a)^2 + \\ & w_2 \frac{1}{N_a} \sum_a ((\tilde{v}_a - \bar{\tilde{v}}) - (v_a - \bar{v}))^2 + \\ & w_3 (\frac{1}{N_a} \sum_a \tilde{v}_a - v_a)^2 \big] \end{aligned} \tag{A.10}$$

We chose these quantities because they are simple functions of the available parameters. While either the direct value loss or the advantage loss could alone lead to zero regret, the advantage loss more directly captures the comparative nature of how the best action is chosen and the direct value loss provides a specific target to reach. The bias loss is intended to make it easier for LSMCP to actually incorporate the estimated $\tilde{Q}$ values.

We also note that while we cannot directly train on the regret, it is a useful signal to monitor training effectiveness and progress.

A.5.2 Training data

In order to support the loss function detailed above, we need to some way of finding "ground truth" expected rollout values for state-action pairs for arbitrary POMDPs. We do this by continuously running rollouts from each state-action until their mean value effectively converges. To make the computational time predictable, we first determine a fixed number of rollouts to use by running examining a small set of state-action pairs and examining the proportion of those that converge to a certain degree. For example, we found that for one POMDP, 512 rollouts lead to 91% of state-action pairs converging to within 2.5% of the value for twice as many rollouts, and 64% converging to within 1.25%.

We then collect training data by running the POMDP with settings that will produce the form of data we need for our loss function. We take only a single sample from the root belief and use this specific state for all actions and rollouts. We do not perform tree search, and instead only perform rollouts for each action. We also use a constant and high number of actions, disabling progressive widening. After running all these rollouts, we extract and save the training data, execute the chosen action, update the root particle belief, and then repeat until the POMDP has terminated. We continue this whole process until enough training data has been acquired. The training data extracted includes states and sets of actions for each state, and corresponding expected rollout values for each of those actions.

After training data is collected, we follow standard data preprocessing practice and normalize the data so that each value has a standard deviation of 1 and a mean of 0.

A.5.3 Batches

So far we have training data consisting of states each with a set of action-value tuples. In order to evaluate our loss function, we need to form similarity-based estimates for those rollout values and decide on the set of reference state-action-rollout value tuples that will be used to form the estimates as described in Eqs. A.3-A.6.

For each training sample, we randomly chose a query state from the training data with its accompanying set of actions and "ground truth" values. We then also chose a random set of reference state-action-rollout value tuples. The number of reference tuples we use is a parameter we call *references per sample*. For more training flexibility, we can also only use a subset of the available query state actions by setting a lower *actions per sample*. Finally, the *batch size* is then the total number of training samples constructed this way for each optimization step.

Because each training sample is constructed from a different random combination of query and reference samples, there is an exponentially large number of possible unique batches we can create. In practice, we have found that there is little benefit to creating more batches than there are unique query states.

A.5.4 Training

We trained all networks using Pytorch 1.10.2 using the AdamW optimizer, a reduce-learning-rate-on-plateau scheduler, gradient clipping, and early stopping. The input to the network is a state-action pair and the output is a D-dimensional vector in a Euclidean space. We use an architecture with fully connected linear layers (no bias) each followed by batch normalization and a leaky ReLU. The final output layer also does not have bias, normalization, or an activation function. These configuration options were chosen to conform with our best understanding of the current best practices, except for our use of a learning rate scheduler. AdamW (and the original Adam variation) is generally considered to not need a scheduler because it maintains a separate adaptive learning rate for each parameter. However in our training we found the scheduler critical to lowering the overall loss. Without it, loss would plateau at far higher values. We suspect this behavior may be related to the relatively high noise and variance in our training data.

A.6 Using similarity

Ideally, if the similarity function truly learns to provide a rollout value equivalence, then constructing augmented values Q' could be done directly with:

$$V' = V + \tilde{V} \tag{A.11}$$

$$N' = N + \tilde{N} \tag{A.12}$$

$$Q' = V'/N' \tag{A.13}$$

However, if there is any bias in $\tilde{Q}$ then this bias will have an unequal effect across the tree, with more bias injected wherever $\tilde{N}$ is larger because a state-action tuple had more total similarity. Unequal bias will lead to bad Q' values even if the $\tilde{Q}$ on their own correctly identify the best action for every state.

There are several ways that we can try to control this bias. One option is to borrow the weighting scheme from RAVE, which ensures that as trials go to infinity, the weight on the biased estimate goes to zero:

$$Q'(n) = \beta \tilde{Q}(n) + (1 - \beta) Q(n) \tag{A.14}$$

$$\beta = \sqrt{\frac{k}{3N(n) + k}} \tag{A.15}$$

A limitation of this approach is that it directly uses the similarity-estimated $\tilde{Q}$ and therefor does not take into account the weight $\tilde{N}$, which we expect should be smaller for less confident estimates and greater for more confident ones.

Another idea that tries to draw meaning from the $\tilde{N}$ quantities, is to artificially limit the weight values in Eqs. A.3 and A.4 such that the total weight for each query state between action choices is equal and then to add the separate value and weight sums to get totals for the belief. We also

divide the total weight by the number of states in the belief $||b||$:

$$\tilde{V}(s,a) = \sum_{i \in \text{ref}} v_i^{\text{ref}} \, \text{Sml}(s, a, s_i^{\text{ref}}, a_i^{\text{ref}}) \tag{A.16}$$

$$\tilde{N}(s,a) = \sum_{i \in \text{ref}} \text{Sml}(s, a, s_i^{\text{ref}}, a_i^{\text{ref}}) \tag{A.17}$$

$$\tilde{Q}(s,a) = \tilde{V}(s,a) / \tilde{N}(s,a) \tag{A.18}$$

$$\text{LimitN}(s) = \min_{a \in \text{node}} \tilde{N}(s,a) \tag{A.19}$$

$$\tilde{V}(b,a) = ||b||^{-1} \sum_{s \in b} \text{LimitN}(s) \tilde{Q}(s,a) \tag{A.20}$$

$$\tilde{N}(b,a) = ||b||^{-1} \sum_{s \in b} \text{LimitN}(s) \tag{A.21}$$

Making a comparison between these two approaches is left for future work.

