2020

Navigation under Obstacle Motion Uncertainty using Markov Decision Processes

Jennifer Quyen Nguyen
West Virginia University, jqnguyen@mix.wvu.edu

Navigation under Obstacle Motion Uncertainty using Markov Decision Processes

Jennifer Nguyen

Thesis submitted to the
Benjamin M. Statler College of Engineering and Mineral Resources
at West Virginia University

in partial fulfillment of the requirements for the degree of

Master of Science in
Mechanical Engineering

Yu Gu, Ph.D., Chair
Jason Gross, Ph.D.
Powsiri Klinkhachorn, Ph.D.

Department of Mechanical and Aerospace Engineering

Morgantown, West Virginia
2020

Keywords: Navigation, Obstacle Avoidance, MDP, POMDP, QMDP

Abstract

Navigation under Obstacle Motion Uncertainty using Markov Decision Processes

Jennifer Q. Nguyen

In terms of navigation, a central problem in the field of autonomous robotics is obstacle avoidance. This research explores how to navigate as well as avoid obstacles by leveraging what is known of the environment to determine decisions with new incoming information during execution. The algorithm presented in this work is divided into two procedures: an *offline* process that uses prior knowledge to navigate toward the goal; and an online execution strategy that leverages results obtained offline to drive safely towards the target when new information is encountered (e.g., obstacles). To take advantage of what is known offline, the navigation problem was formulated as a Markov Decision Process (MDP) where the environment is characterized as an occupancy grid. Baseline dynamic programming techniques were used to solve this, producing general behaviors that drive the robot (or agent) toward the goal and a value function which encodes the value of being in particular states. Then during online execution, the agent uses these offline results and surrounding local information of the environment to operate (e.g., data from a LIDAR sensor). This locally acquired information, which may contain new data not seen prior, is represented as a small occupancy grid and leverages the offline obtained value function to define local goals allowing the agent to make short term plans. When the agent encounters an obstacle locally, the problem becomes a Partially Observable Markov Decision Process (POMDP) since it is uncertain where these obstacles will be in the next state. This is solved by utilizing an approximate planner (QMDP) that uses uncertainty of the obstacle motion and considers all possible obstacle state combinations in the next time step to determine the best action. The approximate planner can quickly solve the POMDP, due to the small size of the local occupancy grid and by using the behaviors produced offline to help speed up convergence, which opens the possibility for this procedure to be executed in real time, on a physical robot. Two simulated environments were created, varying in complexity and dynamic obstacles. Simulation results under complex conditions with narrow operable spaces and many dynamic obstacles show the proposed algorithm has approximately an 85% success rate, in test cases with cluttered environments and multiple dynamic obstacles, and is shown to produce safer trajectories than the baseline approach, which had roughly a 37% success rate, under the assumptions that dynamic obstacles can only move a short distance by the next time step.

Acknowledgments

Without the support from friends, family, and colleagues this work would not be possible. I wish to thank everyone who provided assistance and briefly acknowledge a few of them.

I would first like to thank **Jared Strader** , who was instrumental in helping cultivate the proper methodology and understanding needed to make this research possible. For his assistance and ideas throughout the process.

Next, I would like to show gratitude to **Dr. Yu Gu** for being a great advisor in graduate school and life. Thank you for always being available to offer valuable guidance and support. For providing the opportunity to expand my knowledge, the freedom to explore ideas, and helping me grow.

Additionally, I would like to recognize all members of the **Interactive Robotics Laboratory (IRL)** , who have always been there for help whenever needed. I am extremely grateful for the kindness, willingness to assist, and constant enthusiasm which all helped sustain my motivation. For creating a fun work environment and always finding ways to laugh in difficult times. To mention a few, I would like to thank **Nicholas Ohi** for encouraging me to attend graduate school and always being ready to contribute solutions to help me get unstuck and to **Conner Castle** for helping me build the omnidirectional robot used in this research.

Also, I would like to acknowledge the assistance provided by the **NASA West Virginia Space Grant Consortium** (Training Grant #NNX15AI01H) and **West Virginia Education Policy Commission** (Award #HEPC.dsr.18.5), making this research possible.

Last, I express my deepest gratitude to **David Donley** for his tremendous support and encouragement. For his help during difficult graduate school and life challenges. I am truly grateful and look forward to our future.

Contents

List of Figures

List of Tables

List of Acronyms

RRT Rapidly-exploring Random Trees 7
GP Gaussian Process 9
ORCA Optimal Reciprocal Collision Avoidance 9
APF Artifical Potential Field 7
MDP Markov Decision Process 2
RL Reinforcement Learning 8
PI Policy Iteration 18
G-PI Global Policy Iteration 23
L-PI Local Policy Iteration 26
POMDP Partially Observable Markov Decision Process 4
NN Neural Network 7
MC Monte Carlo 44
MC-sims Monte Carlo Simulations vii
PI+QMDP Policy Iteration and QMDP vi
PI only Policy Iteration only vi

List of Symbols

1

Introduction

1.1 Motivation

The field of robotics is an increasingly growing area of interest with various applications including service robotics (e.g., personal, healthcare, agriculture, and educational), industrial robots, and space exploration. As time goes on, researchers will find more ways to solve various problems using robotics. Advances in perception, localization, mapping, and decision making have provided many capable autonomous navigation strategies with several in the form of path planning. One of the central problems to navigation is obstacle avoidance, especially in domains such as navigating around pedestrians, multi-agent coordination, and self-driving cars.

In cases where the environment is simple, known prior, and contain few static obstacles it is quite easy to navigate around to a specified goal position through simple, pre-programmed instructions (e.g., obstacle on the left, turn right). In these situations, it can be said that the robot is not truly autonomous but following delayed guidance from a human programmer. This becomes much more difficult and infeasible in cluttered, unknown environments, especially in the presence of dynamic obstacles. To handle these difficult environments, some solutions use velocity estimates of obstacles to determine the robot's own velocities to avoid hazards [1–4] but require perfect perception and can be difficult to implement due to ambiguous parameters. Several solutions attempt to model the trajectory or intent of obstacles through machine learning, which are used to determine a navigation strategy around them [5–8]. Unfortunately, the real world is stochastic and complex making it difficult to properly prepare these models. Another issue is these solutions tend to be case specific, meaning it works for that particular robot and is not generalized in a way that can be easily applied to other robots. Solutions including artificial potential fields [9–13] can be computationally expensive and difficult to execute in real time. The advantage of Markov Decision Pro-

cess (MDP) is that they provide a framework to make decisions in stochastic environments but it is difficult to formulate the problem such that the state space captures all the necessary information. Several techniques attempt to include position and velocity in the states such as [14] but the explosion of states makes it difficult to train quickly or well (i.e., *curse of dimensionality*). Other MDP techniques attempt to discretize the states including [15] but are typically not able to generalized well enough to handle new situations.

The motivation behind this work is to avoid using traditional navigation and obstacle avoidance planners by utilizing the flexible MDP framework, to be able to provide a solution at any location in the operating environment given limited information, and quickly consider possible obstacle states to produce safer actions.

1.2 Problem Statement

The objective of this research is to determine a solution for dynamic obstacle avoidance, which began with the following questions:

1. Given what is known prior, which may not be accurate, how can this information be used in an offline training process to offload online computation and provide a closed-loop solution at any location?

2. Then during online operation, is there a way to leverage this prior knowledge to **quickly** determine new actions when encountering new information not previously seen and still account for uncertainty of the obstacle motion?

3. How can we generalize the solution so that it can be applicable to other robots that have the same drive-train?

This work presents an algorithm that provides solutions to the above questions and will be directly answered in the conclusion. The terms 'robot' and 'agent' will be used interchangeably throughout this thesis.

To narrow the scope and focus, the algorithm was designed for a holonomic (i.e., omnidirectional) robot, meaning the robot is able to drive in any direction, operating in a 2-dimensional environment. It is assumed that the robot will be able to accurately localize itself in the global frame as well as accurately map the local surroundings (i.e., 360 degrees) within a squared meter range since it has become a common capability following advances in readily available depth sensors (e.g., 2D LIDAR) and Simultaneous Localization and Mapping (SLAM) techniques. It is also assumed that occulusions do not occur and the obstacles move small distances randomly. Additionally, the global goal is provided and at a minimum the size of the operating environment is known prior.

1.3 Thesis Outline

The remainder of the thesis will be organized as follows:

Chapter 2: This chapter provides a literature review on several obstacle avoidance methods in static and dynamic environments including traditional, current, and learning based approaches.

Chapter 3: This chapter details the necessary background to understand Markov Decision Process (MDP) and Partially Observable Markov Decision Process (POMDP) fundamentals and methods utilized throughout this thesis.

Chapter 4: Within this chapter, defines the design and formulation of the algorithm used to solve navigation in the presence of dynamic obstacles.

Chapter 5: This chapter provides the setups used for evaluation and details the results.

Chapter 6: Lastly, concluding the thesis, is a discussion on the overall results, directly answered questions listed in the problem statement, closing remarks, and proposed future work.

2

Literature Review

This chapter provides a literature review on navigation in terms of obstacle avoidance. Discussed will be methods used static environments including reaction based techniques. Followed by a review on dynamic environments including MDP based approaches.