A.7 Evaluation

In this section we describe two simple POMDP problem domains that we use for evaluating our method and our preliminary results against them. We designed the first problem domain to be as advantageous as possible to the idea of learned similarity and LSMCP. The second problem domain comes from the original description if our comparison method POMCPOW [15].

One important baseline we used was training the same size multilayer perceptron to directly estimate $Q(s,a)$ values, using the same loss function. In order for learned similarity to pose a value-gain over something like deep Q-learning function approximation, then we need to find that LSMCP is more expressive and generalizable for the same size learned network. We will refer to this baseline as the value network, as opposed to the full LSMCP where the network maps state-action tuples into a space where distance determines similarity.

In our evaluations below, we incorporate similarity values as described in Eqs. A.11-A.13 and A.16-A.21 for all action nodes in the entire tree as needed for computing UCB values and choosing the final best action. We do not report execution time in the analysis below because it was generally not competitive with POMCPOW. Both the baseline value network and LSMCP were between approximately 2 and 100 times slower, depending on the parameters.

A.7.1 Buffet

For the initial development of our approach, we designed a continuous POMDP that would be maximally exploitable by information aggregation. In the buffet POMDP, a "diner" tries to explore

their way to the best "food" available. The diner always knows their position, but the only way to find the best food is to move around and try the different foods. The diner has to make an exploration-exploitation tradeoff to maximize reward.

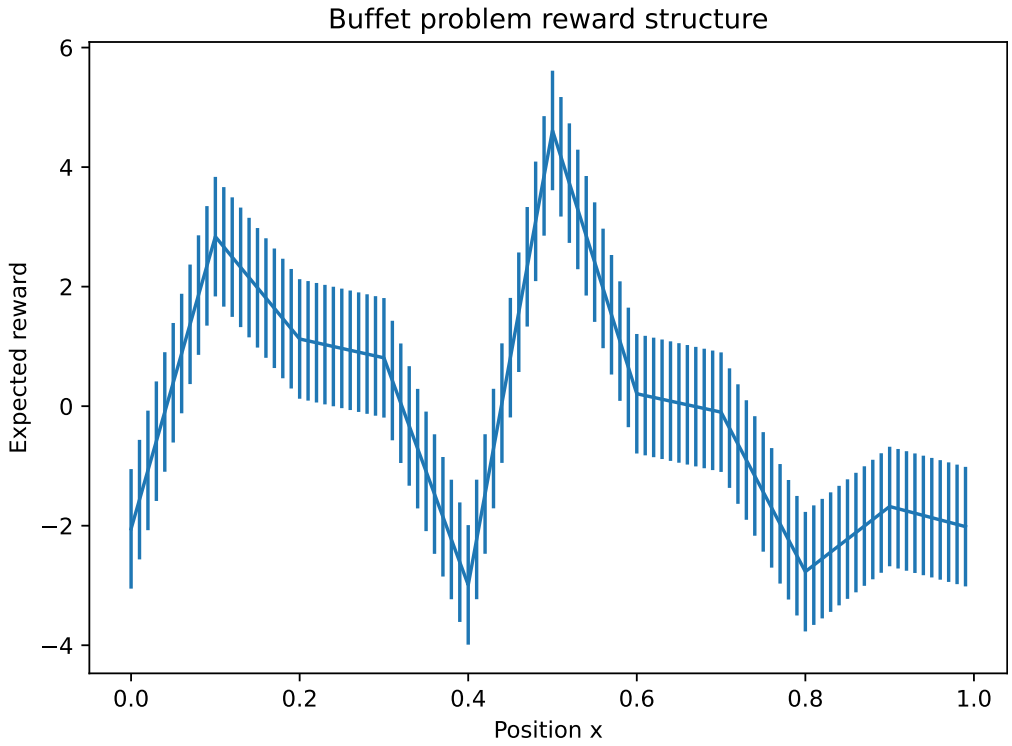

Figure A.1: Reward structure of an example Buffet POMDP instance with $N = 10$ "foods" available. After each action that the diner makes to change their position, they receive a stochastic reward (error bars show the standard deviation of 1) based on their current position relative to the available foods. Because the actions are limited to only a change in position of at most 0.075, a greedy policy may never discover high reward foods (positions) far from the diner.

The position of the diner is known and deterministic and ranges from 0 to 1, modulo 1, always starting at 0. The diner's action space consists of a change in position, limited to the range -0.075 to 0.075, and wrapping around at 0 and 1. There are N "foods" evenly spaced over the range of positions and they each have a stochastic reward. The total reward is linearly interpolated between the nearest rewards. For example, if $N = 2$, the foods will be located at 0 and 0.5. If the expected rewards are 10 and -10, respectively, and the diner is at position 0.1, then the expected reward will be $10 * 0.4/0.5 - 10 * 0.1/0.5 = 6$. For our evaluation, we use $N = 10$, and draw each food's reward from a normal distribution with standard deviation 1, where the mean value is itself drawn from a normal distribution with a standard deviation of 10. We normalize the distributions so the mean expected value across distributions is 0. The reward structure for an example problem instance with $N = 10$ is shown in Figure A.1.