2.1 Static Environments

Reaction-based methods include condition-based (e.g., human designed heuristics) or one-step look ahead interaction guidelines based on the current state configuration to determine local obstacle avoidance maneuvers instead of planning paths. These approaches are normally coupled with high-level global path planners including Rapidly-exploring Random Trees (RRT) [16] or Dijkstra's algorithm [17] that find suitable paths to a goal location and perform reaction maneuvers when an obstacle is detected locally along the route. This requires manually creating decision rules that anticipate all potential scenarios, normally determined through trial and error, using fuzzy logic [18–20], and/or finite state machines [21, 22]. These techniques can work well in simple environments but unfortunately, the real world can be stochastic and complex making it infeasible to account for all situations. Also, this type of implementation becomes case dependent, which reduces adaptability when transitioning to new environments and generalization for other robot configurations. It could be said that these heuristic schemes are not truly autonomous but delayed human implemented logic and instructions.

Several Artifical Potential Field (APF) approaches exist including [9–13], which use virtual forces to determine paths while avoiding obstacles. Attractive forces are given to the goal and repulsive forces are given to the obstacle. A major disadvantage is when the attractive and repulsive force are equal or closely equal, the robot becomes trapped in local minima.

Some learning methods, leverage Neural Network (NN)s to learn navigation in the presence of

static obstacles. Tai et al. [23] formulated a mapless motion planner trained end-to-end through asynchronous deep-Reinforcement Learning (RL) for continuous control of a differential drive mobile robot. The authors used a custom asynchronous Deep Deterministic Policy Gradients (DDPG) to train their motion planner where the input was the state and the output was the action. The state was comprised of sparse 10-dimensional laser range readings, the previous action, and the goal position relative to the robot and the action was linear and angular velocity commands. Training was done in simulation and the model was evaluated in both simulation and reality. The robot was able to navigate successfully to a goal location in unseen static environments. Kahn et al. [24] formulated a model-based RL algorithm that learns a collision prediction model after experiencing low-impact collisions. This model estimates the probability of collision coupled with an uncertainty estimate, allowing a robot to operate cautiously (low velocities) in unknown environments, learn more about it after experiencing safe collisions, and increase its velocity once it becomes highly confident. The capability to learn how to avoid collisions after experiencing them was provided by their uncertainty-aware collision prediction model using deep NNs. The inputs to this network were raw camera data as well as a sequence of controls and the output was the collision probability along with an associated uncertainty estimate. These uncertainty-aware collision estimates were utilized with a speed-dependent collision cost to generate safe trajectories. The limitation to this is that success relies on the accuracy of the model's uncertainty estimate, if an overly optimistic estimation is reported in error then it could lead to disastrous collisions. Also, learning from raw camera data can be risky because small changes such as lighting can drastically alter perception.

These approaches are successful when obstacles are static but do not provide solutions for dynamic environments. Most robots operating in the real world need to account for movement of

obstacles. A review on navigation in the presence of dynamic obstacle avoidance is provided next.

2.2 Dynamic Environments

Trajectory-based methods attempt to determine collision free paths by predicting the motion of obstacles. In these techniques, past observations are used to determine the intent of the obstacle. Aoude et al. [5] use Gaussian Process (GP) mixture models to predict the future motion of obstacles to determine safe paths using chance constraint RRT. Other strategies use a cooperative approach, extending on trajectory-based methods, where they attempt to model how their own maneuvers will affect the motion of other neighboring agents. Trautman et al. [6] modeled the interactions between pedestrians and the robot used for planning in crowded areas using GP. A major pitfall to GPs is that the real-time performance degrades as it receives more data, making it challenging to learn unanticipated trajectories online. Another approach is to use these motion models to determine probability of collision [25, 26] and plan paths accordingly. A major limitation to most of these implementations is that when the environment is too diverse, the agent is unable to determine collision free paths and freezes in place. Also, these methods can be computationally expensive and incorrect predictions could result in catastrophic collisions. Another approach incorporates vehicle dynamics to determine possible future paths using state lattices [27, 28] and can handle constraints such as time, velocity, and/or position. The disadvantage is that evaluating possible future paths can require intensive calculations, making it difficult to be utilized online.

Velocity-based solutions are another form of reaction-based technique that has become attractive because of their robustness and guarantees of local collision free motions [1–4]. A popular approach is Optimal Reciprocal Collision Avoidance (ORCA) [2] which is a rule-based method that uses the position and velocity of its neighbors (i.e., obstacles) to determine a vector of feasible

velocities. The main disadvantages with most velocity-based approaches are that they are sensitive to perception uncertainty (i.e., require perfect sensing of the world to determine collision free motions) and may require hand-tuning several parameters including geometric properties as well as number of obstacles [29].

Some learning-based approaches, incorporate NNs to learn how to operate in dynamic environments. Chen et al. [7] developed a non-communicating multi-agent collision avoidance method using deep-RL. They use deep-RL to learn a value function that encodes cooperative behaviors mapping an agent's own state and its neighbors' states to an action that is collision free but requires perfect sensing. The authors extend their work to induce socially norm pedestrian behaviors in dynamic environments [8]. For both of these works, assumptions of other agents (e.g., pedestrians) trajectories in short time periods are made. The work is further extended without making assumptions on particular obstacles motion models [14]. However, the complex implemented approaches require estimates of the moving obstacles position, velocity, and radius to determine obstacle avoidance policies in dynamic environments making the solution less robust to perception uncertainty and requires heavy online computation.

Navigation amongst pedestrians by formulating the problem into POMDP has been addressed (e.g., [30]) but prior assumptions and the high computational cost limit the applicability to various scenarios. Mueller et al. [15] formulated the collision avoidance problem for small, multi-rotor aircrafts into a POMDP and used QMDP to determine an approximately optimal policy. Although this thesis also uses POMDP and QMDP, the formulation is not the same. In Mueller's formulation, the POMDP consisted of 8 states that are all naturally continuous (e.g., velocities and distances). These states were discretized to be able to use QMDP offline for finding a solution. For the coarse set of discretized states, there were 765,625 states, which took under 3 hours on a single-

core processor to find a solution. The fine set was 9,529,569 discrete states and using the $Q(s, a)$ matrix found in coarse set to help speed convergence, determining a solution took more than 11 days. The discretization scheme considerably impacts performance and needs to be balanced with the high computational cost. Other than the high demand of resources, another limitation is when working with continuous variables in the state space and attempting to discretize them, they do not typically generalize well enough to handle new situations. Although approaches that are able to provide plans for continuous states exist [31, 32], they are too computationally expensive to determine solutions online.

3

Background

An overview of the necessary background is provided within this chapter. The aim is to provided the foundation needed to develop an understanding of concepts and methods utilized throughout this thesis. To cultivate this understanding, a Markov Decision Process (MDP) is described in Section 3.1 and RL in Section 3.2 followed by details on their main subcomponents. An iterative solution is discussed in Section 3.8. Lastly, an introduction on Partially Observable Markov Decision Process (POMDP) in Section 3.9 along with an approach for producing approximate solutions in Section 3.10.

3.1 Markov Decision Processes

A finite Markov Decision Process (MDP) is a mathematical process used for modeling sequential decisions in stochastic environments. A MDP is defined as a tuple $\langle S, A, p(s'|s,a), r(s,a,s'), \gamma \rangle$, where

- S is the set of states (i.e., state space) with $s \in S$
- A is the set of actions (i.e., action space) with $a \in A$
- $p(s'|s,a) = Pr(s_{t+1} = s'|s_t = s, a_t = a)$ is the probability of transitioning to the next state s' after taking an action a from state s (i.e., state transition model)
- $r(s,a,s')$ is the expected immediate reward from taking action a in state s transitioning to state s' (i.e., reward function)
- $\gamma \in [0,1]$ is the discount factor

The transition probability $p(s'|s,a)$ characterizes the dynamics of the environment and holds the *Markov property* (i.e., the future is independent of the past given the present). This means MDPs do not rely on prior history and assume the present contains all the necessary information to predict the future.

The discount factor γ is used to weigh the importance of immediate verses long-term rewards. When γ is closer to 0, immediate rewards are favored whereas values closer to 1 puts more weight towards long-term rewards.

3.2 Reinforcement Learning

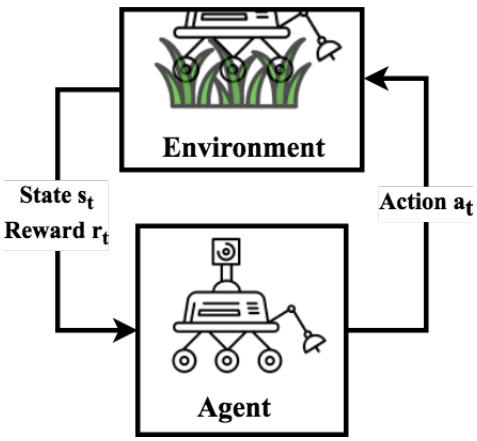


Figure 3.2.1: General Reinforcement Learning system

Reinforcement Learning (RL) is a branch of machine learning commonly used in sequential decision making problems where the dynamics (i.e., state-transition) and reward models are unknown or uncertain. RL can be modeled as a MDP, where an agent (i.e., learner) must interact with its environment to determine the actions that maximizes the long-term accumulation of rewards. Figure 3.2.1 illustrates an overall RL process. The agent observes the state of the environment, receives a reward, then based on the current state uses the policy (or behavior function) to determine the action to execute. The agent executes the action in the environment and this cycle continues.

The learner learns how to act through experience and is not told what actions to take but learns the actions that result in the highest return through trial-and-error. The main challenges in RL

include: balancing exploration and exploitation; delayed reward (i.e., actions may not affect immediate reward but subsequent rewards); and generalization. The designer needs to consider these challenges when crafting the RL algorithm such that the implementation optimizes the desired behavior of the agent. For more information on RL, please refer to [33].