Table A.1: Expected mean rollout value error by number of rollouts for the Buffet POMDP. For 2048 random state-action pairs, we examined how quickly the mean rollout values converged as we ran more and more rollouts. We then determined the percentage of those 2048 state-action pairs that where within some error margin (10, 5, 2.5, and 1.25 percent) of the later converged values.

# rollouts	$< 10\%$	$< 5\%$	$< 2.5\%$	$< 1.25\%$
2048	87.9	77.2	57.5	34.5
4096	92.1	83.5	66.9	45.4
8192	94.6	88.6	77.2	57.0
16384	96.2	91.9	83.1	69.0

This problem intentionally provides a lot of correlated information. For a given belief, expected rewards are related when positions are less than $\frac{1}{N}$ apart. Even across multiple beliefs, expected rewards are related when positions are close and the beliefs are similar in the vicinity of those positions as well.

We limit the number of particles in the root particle filter to the number of Monte Carlo trials being used. The root particle filter is the same "observation adaptive particle filter" as used by the authors of POMCPOW [15] in their evaluation code, which is available online [114].

Parameters for data collection and training, including the final training regret, are given in Table A.2. We see that the value network achieves a lower regret, even though it is the comparison baseline for using a neural network. The number of rollouts to perform for each state-action tuple during data collection was chosen by looking at Table A.1 and choosing the lowest option that achieved less than 1.25% error for more than 50% of the tuples. Mean reward running the Buffet problem for POMCPOW, the value network, and for LSMCP are in Table A.3. POMCPOW alone achieves the worst performance and we see that the baseline value network generally does better than LSMCP, while also being faster than LSMCP.

Even though the modified forms of POMCPOW do earn higher rewards here, please note that the network evaluations result in a significant slow down, such that POMCPOW is still faster for the same reward level. While further optimizations are possible, including approximate nearest neighbors, we have not evaluated that option yet here. Finally we also note that the buffet POMDP was entirely designed for LSMCP to do well, so we need to try another POMDP for a more fair evaluation.

Table A.2: Parameters and results for data collection and network training. Mean regret (see above on the loss function) is given in standard deviations.

Parameter/result	Buffet	Van der Pol tag
Actions per state	32	32
Rollouts per state-action tuple	8192	512
References per query	1024	1024
Batch size (value network)	2048	4096
Batch size (LSMCP)	256	512
Dimension D (LSMCP)	6	4
Mean regret (value network)	0.066	0.27
Mean regret (LSMCP)	0.097	0.35

Table A.3: Mean reward on the Buffet POMDP by method. Both the baseline value network and LSMCP achieve about the same reward as POMCPOW but with only half as many trials, achieving 2x sample efficiency for this problem. However, POMCPOW with 128 trials is still faster than LSMCP with 64 trials.

Number of trials	POMCPOW	Value network	LSMCP
32	126.8 ($\pm$1.5)	171.1 ($\pm$1.6)	152.6 ($\pm$1.5)
64	156.5 ($\pm$1.5)	192.4 ($\pm$1.6)	189.1 ($\pm$1.6)
128	190.5 ($\pm$1.5)		

A.7.2 Van der Pol tag

In the Van der Pol tag problem [15], the agent attempts to "tag" a target that moves according to the Van der Pol differential equation in an oscillatory pattern. There are barriers that block the agent, but not the target, and the agent must pay a cost to take accurate measurements of the target's position. This is a fully continuous POMDP across state, action, and observations spaces.

The agent starts at the origin and the target starts at a random initial position from $[-4, 4] \times [-4, 4]$. If the agent gets within 0.1 of the target, it receives a reward of 100 and the problem terminates. The target moves according to the RK4-integrated Van der Pol differential equations: $\dot{x} = \mu(x - \frac{x^3}{3} - y)$ and $\dot{y} = \frac{x}{\mu}$, where $\mu = 2$. The target's position is also randomly perturbed each step with $\sigma = 0.05$ for both x and y. The agent moves at a constant velocity of 1 and chooses any angle for its motion. Barriers block the agent from crossing the $x = 0$ axis for $y \in [-2.8, -0.2] \cup [0.2, 2.8]$ and vice versa for the $y = 0$ axis. The agent receives a measurement of the target's position each step and chooses whether to pay a cost of -5 to receive a measurement with $\sigma = 0.1$ instead of $\sigma = 5$ (not useful). There is a cost of -1 for each step before termination. Observations consist of 8 beams with mean of either distance to the target (for the beam that includes the target) or a constant distance of 1 for the other beams. The Van der Pol differential

equations are illustrated as a vector field alongside the barriers in Figure A.2.

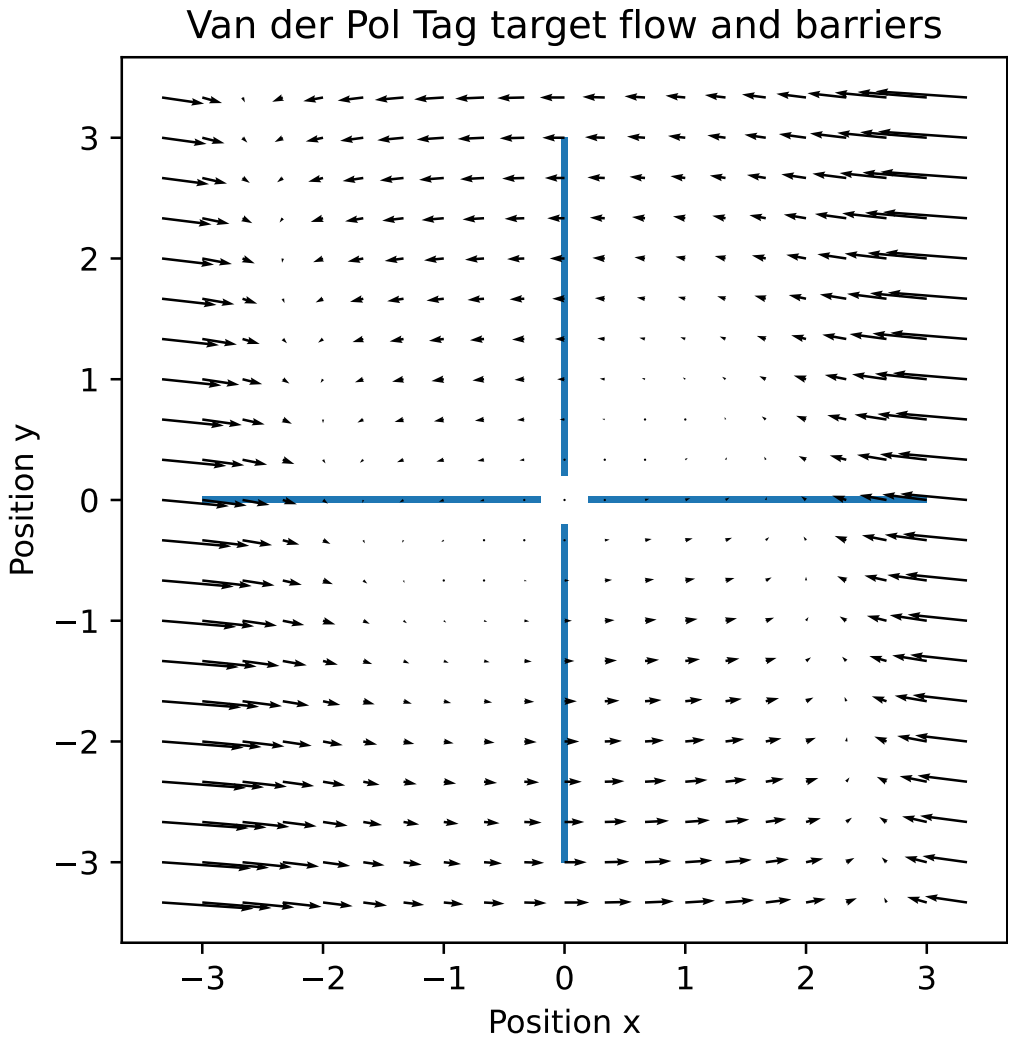

Figure A.2: Illustration of the Van der Pol vector field and barriers. The vector field determines the motion of the target from its random initial position. The barriers block the motion of the agent only.

This problem is interesting because it is simple, fully continuous, and has no obvious heuristic policy due to the barriers and difference in dynamics between the agent and target.

We make one change in how Van der Pol tag is configured as compared to the original authors. Because we do testing with far fewer Monte Carlo trials, we limit the number of particles in the root particle filter to the number of Monte Carlo trials being used, instead of having it fixed at 10,000 particles.

Parameters for data collection and training, including the final training regret, are given in Table A.2. These mean regrets are about four times worse than we had achieved in the buffet POMDP. The number of rollouts to perform for each state-action tuple during data collection was chosen by

Table A.4: Expected mean rollout value error by number of rollouts for the Van der Pol Tag POMDP. For 2048 random state-action pairs, we examined how quickly the mean rollout values converged as we ran more and more rollouts. We then determined the percentage of those 2048 state-action pairs that where within some error margin (10, 5, 2.5, and 1.25 percent) of the later converged values.

# rollouts	$< 10\%$	$< 5\%$	$< 2.5\%$	$< 1.25\%$
128	99.1	90.3	63.4	36.0
256	99.9	97.3	77.9	48.7
512	99.9	99.6	91.1	64.5
1024	99.9	99.9	97.7	79.3

Table A.5: Mean reward on the Van der Pol Tag POMDP by method. The baseline value network generally outperforms POMCPOW, but with a low confidence and significantly worse time. LSMCP is the worst performing in both mean final reward and computational time.

Number of trials	POMCPOW	Value network	LSMCP
256	3.1 (± 1.0)	5.3 (± 0.9)	-4.3 (± 2.7)
512	8.9 (± 0.9)	8.5 (± 2.6)	-3.5 (± 2.6)
1024	10.8 (± 0.9)	15.4 (± 2.9)	-1.7 (± 4.6)

looking at Table A.4, using the same criteria as for Buffet. Comparing tables, we see that Van der Pol Tag has much lower rollout value variance than the Buffet problem. Mean reward running the Buffet problem for POMCPOW, the value network, and for LSMCP are in Table A.5.

LSMCP does much worse than POMCPOW even as it is much slower. The baseline value network generally does better than POMCPOW, but without a very high confidence, and it is still significantly slower overall. LSMCP overall exhibits significantly worse performance. At lower numbers of trials not shown here, LSMCP actually appears to beat POMCPOW, but this is simply because it learns to not pay for costly observations, not because it completes the task successfully more than POMCPOW.

A.8 Challenges and alternatives

The performance advantage when using the direct value network against POMCPOW validates that the training data and loss function are sufficient to learn values that improve overall performance of the search. However, as we see by the worse mean regret during training of the LSMCP distance-based network as compared to the direct value network, we are somehow not expressive enough to capture the information we need. This might be because of constraints based around our choice of Euclidean distance and another distance metric might do better. This might be because not enough

similarities can even be drawn from the 1024 references per query. This might be because the Van der Pol tag problem simply is not exploitable from a similarity perspective.