The main subcomponents of a MDP and RL system are a model, a policy, a reward, and a value function. These elements will be detailed in the following next sections.

3.3 Model

The *state-transition model* $p(s'|s, a)$ is the agent's representation of the world or environment. The model predicts what the environment will do next when taking some action a in some state s. These models considers potential future situations to determine actions.

3.4 Policy

The *policy* π can be thought of as the decision-making rule. It is the agent's behavior function which is a mapping of states to actions. There are two types of policies:

- **Deterministic**: $\pi(s)$
- **Stochastic**: $\pi(a|s) = Pr(a_t = a|s_t = s)$

The focus within this thesis will be on deterministic policies $\pi(s)$.

3.5 Reward

The *reward function* defines the goal(s) of the agent and returns a numerical value (i.e., reward) given a current state, action, and future state at every time step (shown in Figure 3.2.1). The reward

defines the immediate good and bad situations (i.e., states) and closely relates to how animals in nature experience positive and negative reward signals when learning how to achieve a task.

The reward function is the main source for adjusting the policy. During execution, if the policy's action results in a low reward signal then the policy may update to better handle the scenario in future since the primary goal of the agent is to maximize the received total reward over time. Designing a reward function is one of the most challenging aspects because it requires manually crafting rewards such that the agent achieves desirable behavior and often requires fine-tuning until the desired behavior is achieved.

3.6 Value Function

Whereas the reward function represents the immediate reward, the *value function* estimates the expected long-term, accumulated reward from an observed state. Meaning, the value function represents how good it is to be in a state and the total rewards expected to be accumulated when starting from that state. For example, the immediate reward of a given state may be low but the value of the state may be high.

The *value function* $V_\pi(s)$ is defined as the expected total sum of rewards starting at state s and following the policy π thereafter:

$$V_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s \right]$$

Equivalently, the *Q-function* $Q_\pi(s, a)$, also known as the *action-value function*, is defined as the expected total sum of rewards taking action a in state s following the policy π thereafter:

$$Q_\pi(s,a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s, a_t = a \right]$$

3.7 Optimal Policies & Value Functions

An *optimal policy* π_* is a policy that maximizes the expected sum of rewards: $\pi \geq \pi'$ if and only if $V_\pi(s) \geq V_{\pi'}(s)$ for all $s \in S$. A policy π can be bad or good actions but an optimal policy π_* are the best actions. The optimal policy optimizes the amount of reward that the agent is expected to receive over its lifetime [34]:

$$\pi_* = \arg\max_\pi V_\pi(s), \ \forall\, s \in S$$

Therefore, the *optimal value function* $V_*(s)$ is defined as:

$$V_*(s) = \max_\pi V_\pi(s), \ \forall\, s \in S$$

Correspondingly, the *optimal Q-function* $Q_*(s,a)$ is defined by:

$$Q_*(s,a) = \max_\pi Q_\pi(s,a), \ \forall\, s \in S \text{ and } a \in A$$

Expanding these two equations, the *Bellman optimality equation* for $V_*(s)$ and $Q_*(s,a)$, respectively, are:

$$V_*(s) = \max_a \sum_{s'} p(s'|s,a)[r + \gamma V_*(s')]$$

$$Q_*(s,a) = \sum_{s'} p(s'|s,a)[r + \gamma \max_{a'} Q_*(s',a')]$$

Please refer to [33] for formal derivations.

3.8 Policy Iteration

Policy Iteration (PI) is a *dynamic programming* approach to solving finite MDPs, which involves iteratively improving the policy through roll-outs until convergence of the optimal policy and optimal value function. There are two main steps in PI, policy evaluation, which computes a consistent value function for the current policy, and policy improvement, which greedily improves the policy given the current value function. Given an intial, arbitrary policy π the algorithm steps are as follows:

1. **Policy Evaluation**: Compute $V_\pi(s)$ from π

   $$V_\pi(s) = \sum_{s'} p(s'|s,\pi(s))[r(s,\pi(s),s') + \gamma V_\pi(s')],\ \forall\, s \in S$$

2. **Policy Improvement**: Improve π from $V_\pi(s)$

   $$\pi(s) = \arg\max_a \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma V_\pi(s')],\ \forall\, s \in S$$

These steps are repeated until the policy can not be further improved or for a finite number of iterations specified by the algorithm designer. The end result is the optimal (or near optimal) policy and value function. The full algorithm is shown in algorithm 1.

3.9 Partially Observable Markov Decision Processes

Section 3.1 introduced a fully observable MDP, meaning the current state is fully observable (i.e., the current state is known). This section introduces Partially Observable Markov Decision Process

(POMDP), an extension of MDP, where the current state is not precisely known (e.g., due to noisy sensors) and uses a belief of the current state.

A POMDP is defined as a tuple $\langle S, A, p(s'|s,a), r(s,a,s'), \gamma, \Omega, O\rangle$, where the first 5 elements represent the underlying MDP (see Section 3.1 for details) and

- Ω is the set of observations with $o \in \Omega$
- O is the set of conditional observation probabilities $p(o|s', a)$

The agent in some state s takes some action a, causing the agent to transition to some next state s' with probability $p(s'|s,a)$. The agent also receives an observation o on the next state s' with probability $p(o|s', a)$. Decisions are based on a history of actions and observations or tracking the belief state. A *belief state* b is a probability distribution over the underlying MDP states, where a probability $b(s)$ is attributed to being in state s. A MDP policy maps states to actions whereas a POMDP policy maps belief states to actions.

The benefit of POMDPs is that it provides a means of addressing uncertainty. Unfortunately, exact solutions for finding the optimal policy are computationally expensive and can be intractable to solve. Instead approximation methods can generally solve POMDPs well.

3.10 QMDP

QMDP is method of solving POMDPs by using the solution of the underlying MDP to determine an approximate optimal policy [35] and is said to be almost as computationally efficient as MDPs [36]. It computes upper bounds of the exact optimal value function, which can be used to determine actions corresponding to the highest value. A QMDP does not require use an observation model and instead incorporates uncertainty by calculating beliefs used to determine a policy defined over belief states assuming all state uncertainty disappears on subsequent steps:

$$\pi(b) = \arg\max_a \sum_s b(s) Q_{MDP}(s, a)$$

where $Q_{MDP}(s, a)$ uses the underlying MDP's value function $V(s')$ described in Section 3.6 and Section 3.7:

$$Q_{MDP}(s, a) = \sum_{s'} p(s'|s, a)[r(s, a, s') + \gamma V(s')]$$

The main advantages of QMDPs are that they are simple and provide fast approximations to small domained POMDP problems. Major limitations to this approach is that performance degrades and can become infeasible to compute for large POMDPs. Also, it does not take advantage of information gathering actions that could help identify the partially observable environment.

4

Algorithm Design

Within this chapter, contains the formulation of the Markov Decision Process (MDP) and Partially Observable Markov Decision Process (POMDP) used to solve the problem of navigating uncertain dynamic obstacles. The idea for the algorithm was to have the capability of determining the general policies offline to some specified goal position with limited or possibly inaccurate prior information of the global environment. The intention was for the procedure to be general enough so they could be later transferred to the robot. Then during online execution, the agent would use surrounding local information, that could contain new information, to determine the policy to execute. Since the size of the local world would be quite small, providing a solution for real time operation would be possible. The presented algorithm was designed for a holonomic (i.e., omnidirectional) robot.

There are two main phases to the implemented approach and can be visualized in Figure 4.0.1:

1. *Offline*: A known prior global world in the form of an occupancy grid is trained using policy iteration to determine the optimal global policies to a given goal location.
2. *Online*: Leveraging data from the offline training phase, the agent uses local information to determine if the world has changed. If the world has not changed and no obstacles are present, it follows the global policies determined in the offline phase. If the world has changed or an obstacle is detected, the agent performs a local procedure to determine the new optimal local policy to execute that avoids potential dynamic obstacles using QMDP. In parallel, global policy iteration is performed with the updated information. Due to the small size of the local region, the local procedure is able to determine the best action to execute before the global policy iteration has completed. Then the agent executes the action and the process repeats until the agent has reached the goal or a collision occurred.

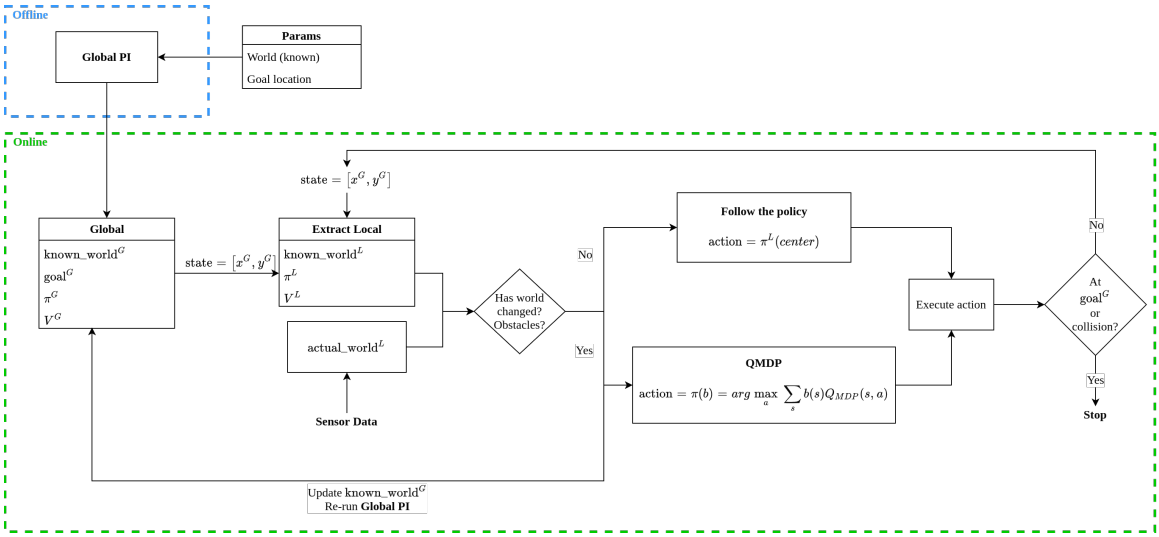


Figure 4.0.1: A high-level overview of the algorithm where the blue box is the offline process and the green is the online.