One alternative would be to find a way to learn just how much similarity can be exploited, so that at worst LSMCP does not degrade performance. Then it might be useful to search other POMDPs to see if they might be more exploitable. If none are, then perhaps there is a more fundamental issue in how we are trying to learn and express similarity. One observation is that in the buffet POMDP, we only manage to achieve a 2x sample efficiency with LSMCP (see Table A.3). The degree of exploitability should be greater than that. There are likely also issues in LSMCP as it currently stands with incorporating similarity-derived expected rollout values into POMCPOW.

Another angle that might be useful is to try to derive properties and bounds on how exploitable a simple POMDP like buffet should be from a mathematical perspective. If we can find rigorous bounds and the conditions that they hold under, this may help reformulate similarity, Euclidean dimension size, and a distance metric so that they are more effective together.

A.8.1 Acknowledgements

This work was supported by grants from the NSF (1830615).

BIBLIOGRAPHY

[1] R. J. Meinhold and N. D. Singpurwalla, "Understanding the Kalman filter," *The American Statistician*, vol. 37, no. 2, pp. 123–127, 1983.

[2] 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*, vol. 30, no. 7, pp. 895–913, 2011.

[3] P. Müller, H. Wymeersch, and R. Piché, "UWB positioning with generalized Gaussian mixture filters," *IEEE Transactions on Mobile Computing*, vol. 13, no. 10, pp. 2406–2414, 2014.

[4] E. A. Wan and R. Van Der Merwe, "The unscented Kalman filter for nonlinear estimation," in *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373)*. Ieee, 2000, pp. 153–158.

[5] Y. S. Shmaliy, "Suboptimal FIR filtering of nonlinear models in additive white Gaussian noise," *IEEE Transactions on Signal Processing*, vol. 60, no. 10, pp. 5519–5527, 2012.

[6] Y. Xu, Y. S. Shmaliy, Y. Li, and X. Chen, "UWB-based indoor human localization with time-delayed data using EFIR filtering," *IEEE Access*, vol. 5, pp. 16 676–16 683, 2017.

[7] H. Nurminen, T. Ardeshiri, R. Piché, and F. Gustafsson, "A NLOS-robust TOA positioning filter based on a skew-t measurement noise model," in *2015 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. IEEE, 2015, pp. 1–7.

[8] C. E. Rasmussen *et al.*, "The infinite Gaussian mixture model." in *NIPS*, vol. 12. Citeseer, 1999, pp. 554–560.

[9] J. E. Griffin and M. F. Steel, "Bayesian nonparametric modelling with the Dirichlet process regression smoother," *Statistica Sinica*, pp. 1507–1527, 2010.

[10] E. Schulz, M. Speekenbrink, and A. Krause, "A tutorial on Gaussian process regression: Modelling, exploring, and exploiting functions," *Journal of Mathematical Psychology*, vol. 85, pp. 1–16, 2018.

[11] C. H. Tong, P. Furgale, and T. D. Barfoot, "Gaussian process Gauss–Newton for non-parametric simultaneous localization and mapping," *The International Journal of Robotics Research*, vol. 32, no. 5, pp. 507–525, 2013.

[12] A. Haggenmiller, M. Krogius, and E. Olson, "Non-parametric error modeling for ultra-wideband localization networks," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2019, © 2019 IEEE. Reprinted, with permission.

[13] A. Haggenmiller, C. Kabacinski, M. Krogius, and E. Olson, "The masked mapper: Masked metric mapping," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2020, © 2020 IEEE. Reprinted, with permission.

[14] A. Haggenmiller and E. Olson, "Monte-Carlo policy-tree decision making," University of Michigan APRIL Laboratory, Tech. Rep., March 2022.

[15] 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.

[16] A. Garivier and O. Cappé, "The KL-UCB algorithm for bounded stochastic bandits and beyond," in *Proceedings of the 24th annual conference on learning theory*. JMLR Workshop and Conference Proceedings, 2011, pp. 359–376.

[17] A. G. Cunningham, E. Galceran, R. M. Eustice, and E. Olson, "MPDM: Multipolicy decision-making in dynamic, uncertain environments for autonomous driving," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, June 2015.

[18] 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)*. IEEE, 2020, pp. 3291–3297.

[19] *APS014: DW1000 Antenna Delay Calibration*, decaWave, 2014, version 1.01. [Online]. Available: https://decawave.com/sites/default/files/aps014-antennadelaycalibrationofdw1000-basedproductsandsystems_v1.01.pdf

[20] *APS006 Part 2 Application Node: Non Line of Sight Operation and Optimizations to Improve Performance in DWM1000 Based Systems*, decaWave, 2014, version 1.4. [Online]. Available: https://decawave.com/sites/default/files/aps006_part2_nlos_operation_and_optimizations.pdf

[21] A. A. Kannan, B. Fidan, G. Mao, and B. D. Anderson, "Analysis of flip ambiguities in distributed network localization," in *Information, Decision and Control, 2007. IDC'07*. IEEE, 2007, pp. 193–198.

[22] D. Moore, J. Leonard, D. Rus, and S. Teller, "Robust distributed network localization with noisy range measurements," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*. ACM, 2004, pp. 50–61.

[23] K. Chintalapudi, A. Padmanabha Iyer, and V. N. Padmanabhan, "Indoor localization without the pain," in *Proceedings of the sixteenth annual international conference on Mobile computing and networking*. ACM, 2010, pp. 173–184.

[24] R. Faragher and R. Harle, "Location fingerprinting with bluetooth low energy beacons," *IEEE journal on Selected Areas in Communications*, vol. 33, no. 11, pp. 2418–2428, 2015.