4.1 Offline: Determining General Behaviors and Values

This section describes the algorithm used to extract general global policies and the value function from known or possibly inaccurate prior knowledge of the environment. Of which will be used in local online procedure (discussed in Section 4.2) to incorporate the gained information by defining local goal states and speed convergence when computing new local policies.

4.1.1 Global Policy Iteration (G-PI)

Discussed in this section, is the formulation of the MDP for navigation in a occupancy grid (or grid world) setting. This is solved using Global Policy Iteration (G-PI), which is a dynamic programming technique, to determine the general behaviors and the value function which will be used during the offline process.

States

The environment is represented as a two-dimensional grid world. The global states S^G consist of all obstacle free $s^G = (x^G, y^G)$ grid cells.

Actions

Since the algorithm is designed for a omnidirectional robot, there are 8 available actions that will take the agent to the next grid cell. The actions (depicted in Figure 4.1.1) are North (N or $\uparrow$), West (W or $\leftarrow$), East (E or $\rightarrow$), South (S or $\downarrow$), North-West (NW or $\nwarrow$), North-East (NE or $\nearrow$), South-West (SW or $\swarrow$), and South-East (SE or $\searrow$). It is important to note the order of the actions, they are as such so when determining the policy, if there are multiple actions that yield the maximum value then non-diagonal actions are selected first.

$a_5 = \text{NW}$ $\nwarrow$	$a_1 = \text{N}$ $\uparrow$	$a_6 = \text{NE}$ $\nearrow$
$a_2 = \text{W}$ $\leftarrow$		$a_3 = \text{E}$ $\rightarrow$
$a_7 = \text{SW}$ $\swarrow$	$a_4 = \text{S}$ $\downarrow$	$a_8 = \text{SE}$ $\searrow$

Figure 4.1.1: The order of available actions: N, W, E, S, NW, NE, SW, SE.

Rewards

The global rewards are defined as follows,

$$R^G(s^G) = \begin{cases} -50 & \text{if obstacle} \\ -10 & \text{if surrounding obstacle} \\ 50 & \text{if goal} \\ -1 & \text{otherwise} \end{cases} \tag{4.1}$$

Rewards surrounding obstacles in the immediate next cells are -10 to encourage the agent to stay further away from obstacles. All other free space is given a reward of -1 to motivate the agent to reach the goal quickly.

Transition Model

The state-transition model $p(s'|s, a)$ used for calculating the value function (and policies) can be modeled and updated online during execution. Initially, all possible state-action transitions are unity. The possible states (shown in Figure 4.1.2) for determining the transition probabilities are the immediate cells that surround the agent as well as the cell that the agent occupies (i.e., the center cell).

Online, for each action, the agent tracks the number of times the action took the agent to the next state. Then when estimating the value function, for each action and each possible next state, the transition probability equals the number of times the action took the agent to that particular state divided by the sum of all the action's next states. For each action, all transitions to each state sum to one. For example, if a_5 took the agent to st_1 twice, st_4 once, and st_5 once. Then the transition

probabilities for a_5 would be $st_1 = \frac{2}{4} = \frac{1}{2}$, $st_4 = \frac{1}{4}$, and $st_5 = \frac{1}{4}$.

st_1	st_2	st_3
st_4	st_5	st_6
st_7	st_8	st_9

Figure 4.1.2: The possible next states st_i for all actions.

4.2 Online: Leverage Prior Information and Motion Uncertainty

The global world is given and known prior whereas the local world may contain new knowledge of the world not seen during offline training. During online execution, the agent uses newly gathered surrounding information from sensors such as a 2D LIDAR and compares the neighboring region to the global world (at the local context). If no change or obstacles are detected, the agent follows the global policy. If a change or obstacles have been detected, then the known global world is updated and G-PI is executed in parallel of local procedures, which determines the new best policy. Two local procedures Local Policy Iteration (L-PI) and QMDP were designed and are described in Section 4.2.1 and Section 4.2.2 respectively.

4.2.1 Local Policy Iteration (L-PI)

When obstacles are detected, the problem becomes a POMDP because it is uncertain where the obstacle will move to at the next time step. This is solved by using an approximator, QMDP, which finds a solution by solving the underlying MDP and will be detailed in Section 4.2.2. This section describes the solution to solving the underlying MDP using L-PI. The environment is represented as a small $m \times m$ grid world where the robot is always in the center. This process provides a $\frac{m+1}{2}$-step look ahead path (i.e., short term plan) to a specified local goal.

States

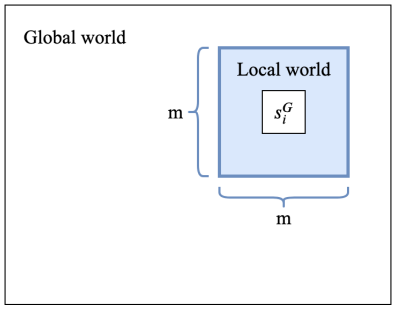


Figure 4.2.1: When the agent is in the global state s_i^G, the equivalent local world is shaded in blue and $s_i^G \equiv s_{center}^L$.

Similarly to the global policy iteration states described in Section 4.1.1, the local states S^L are all the obstacle free $s^L = (x^L, y^L)$ local grid cells. The size of the local grid world is square $m + 2 \times m + 2$ where m is an odd value so the agent is located in the very center. The center of the local world represents the same location the agent occupies in the global world. The same goes for the surrounding local region, which represents the same area in the global world, which will be termed local context. This is illustrated in Figure 4.2.1. The additional 2 rows and columns serve as a protective outer border since it is not known what lies beyond the $m \times m$ grid.

Actions

The local actions are the same as the global actions described in Section 4.1.1.

Rewards

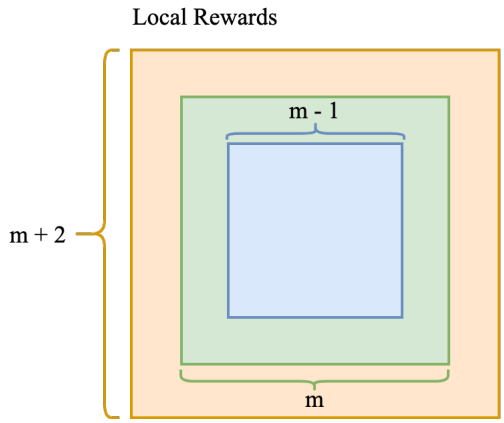


Figure 4.2.2: The local rewards structure: (Orange) has -5 rewards. (Green) contains a reward of 30 where the maximum extracted local value function are located and follows the same global rewards defined in Equation 4.1 otherwise. (Blue) uses the same global rewards defined in Equation 4.1.

The local rewards are similar to the global rewards described in Section 4.1.1 with some minor adjustments. When the agent is at the global state s_i^G, the agent extracts the equivalent $m \times m$ value function. The value function represents the expected long-term received rewards for each state. The location(s) of the maximum of the outer $m \times m$ rows and columns (i.e., 2^{nd} and m^{th} rows and columns) that are not occupied by a local obstacle and are not zero are given a local goal of 30. The the other outer $m \times m$ and inner $m - 1 \times m - 1$ follows the same global rewards as defined in Equation 4.1. The outer protective border (i.e., 1^{st} and $(m + 2)^{th}$ rows and columns) are given a -5 reward because it is unknown. A visual is shown in Figure 4.2.2 and the local reward function

is as follows,

$$R^L(s^L, V_\pi^L) = \begin{cases} -5 & \text{if } s^L \in (m+2) \text{ outer protective border} \\ -50 & \text{if obstacle } s^L \in \text{inner } m \times m \\ -10 & \text{if surrounding obstacle } s^L \in \text{inner } m \times m \\ 50 & \text{if } s^L == s^G_{goal} \\ 30 & \text{if } s^L_j == \max_j(V^L_\pi) \text{ of outer } m \times m \\ -1 & \text{otherwise} \end{cases} \tag{4.2}$$

Transition Model

The same transition model from Section 4.1.1 is utilized.

4.2.2 Local QMDP

The previous sections describing G-PI and L-PI determine paths around obstacles but only works well with static obstacles under a MDP. When dynamic obstacles are considered, the problem becomes a POMDP since the motion of the obstacles are not precisely known and uncertain. This section describes the QMDP solution to solving the POMDP, using beliefs of the next possible obstacle states to determine actions that avoid potential dynamic obstacles. The QMDP implementation considers one step look ahead and the motion of all locally detected obstacles (static or dynamic) are assumed random. The formulation uses L-PI to solve each underlying MDP for all potential next obstacle states in conjunction with its belief to determine the policy.