[25] J. Wang, Q. Gao, Y. Yu, H. Wang, and M. Jin, "Toward robust indoor localization based on Bayesian filter using chirp-spread-spectrum ranging," *IEEE Transactions on Industrial Electronics*, vol. 59, no. 3, pp. 1622–1629, 2012.

[26] C. Zhang, M. Kuhn, B. Merkl, M. Mahfouz, and A. E. Fathy, "Development of an UWB indoor 3d positioning radar with millimeter accuracy," in *Microwave Symposium Digest, 2006. IEEE MTT-S International*. IEEE, 2006, pp. 106–109.

[27] A. A. Kannan, G. Mao, and B. Vucetic, "Simulated annealing based localization in wireless sensor network," in *Local Computer Networks, 2005. 30th Anniversary. The IEEE Conference on*. IEEE, 2005, pp. 2–pp.

[28] T.-C. Liang, T.-C. Wang, and Y. Ye, "A gradient search method to round the semidefinite programming relaxation solution for ad hoc wireless sensor network localization," *Sanford University, formal report*, vol. 5, 2004.

[29] X. Ji and H. Zha, "Sensor positioning in wireless ad-hoc sensor networks using multidimensional scaling," in *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 4. IEEE, 2004, pp. 2652–2661.

[30] C. Di Franco, A. Prorok, N. Atanasov, B. Kempke, P. Dutta, V. Kumar, and G. J. Pappas, "Calibration-free network localization using non-line-of-sight ultra-wideband measurements," in *Proceedings of the 16th ACM/IEEE International Conference on Information Processing in Sensor Networks*, ser. IPSN '17. New York, NY, USA: ACM, 2017, pp. 235–246. [Online]. Available: http://doi.acm.org/10.1145/3055031.3055091

[31] B. Kempke, P. Pannuto, B. Campbell, and P. Dutta, "Surepoint: Exploiting ultra wideband flooding and diversity to provide robust, scalable, high-fidelity indoor localization," in *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*. ACM, 2016, pp. 137–149.

[32] F. Despaux, K. Jaffres-Runser, A. Van den Bossche, and T. Val, "Accurate and platform-agnostic time-of-flight estimation in ultra-wide band," in *Personal, Indoor, and Mobile Radio Communications (PIMRC), 2016 IEEE 27th Annual International Symposium on*. IEEE, 2016, pp. 1–7.

[33] J. D. Hol, F. Dijkstra, H. Luinge, and T. B. Schon, "Tightly coupled UWB/IMU pose estimation," in *2009 IEEE International Conference on Ultra-Wideband*. IEEE, 2009, pp. 688–692.

[34] L. Zwirello, X. Li, T. Zwick, C. Ascher, S. Werling, and G. F. Trommer, "Sensor data fusion in UWB-supported inertial navigation systems for indoor navigation," in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 3154–3159.

[35] *DWM1000 Datasheet*, decaWave, 2016, version 1.6. [Online]. Available: https://www.decawave.com/sites/default/files/resources/DWM1000-Datasheet-V1.6.pdf

[36] *APS014 Application Node: Sources of Error in DW1000 Based Two-Way Ranging (TWR) Schemes*, decaWave, 2014, version 1.0. [Online]. Available: https://decawave.com/sites/default/files/aps011_sources_of_error_in_twr.pdf

[37] *DW1000 User Manual*, decaWave, 2017, version 2.15. [Online]. Available: https://www.decawave.com/sites/default/files/dw100020user20manual_0.pdf

[38] X. Wang, R. Marcotte, G. Ferrer, and E. Olson, "AprilSAM: real-time smoothing and mapping," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 2486–2493.

[39] B. Kuipers, R. Browning, B. Gribble, M. Hewett, and E. Remolina, "The spatial semantic hierarchy," *Artificial intelligence*, 2000.

[40] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous robots*, vol. 4, no. 4, pp. 333–349, 1997.

[41] S. Thrun, D. Hahnel, D. Ferguson, M. Montemerlo, R. Triebel, W. Burgard, C. Baker, Z. Omohundro, S. Thayer, and W. Whittaker, "A system for volumetric robotic mapping of abandoned mines," in *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, vol. 3. IEEE, 2003, pp. 4270–4275.

[42] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2D LIDAR SLAM," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 1271–1278.

[43] S. Thrun, "Learning metric-topological maps for indoor mobile robot navigation," *Artificial Intelligence*, vol. 99, no. 1, pp. 21–71, 1998.

[44] T. Röfer, "Route navigation using motion analysis," in *International Conference on Spatial Information Theory*. Springer, 1999, pp. 21–36.

[45] A. Lankenau, T. Röfer, and B. Krieg-Brückner, "Self-localization in large-scale environments for the bremen autonomous wheelchair," in *International Conference on Spatial Cognition*. Springer, 2002, pp. 34–61.

[46] B. Kuipers, J. Modayil, P. Beeson, M. MacMahon, and F. Savelli, "Local metrical and global topological maps in the hybrid spatial semantic hierarchy," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, vol. 5. IEEE, 2004, pp. 4845–4851.

[47] P. Beeson, J. Modayil, and B. Kuipers, "Factoring the mapping problem: Mobile robot map-building in the hybrid spatial semantic hierarchy," *The International Journal of Robotics Research*, vol. 29, no. 4, pp. 428–459, 2010.

[48] C. Johnson, "Topological mapping and navigation in real-world environments," Ph.D. dissertation, University of Michigan, 2018.