States

Similarly to the states described in Section 4.2.1, the size of the local world is a square $m \times m$ where m is an odd value so that the agent is located in the center of the grid world. In L-PI the states were all the obstacle free local grid cells whereas in QMDP the states S^Q are all the possible combinations the detected obstacle could be in at the next time step. Since only one step look ahead is being considered, for any given obstacle, there are 9 possible states the obstacle could end up in (i.e., in the cells that it currently occupies and the surrounding immediate next cells). This means for n number of obstacles, there are 9^n possible states except for when the next possible positions of the obstacles are out of the local area. Using a pessimistic approach, it is assumed that the obstacle will not leave the local frame. Therefore, for each obstacle i, there are p_i positions that could be out of frame making the number of possible states $s^Q \in S^Q$ be $\prod_{i=1}^{n}(9 - p_i)$. An example is illustrated in Figure 4.2.3.

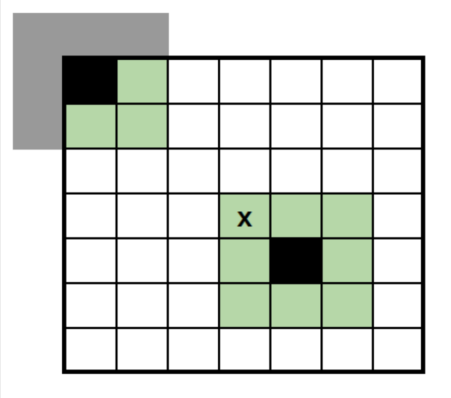
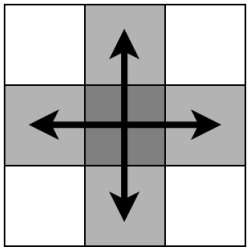


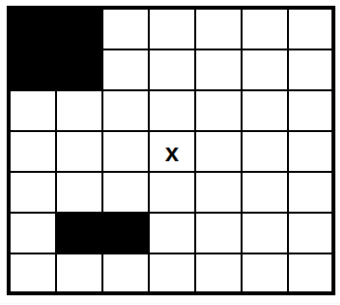
Figure 4.2.3: The local world is 7×7 containing 2 obstacles (black) and the agent's location is in the center (×). The top left obstacle has 5 positions that are out of the local area (gray), while the other obstacle has 0. Green represents the possible next states for each obstacle. The number of possible obstacle combinations (i.e., QMDP states) is (9-5)(9−0)=36.

The number of obstacles n is determined based on their connectivity of each cell. If their edges

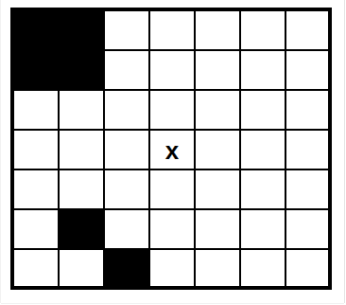
touch along the horizontal and vertical direction then it is considered to be part of the obstacle (Figure 4.2.4a). If their corners touch and not their edges then they are considered two separate obstacles (example shown in Figure 4.2.4c).



(a) Obstacle connectivity



(b) 2 obstacles



(c) 3 obstacles

Figure 4.2.4: **(a)** Obstacle connectivity and **(b)-(c)** local world examples.

Beliefs

The local world was treated as an image matrix, where ones represented an obstacle and zeros represented free space. The motion of all locally detected obstacles was assumed random following a Gaussian distribution. Using a 2D Gaussian smoothing kernel with standard deviation of 0.5 on the local world to blur the obstacles such that the highest values were the current location of the obstacle and the lower values in the surrounding cells. For each blurred obstacle, the values summed to 1. Using a pessimistic approach, it is assumed that all locally detected obstacles (static

or dynamic) will move and unlikely stay in the same location. To determine the beliefs of obstacle states at the next time step, required inverting the Gaussian so that for each obstacle the current location of the obstacle had the lowest value, the surrounding cells were higher, and each summed to 1. These values were used to determine the probability $b(s)$. If there were multiple obstacles, then the belief was a product of beliefs corresponding to each obstacle state:

$$b(s_j^Q) = \prod_{i=1}^{n} u_{i,s_j^Q} \tag{4.3}$$

where j is the index of a possible state $s_j^Q \in S^Q$ and n is the number of obstacles. An illustrative example is shown in Figure 4.2.5 (continuing on example Figure 4.2.3).

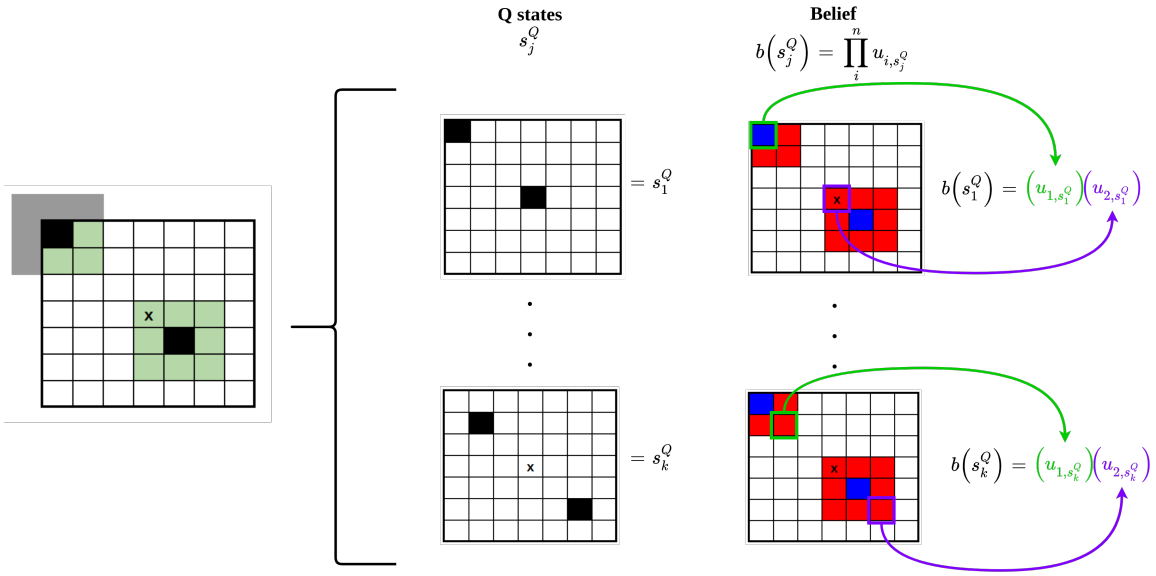


Figure 4.2.5: Continuing on the example of a 7×7 local world containing two obstacles. Each Q state s_j^Q is a combination the obstacle could be in at the next time step alongside the corresponding belief $b(s_j^Q)$.

These potential next states become the beliefs used to determine the next action:

$$\pi(b) = \arg\max_{a} \sum_{s^Q} b(s^Q) Q_{MDP}(s_{center}, a) \tag{4.4}$$

where s_{center} is the center of the local $m \times m$ world and $Q_{MDP}(s_{center}, a)$ for each state s^Q is solved by using value function V_{MDP} from L-PI:

$$Q_{MDP}(s_{center}, a) = \sum_{s'} p(s'|s_{center}, a)[r(s_{center}, a, s') + \gamma V_{MDP}(s')]$$

Actions

The local QMDP actions are the same as the global actions detailed in Section 4.1.1.

Rewards

For each possible state $s^Q \in S^Q$ the same rewards described in Figure 4.2.1 are utilized.

Transition Model

The same transition model from Section 4.1.1 is used.

4.3 Combined Policy Iteration & QMDP

Thus far, the global solution in Section 4.1 and local solutions in Section 4.2 have been described. This section details how these solutions are combined and utilized for online execution. Both the *Offline* and *Online* procedures use policy iteration, which is specified in algorithm 1. The *Offline* procedure (algorithm 2), requires the agent's goal location s^G_{goal} and what is known about the global

world $known\_world^G$. If nothing is known about the world, then $known\_world^G$ is empty with the size of the environment the agent will be operating in. This procedure determines the optimal value function V_*^G and the optimal policies π_*^G, which are used in the *Online* procedure. The discount factor γ^G is 0.9 to put more weight towards long-term rewards.

The *Online* procedure, formalized in algorithm 4, is executed at run time and operating in the global world. The agent uses locally collected data (e.g., 2D LIDAR) and compares it to $known\_world^G$ in the local context. If the worlds match and no obstacles are detected, then the agent executes the global policy π_* at the current state s_{curr}^G. If a change is detected and no obstacles are present, L-PI is executed leveraging the locally extracted policies from global to speed up computation. When obstacles are detected, then the agent extracts the local equivalent from the global value function V_*^G and policies π_*^G to be used in QMDP (algorithm 3). In QMDP (introduced in Section 4.2.2), the beliefs for each detected obstacle is determined by inverting the response of the Gaussian filter such that each obstacle sums to 1. The set of possible Q states S^Q are all the combinations the obstacle could be in at the next time step. Then for each Q state s^Q, L-PI determines the value function V_{MDP}, using the local policies extracted from global to quicken convergence. Since this is a short-term planner, the local discount factor γ^L is 0.4. Then V_{MDP} is used to calculate Q_{MDP} from the center state of the $m \times m$ local world. Only the center state is being considered because we are only interested in determining the best action to execute from the center of the locally detected world (i.e., s_{curr}^G). The policy which yields the maximum action value from the center state is executed. Then G-PI, which can be done in parallel, is re-executed on the updated $known\_world^G$ to renew the global value function V_*^G and global policies π_*^G. This *Online* process is repeated until the agent has reached the goal, has collided with an obstacle, or has reached the maximum number of steps

Algorithm 1: Policy Iteration

Function *policy_iteration*$(S, r, V_\pi, \pi, \gamma, max\_iterations)$**:**

  for *iter* $= 1$ **to** *max_iterations* **do**

    `// 1. policy evaluation`

    Loop

      $V_{temp} = V_\pi$

      foreach $s \in S$ **do**

        $V_\pi(s) = \sum_{s'} p(s'|s, \pi(s))[r(s, \pi(s), s') + \gamma V_\pi(s')]$ `// following policy`

    until $V_{temp} == V_\pi$;

    `// 2. policy improvement`

    $\pi_{temp} = \pi$

    foreach $s \in S$ **do**

      $\pi(s) = \arg\max_a \sum_{s'} p(s'|s, a)[r(s, a, s') + \gamma V_\pi(s')]$

    if $\pi_{temp} == \pi$ **then**

      break `// break iterations loop`

  Return: $[V_* \approx V_\pi,\ \pi_* \approx \pi]$

Algorithm 2: Global Policy Iteration (*Offline*)

Input: $known\_world^G$

s^G_{goal}

Initialize: $S^G =$ all free space $\in known\_world^G$

$V_\pi(s) \in \mathbb{R}$ and $\pi(s) \in A$ arbitrarily $\forall\ s \in S^G$

$r(s) = R^G(s) \quad \forall s \in S^G$ `// Equation 4.1`

$\gamma^G = 0.9$

$max\_iterations = 500$

Result: $[V^G_*, \pi^G_*] = policy\_iteration(S^G, r, V_\pi, \pi, \gamma^G, max\_iterations)$

that was allowed.

Algorithm 3: Local QMDP

Function $QMDP(world^L, V_\pi^L, \pi^L, \gamma^L, max\_iterations)$**:**

- $u = beliefs(world^L)$ `// inverted Gaussian filter on` $world^L$
- $S^Q =$ all possible world combinations of obstacles $\in world^L$ at next time step
- $Q = zeros(3)$ `//` 3×3 `matrix representing each` $a \in A$
- **foreach** $s^Q \in S^Q$ **do**
  - $S^L =$ all free space in s^Q
  - $r(s^L) = R^L(s^L, V_\pi^L) \quad \forall s^L \in S^L$ `// Equation 4.2`
  - $V_\pi = zeros(size(s^Q))$
  - $\pi = \pi^L$ `// using extracted local policies to speed up convergence`
  - $[V_{MDP}, \sim] = policy\_iteration(S^L, r, V_\pi, \pi, \gamma^L, max\_iterations)$
  - $b(s^Q) = \prod_{i=1}^{n} u_{i,s^Q}$ `// Equation 4.3`
  - **foreach** $a \in A$ **do**
    - `/* where` s_{center} `is the center of` $world^L$ `and` s' `is the next state from` s_{center} `when executing action` a `*/`
    - $Q_{MDP}(s_{center}, a) = \sum_{s'} p(s'|s_{center}, a)[r(s_{center}, a, s') + \gamma^L V_{MDP}(s')]$
    - $Q(s_{center}, a)$ += $b(s^Q) Q_{MDP}(s_{center}, a)$
- **Return:** $\pi = \arg\max_a Q(s_{center}, a)$

Algorithm 4: *Online* procedure

Input: $known\_world^G$, s^G_{goal}, V^G_*, π^G_*, $max\_iterations$

Initialize: max_steps = maximum number of steps to reach goal
$\gamma^L = 0.4$
s^G_{curr} = current $[x, y]$ global position

1 **for** $step = 1$ **to** max_steps **do**
2 **if** $s^G_{curr} == s^G_{goal}$ **then**
  Return: *success*
3 **if** $s^G_{curr} == obstacle$ **then**
  Return: *failure*
4 $actual\_world^L$ = actual local world `// (e.g., from sensors)`
5 **if** $actual\_world^L == known\_world^G(local\_context)$ **then**
6 $a = {\pi_*}^G(s^G_{curr})$ `// following global policy`
7 **else**
8 $known\_world^G(local\_context) = actual\_world^L$ `// update known global world`
9 $r(s) = R^G(s) \quad \forall s \in S^G$ `// update global rewards (Equation 4.1)`

  `/* Extract global value function and policies for local context */`
10 $V^L_\pi = V^G_*(local\_context)$
11 $\pi^L = \pi^G_*(local\_context)$

12 $a = QMDP(actual\_world^L, V^L_\pi, \pi^L, \gamma^L, max\_iterations)$

  `/* Re-run global policy iteration (can be done in parallel) */`
13 $\pi^G_*(s^G_{curr}) = a$ `// update policy to speed up convergence`
14 $[V^G_*, \pi^G_*] = policy\_iteration(S^G, r, V_\pi, \pi, \gamma^G, max\_iterations)$

15 s^G_{curr} += a `// execute action and transition to next state`

Return: *failure* `// did not reach` s^G_{goal} `in` max_steps

5

Evaluation

This chapter describes how the proposed algorithm (defined in chapter 4) was evaluated. Section 5.1 details the setup for simulation and for hardware. Originally, the simulation was used to verify proof of concept and experiments were going to be conducted on the physical robot. In response to the COVID-19 pandemic, the university closed down all on-campus operations and unfortunately the hardware implementation was halted. The simulation setup is specified in Section 5.1.1 followed by the hardware setup in Section 5.1.2, with a brief explanation of intended experiments. Lastly, the chapter ends with a discussion on results from the simulation in Section 5.2.

5.1 Evaluation Setup

5.1.1 Simulation Setup

To validate and test the proposed algorithm, a simulation environment was setup in MATLAB. The agent is operating in a 2D grid representation of the world. Each world used for evaluation was 100×100 binary matrix where zeros represented free space and ones represented obstacles. In each test, a known world with or without static obstacles is used for offline training (all with a static obstacle border to keep the agent in the environment) outlined in algorithm 2. This known world as well as the resultant value function and policies are used in the online procedure. Before the online procedure begins, a new matrix depicting the actual world containing static and/or dynamic obstacles not seen in training is created. During execution of the online procedure, the agent is at some global $[x, y]$ (i.e., state) location in the grid. From there, the equivalent 7×7 local matrix (with the agent being in the center) is extracted from the actual world. Meaning, at some global $[x, y]$ position, 3 rows above and below as well as 3 columns to the left and right are extracted. This local actual world represents new information that could be obtained through sensors such as a 2D

LIDAR. The same extraction is performed on the known world and compared with the actual. This was how the known and actual information used in the formulation presented in algorithm 4. The worlds used for evaluation are presented in Section 5.2.

5.1.2 Hardware Setup

A modified TurtleBot3 Burger from ROBOTIS [37] was used to implement the experiments. The original TurtleBot3 was modified from a differential drive train to a holonomic with 3 Mecanum wheels. The modified omnidirectional Turtlebot3 (OmniBot), designed by Castle [38], was modified further. The OpenCR board was removed and all low-level motor commands came from the onboard Raspberry Pi 3 (RPi). The kinematic equations to drive OmniBot derived from [39] are as follows:

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \frac{1}{r} \begin{bmatrix} -d & 1 & 0 \\ -d & -\frac{1}{2} & -\sin(\frac{\pi}{3}) \\ -d & -\frac{1}{2} & \sin(\frac{\pi}{3}) \end{bmatrix} \begin{bmatrix} \omega_z \\ v_x \\ v_y \end{bmatrix}$$

where u_1, u_2, and u_3 are the rotational velocity commands sent to the motors, r is the radius of the Mecanum wheels, d is the distance from the center of OmniBot to each wheel (each wheel evenly spaced 120° from each other), ω_z is the rotational velocity, and v_x and v_y are the translational velocities in the x and y directions.

OmniBot was designed such that RPi would only handle the low level actuator commands and monitor battery status. All other computation was performed on a remote desktop computer and control actions were sent wirelessly to OmniBot. The OmniBot software utilized the Robot Operating System (ROS), a common robotics framework which handles communication between

different executable programs (i.e., nodes). Global localization came from a Vicon motion capture system, which uses retro-reflective markers on top of the physical robot to determine the full pose ($x, y, z, roll, pitch,$ and yaw) of the robot in 3D space. OmniBot's operating environment was on an air hockey table equipped with 6 Vicon cameras, an overhead projector, and desktop computer (see [38] for further details).

To execute the proposed approach in this thesis, several nodes were created. A control node, given a relative goal position would use the Vicon's global positioning data to move the OmniBot to the relative position. Relative goal positions were given, instead of global, since a robot operating in the real world (without a motion capture system) would most likely be given relative goal commands. A policy iteration node advertised two action services, one for G-PI and the other for L-PI. G-PI was implemented as a ROS action server because it does not block the program that called it. This allows for G-PI to be run in parallel so that OmniBot can continuing operating. The operable space on top of the air hockey table was 100 $\times$ 200cm and OmniBot was 20 $\times$ 20cm. Initially, it was thought that the grid sizes (of the grid world) should be the same size as OmniBot. As such, the global grid world used in G-PI was 5 rows by 10 columns (i.e., 50 states) and each grid size was 20cm^2. The local world size used in L-PI was 3 by 3. Finally, an executive node, which would interact with all implemented nodes and control OmniBot autonomously.

The state transitions were monitored and updated in the executive node. The very first time OmniBot was executed, all state-action transitions were unity. During operation, when a new state-action transition was detected, the transitions were updated and saved so that these transitions could be used again.