[49] J.-L. Blanco, J.-A. Fernández-Madrigal, and J. Gonzalez, "Toward a unified bayesian approach to hybrid metric–topological SLAM," *IEEE Transactions on Robotics*, vol. 24, no. 2, pp. 259–270, 2008.

[50] S. Thrun, W. Burgard, and D. Fox, "A probabilistic approach to concurrent mapping and localization for mobile robots," *Autonomous Robots*, vol. 5, no. 3-4, pp. 253–271, 1998.

[51] C. Kerl, J. Sturm, and D. Cremers, "Dense visual SLAM for RGB-D cameras," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 2100–2106.

[52] J. Neira, J. D. Tardós, and J. A. Castellanos, "Linear time vehicle relocation in SLAM," in *ICRA*. Citeseer, 2003, pp. 427–433.

[53] C. Estrada, J. Neira, and J. D. Tardós, "Hierarchical SLAM: Real-time accurate mapping of large environments," *IEEE transactions on Robotics*, vol. 21, no. 4, pp. 588–596, 2005.

[54] R. Dubé, D. Dugas, E. Stumm, J. Nieto, R. Siegwart, and C. Cadena, "Segmatch: Segment based place recognition in 3d point clouds," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 5266–5272.

[55] M. Bosse, P. Newman, J. Leonard, and S. Teller, "Simultaneous localization and map building in large-scale cyclic environments using the Atlas framework," *The International Journal of Robotics Research*, vol. 23, no. 12, pp. 1113–1139, 2004.

[56] E. Olson, "M3RSM: Many-to-many multi-resolution scan matching," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, June 2015.

[57] K. J. Astrom, "Optimal control of Markov decision processes with incomplete state estimation," *J. Math. Anal. Applic.*, vol. 10, pp. 174–205, 1965.

[58] R. D. Smallwood and E. J. Sondik, "The optimal control of partially observable Markov processes over a finite horizon," *Operations research*, vol. 21, no. 5, pp. 1071–1088, 1973.

[59] G. A. Hollinger, "Long-horizon robotic search and classification using sampling-based motion planning." in *Robotics: Science and Systems*, vol. 3, 2015.

[60] T. Jaakkola, S. P. Singh, and M. I. Jordan, "Reinforcement learning algorithm for partially observable Markov decision problems," *Advances in neural information processing systems*, pp. 345–352, 1995.

[61] D. Silver and J. Veness, "Monte-Carlo planning in large POMDPs," in *Advances in neural information processing systems*, 2010, pp. 2164–2172.

[62] K. Hsiao, L. P. Kaelbling, and T. Lozano-Perez, "Grasping POMDPs," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, 2007, pp. 4685–4692.

[63] M. L. Littman, "A tutorial on partially observable Markov decision processes," *Journal of Mathematical Psychology*, vol. 53, no. 3, pp. 119–125, 2009.

[64] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *European conference on machine learning*. Springer, 2006, pp. 282–293.

[65] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of Monte Carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[66] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search," in *International conference on computers and games*. Springer, 2006, pp. 72–83.

[67] G. E. Monahan, "State of the art—a survey of partially observable Markov decision processes: theory, models, and algorithms," *Management science*, vol. 28, no. 1, pp. 1–16, 1982.

[68] J. Satia and R. Lave, "Markovian decision processes with probabilistic observation of states," *Management Science*, vol. 20, no. 1, pp. 1–13, 1973.

[69] A. McGovern, R. S. Sutton, and A. H. Fagg, "Roles of macro-actions in accelerating reinforcement learning," in *Grace Hopper celebration of women in computing*, vol. 1317, 1997, p. 15.

[70] G. Theocharous and L. P. Kaelbling, "Approximate planning in POMDPs with macro-actions," in *Advances in Neural Information Processing Systems*, 2004, pp. 775–782.

[71] R. He, E. Brunskill, and N. Roy, "PUMA: Planning under uncertainty with macro-actions." in *AAAI*, 2010, p. 7.

[72] A.-A. Agha-Mohammadi, S. Chakravorty, and N. M. Amato, "FIRM: Feedback controller-based information-state roadmap–a framework for motion planning under uncertainty," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 4284–4291.

[73] A. Bry and N. Roy, "Rapidly-exploring random belief trees for motion planning under uncertainty," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 723–730.

[74] F. Doshi-Velez, "The infinite partially observable Markov decision process," *Advances in neural information processing systems*, vol. 22, pp. 477–485, 2009.

[75] M. J. Beal, Z. Ghahramani, and C. E. Rasmussen, "The infinite hidden Markov model," *Advances in neural information processing systems*, vol. 1, pp. 577–584, 2002.

[76] J. Van Gael, Y. Saatci, Y. W. Teh, and Z. Ghahramani, "Beam sampling for the infinite hidden Markov model," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 1088–1095.

[77] C. Amato and F. Oliehoek, "Scalable planning and learning for multiagent POMDPs," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, no. 1, 2015.

[78] C. Guestrin, D. Koller, and R. Parr, "Multiagent planning with factored MDPs," in *NIPS*, vol. 1, 2001, pp. 1523–1530.

[79] G. Best, O. M. Cliff, T. Patten, R. R. Mettu, and R. Fitch, "Decentralised Monte Carlo tree search for active perception," in *WAFR*, 2016.

[80] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of Go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[81] T. M. Moerland, J. Broekens, A. Plaat, and C. M. Jonker, "A0c: Alpha zero in continuous action space," *arXiv preprint arXiv:1805.09613*, 2018.

[82] W. Ding, J. Chen, and S. Shen, "Predicting vehicle behaviors over an extended horizon using behavior interaction network," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8634–8640.

[83] C. Paxton, V. Raman, G. D. Hager, and M. Kobilarov, "Combining neural networks and tree search for task and motion planning in challenging environments," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 6059–6066.

[84] M. Mukadam, A. Cosgun, A. Nakhaei, and K. Fujimura, "Tactical decision making for lane changing with deep reinforcement learning," in *NIPS 2017 Workshop on Machine Learning for Intelligent Transportation Systems*, 12 2017.

[85] E. Galceran, A. G. Cunningham, R. M. Eustice, and E. Olson, "Multipolicy decision-making for autonomous driving via changepoint-based behavior prediction," in *Proceedings of Robotics: Science and Systems (RSS)*, Rome, Italy, July 2015.

[86] D. Mehta, G. Ferrer, and E. Olson, "Fast discovery of influential outcomes for risk-aware MPDM," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2017.

[87] ——, "Backprop-MPDM: Faster risk-aware policy evaluation through efficient gradient optimization," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2018.

[88] B. Zhou, W. Schwarting, D. Rus, and J. Alonso-Mora, "Joint multi-policy behavior estimation and receding-horizon trajectory planning for automated urban driving," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 2388–2394.

[89] R. Morton and E. Olson, "Robust sensor characterization via max-mixture models: GPS sensors," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, November 2013.

[90] R. W. Wolcott and R. M. Eustice, "Robust LIDAR localization using multiresolution Gaussian mixture maps for autonomous driving," *The International Journal of Robotics Research*, vol. 36, no. 3, pp. 292–319, 2017.

[91] G. Chaslot, M. Winands, H. Herik, J. Uiterwijk, and B. Bouzy, "Progressive strategies for Monte-Carlo tree search," *New Mathematics and Natural Computation*, vol. 04, pp. 343–357, 11 2008.

[92] D. Michie and R. A. Chambers, "BOXES: An experiment in adaptive control," *Machine intelligence*, vol. 2, no. 2, pp. 137–152, 1968.

[93] J.-Y. Audibert, R. Munos, and C. Szepesvári, "Exploration–exploitation tradeoff using variance estimates in multi-armed bandits," *Theoretical Computer Science*, vol. 410, no. 19, pp. 1876–1902, 2009.

[94] Y. Abbasi-Yadkori, D. Pál, and C. Szepesvári, "Improved algorithms for linear stochastic bandits," *Advances in neural information processing systems*, vol. 24, pp. 2312–2320, 2011.

[95] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Physical review E*, vol. 62, no. 2, p. 1805, 2000.

[96] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, Tech. Rep., 1992.

[97] 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*, vol. 4, no. 3, pp. 2997–3004, 2019.

[98] A. Somani, N. Ye, D. Hsu, and W. S. Lee, "DESPOT: Online POMDP planning with regularization," *Advances in neural information processing systems*, vol. 26, pp. 1772–1780, 2013.

[99] R. Coulom, "Computing "elo ratings" of move patterns in the game of go," *ICGA journal*, vol. 30, no. 4, pp. 198–208, 2007.

[100] A. Couëtoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard, "Continuous upper confidence trees," in *International conference on learning and intelligent optimization*. Springer, 2011, pp. 433–445.

[101] Y. Luo, H. Bai, D. Hsu, and W. S. Lee, "Importance sampling for online planning under uncertainty," *The International Journal of Robotics Research*, vol. 38, no. 2-3, pp. 162–181, 2019.

[102] N. P. Garg, D. Hsu, and W. S. Lee, "DESPOT-alpha: Online POMDP planning with large state and observation spaces." in *Robotics: Science and Systems*, 2019.

[103] H. Kurniawati, D. Hsu, and W. S. Lee, "SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces." in *Robotics: Science and systems*, vol. 2008. Citeseer, 2008.

[104] H. Finnsson and Y. Björnsson, "Simulation-based approach to general game playing." in *Aaai*, vol. 8, 2008, pp. 259–264.

[105] H. Finnsson *et al.*, "Simulation-based general game playing," Ph.D. dissertation, Háskólinn í Reykjavík, June 2012.

[106] M. J. Tak, M. H. Winands, and Y. Bjornsson, "N-grams and the last-good-reply policy applied in general game playing," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 73–83, 2012.

[107] M. J. Tak, M. H. Winands, and Y. Björnsson, "Decaying simulation strategies," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 4, pp. 395–406, 2014.

[108] S. Gelly and D. Silver, "Monte-Carlo tree search and rapid action value estimation in computer go," *Artificial Intelligence*, vol. 175, no. 11, pp. 1856–1875, 2011.

[109] T. Cazenave, "Generalized rapid action value estimation," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[110] C. F. Sironi and M. H. Winands, "Comparison of rapid action value estimation variants for general game playing," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2016, pp. 1–8.

[111] A. Couëtoux, M. Milone, M. Brendel, H. Doghmen, M. Sebag, and O. Teytaud, "Continuous rapid action value estimates," in *Asian conference on machine learning*. PMLR, 2011, pp. 19–31.

[112] M. H. Lim, C. J. Tomlin, and Z. N. Sunberg, "Voronoi progressive widening: Efficient online solvers for continuous state, action, and observation POMDPs," in *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE, 2021, pp. 4493–4500.

[113] J. Mern, A. Yildiz, Z. Sunberg, T. Mukerji, and M. J. Kochenderfer, "Bayesian optimized Monte Carlo planning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 13, 2021, pp. 11 880–11 887.

[114] Z. N. Sunberg. ContinuousPOMDPTreeSearchExperiments: Experiments for designing tree search algorithms for continuous POMDPs. [Online]. Available: https://github.com/zsunberg/ContinuousPOMDPTreeSearchExperiments.jl