In these developmental stages, virtual obstacles were implemented. Similarly to the simulation set up described in Section 5.1.1, there was a known world and an actual world. For the hardware

case, the known world was always empty and the actual world contained new obstacles not known to OmniBot until detected locally (i.e., in the 3×3 local world).

At executive start up, OmniBot would be at some arbitrary starting location and G-PI was executed with some arbitrary goal. Once G-PI completed, the equivalent 3×3 local world was extracted from the actual world and compared with the known. When a change was detected, the world was updated and L-PI was executed. When L-PI completed, the global policy at the current was updated with the center of the local policies. While G-PI was being re-ran, the policy to execute was sent to the control node and the robot moved to the next state. This process was repeated until OmniBot reached the specified goal state. Figure 5.1.1 illustrates the OmniBot setup on the air hockey table configuration.

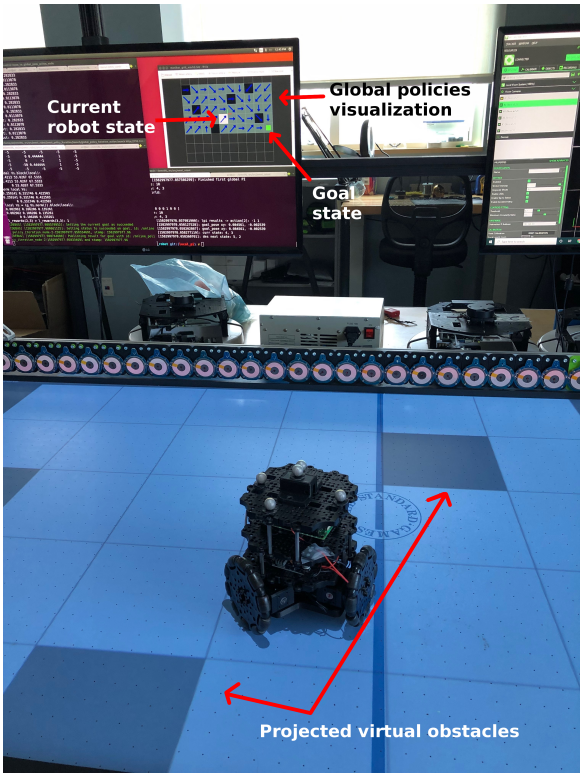


Figure 5.1.1: OmniBot operating on the custom air hockey table with projected obstacles.

Preliminary tests showed that OmniBot was able to reach intended goal states at any starting state but when diagonal actions were executed next to a virtual obstacle the robot would virtually run into the obstacle while moving to the next state. This lead to adding the -10 penalty around obstacles to the reward function to encourage the agent to stay further away from hazards but the resolution of the global and local grid worlds were not fine enough to properly navigate and account for dynamic obstacles.

Before the university closed laboratory operations in response to the COVID-19 pandemic, the next steps were to refine the resolutions to be 10×20 (i.e., 200 states) globally and 5×5 locally. Then the QMDP portion was to be implemented. Once tested and verified, the final step was to build 2 more OmniBots, implement the proposed approach on each robot, then evaluate their independent responses in a multi-agent scenario. After the university closed, since the OmniBots heavily relied on the Vicon motion capture system for a ground truth state estimate, the implementation and experiments could not be finished or conducted further.

5.2 Simulation Evaluation

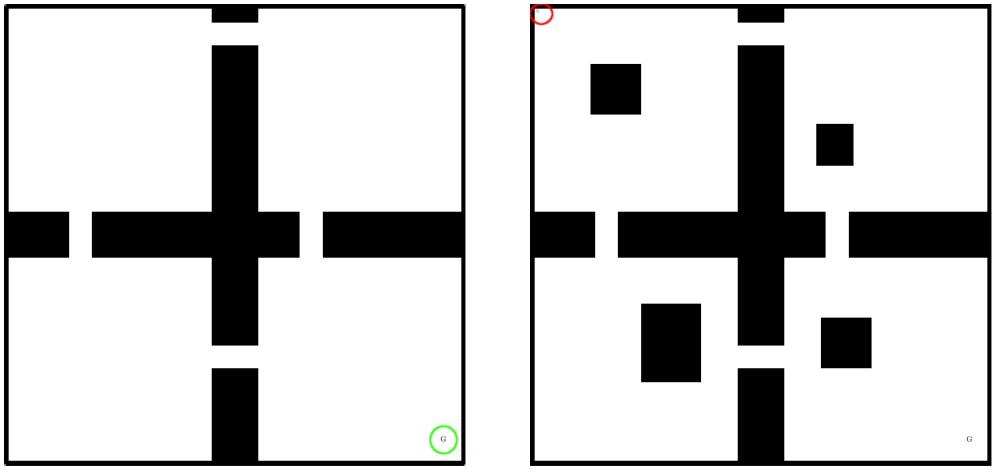
This section describes the evaluation procedures used to measure performance. All global worlds, known and actual, are of size 100×100 and the local are 7×7. This 7×7 local environment provides a 3-step look ahead plan for newly obtained information (e.g., obstacles). The actual worlds contained a number of dynamic obstacles not seen in the offline training process, which move randomly based on a Gaussian distribution.

There are two presented case studies used for assessment, each containing 3 evaluations. The first, is a simple world where the environment is divided into 4 quadrants. The second, embodies a maze world. For each evaluation, 100 MC-sims (i.e., trials) were conducted, comparing results

from both the presented algorithm introduced in Section 4.3 termed Policy Iteration and QMDP (PI+QMDP) and pure Policy Iteration only (PI only). Failures are defined as collisions and when collisions occurred, the Monte Carlo (MC) trial was terminated. The results presented include the total number of dynamic obstacles (# Obs.) in the environment, number of failures (# Fail.), the average and median steps taken in the said failures (Avg. FS and Med. FS respectively), as well as the average and median successful steps (Avg. SS and Med. SS respectively) taken to the goal. What is also presented for all evaluations are the frequency of which the agent followed a trajectory and frequency of where collisions occurred. In all simulations, the goal state was set to $s^G_{goal} = [95, 95]$ and the state where the agent started was $s = [2, 2]$.

5.2.1 Simple World Results

Evaluation 1



(a) Known world used for offline training, where the green circle indicates the specified goal.

(b) Actual world used for online execution with 4 dynamic obstacles. The red circle (top left) indicates the agent's starting position.

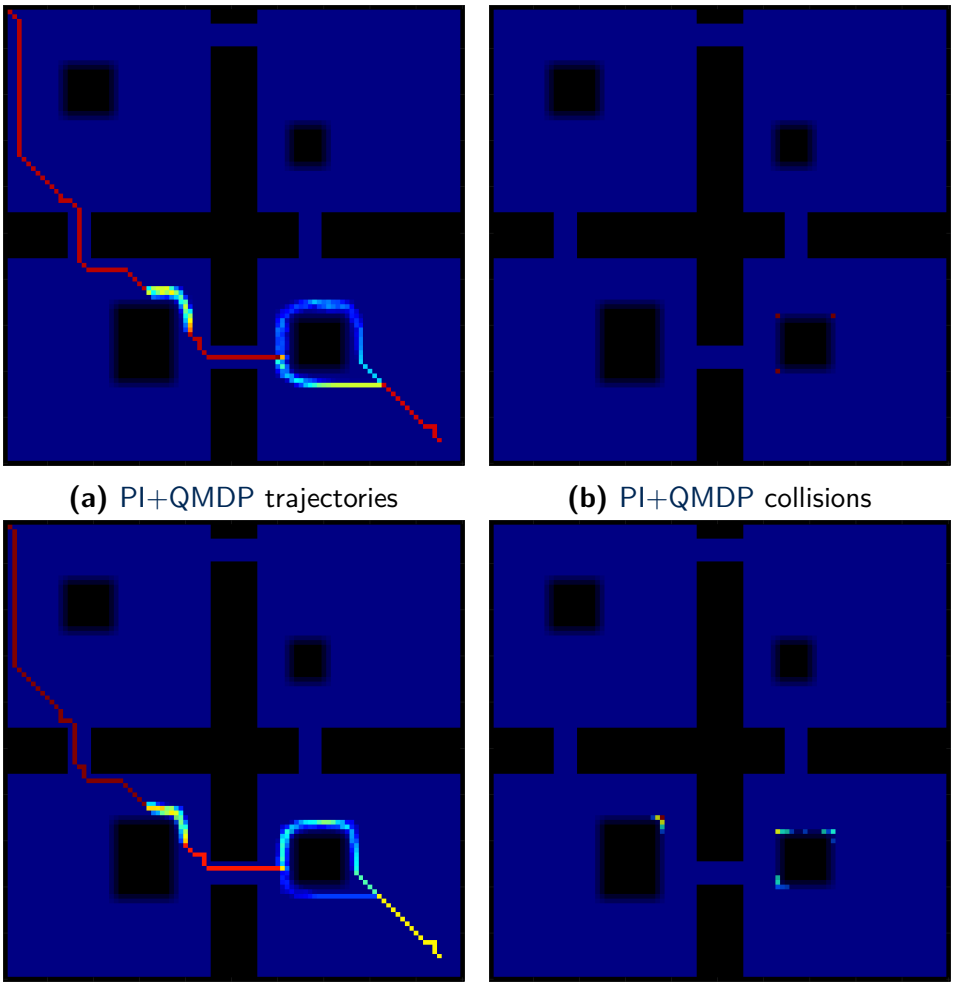
Figure 5.2.1: Simple World - Evaluation 1: Global Worlds

For this scenario, the global world used for offline training (i.e., known world) is displayed in Figure 5.2.1a and the specified goal state $s^G_{goal} = [95, 95]$. Online, the agent started at state $s = [2, 2]$ in the actual world (shown in Figure 5.2.1b) which contained 4 new obstacles that were not seen in training and were dynamic. The results presented in Table 5.2.1, show that PI+QMDP finished 97% of the simulations successfully whereas PI only successfully reached the goal in only 63% of the trials. In PI+QMDP, there were 3 collisions, which all occurred at the obstacle closest to the goal (Figure 5.2.2b). These collisions took place because the obstacle had jumped two spaces over from the previous time step which the PI+QMDP formulation does not consider, it only consid-

ers obstacles potentially moving in the immediate surrounding cells (i.e., one space). Figure 5.2.2 compare the frequency of the trajectories taken by the agent when using PI+QMDP versus PI only as well as locations collisions occurred. In these figures, black areas are the obstacles that appear in the actual world and the 4 dynamic obstacles are blurred to illustrate their movement in all 100 simulations. Areas that are closer to dark blue are closer to zero and red means higher frequency. Although there are 4 dynamic obstacles, during the online procedure the agent only encounters 2 (the 2 in the bottom rooms). When looking at the PI+QMDP trajectories (Figure 5.2.2a) through narrow passages (e.g., leaving the upper left hand room and into the lower left), it can be seen that the agent prefers to be in the center of the two static obstacles that are to the left and right. Comparatively, the agent operating under PI only navigates much closer. Also in PI+QMDP, when navigating around the closest dynamic obstacle to the goal, the agent prefers to go under the obstacle whereas in PI only prefers over. Going under, reduces the amount of steps to reach the goal, especially since the obstacle is moving. For example, when the agent goes above and the obstacle moves up, then the agent will also need to move up to avoid colliding with the obstacle, moving further away from the goal.

4 Obs.	# Fail.	Avg. FS	Med. FS	Avg. SS	Med. SS
PI+QMDP	3	124.67	120	157.76	152
PI only	37	104.97	114	160.46	163

Table 5.2.1: Simple World - Evaluation 1: Results



(a) PI+QMDP trajectories **(b)** PI+QMDP collisions

(c) PI only trajectories **(d)** PI only collisions

Figure 5.2.2: Simple World - Evaluation 1: PI+QMDP versus PI only trajectories and collisions.

Evaluation 2

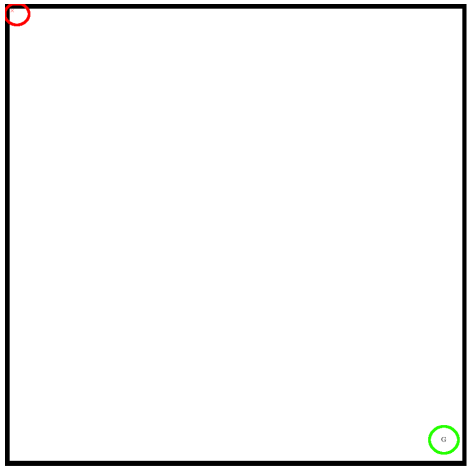
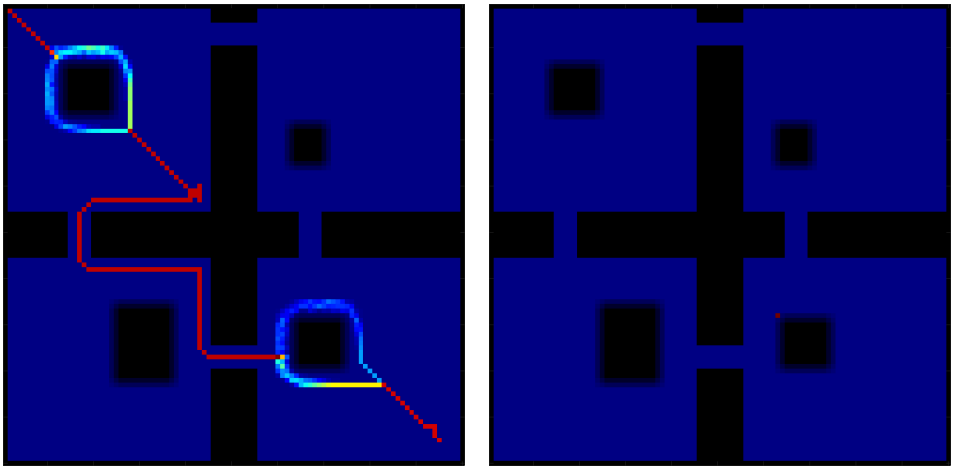


Figure 5.2.3: Simple World - Evaluation 2: Known World

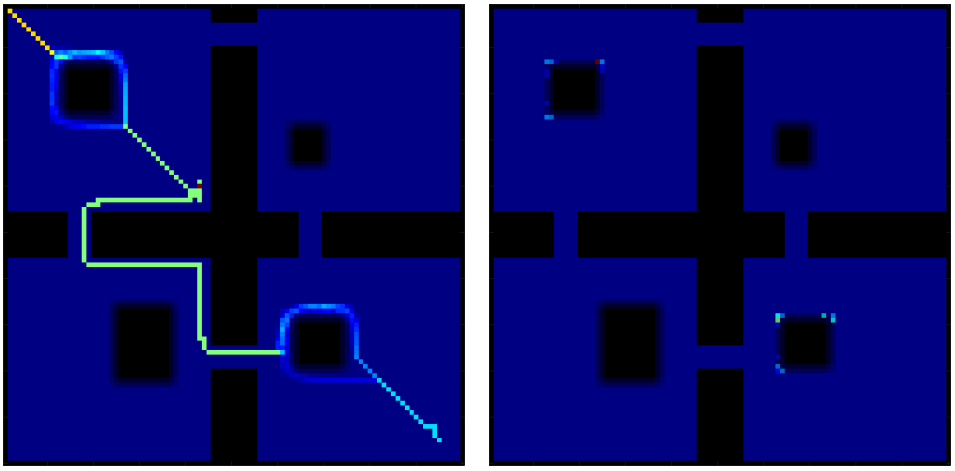
In this evaluation, the same experiment with the same number of dynamic obstacles was conducted except the prior known world used for training was unknown (Figure 5.2.3). The actual world is the same as seen in Figure 5.2.1b. In this situation, the agent had no prior knowledge of the operating environment other than the size. The produced global value function, in the offline procedure, encoded enough information to propel the agent in both PI+QMDP and PI only toward the goal but required roughly 50 more steps to get there. The results shown in Table 5.2.2 show that the PI+QMDP agent had 98% successful simulations and PI only successfully completed 52%. Both trajectories in Figure 5.2.4a and Figure 5.2.4c demonstrate a wall following strategy and in the case of PI+QMDP, the agent chooses to stay slightly further away from the walls. What is interesting is the updated known world after each online execution trial (an example is shown on the last PI+QMDP MC-sims episode). It confirms that the global value function is expressive enough to drive the agent to the goal with limited information. The final updated worlds after each MC-sims in both PI+QMDP and PI only are roughly the same.

4 Obs.	# Fail.	Avg. FS	Med. FS	Avg. SS	Med. SS
PI+QMDP	2	169	169	207.86	203
PI only	48	100.52	102.50	214.02	215

Table 5.2.2: Simple World - Evaluation 2: Results



(a) PI+QMDP trajectories **(b)** PI+QMDP collisions



(c) PI only trajectories **(d)** PI only collisions

Figure 5.2.4: Simple World - Evaluation 2: PI+QMDP versus PI only trajectories and collisions.

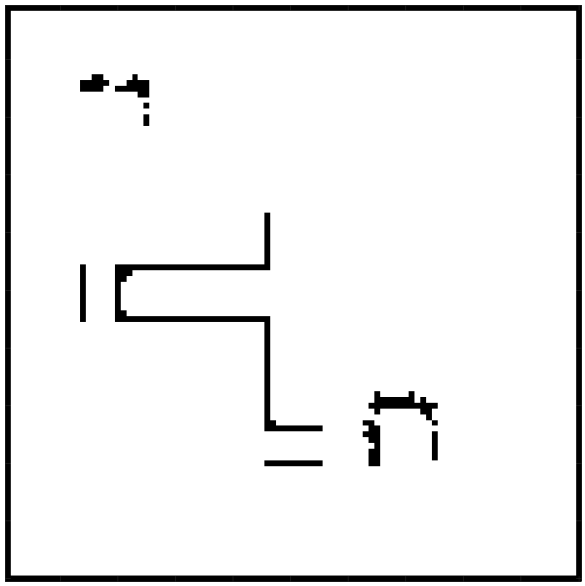


Figure 5.2.5: Simple World - Evaluation 2: PI+QMDP known world in last MC-sims trial.

Evaluation 3

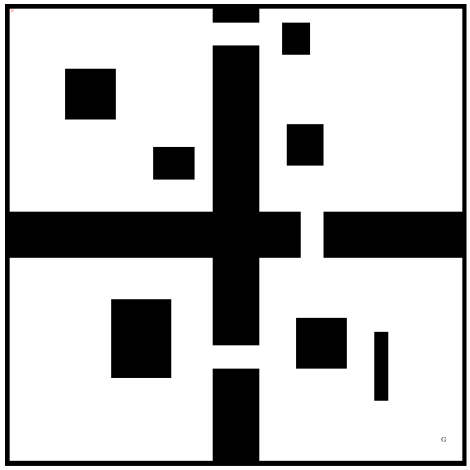
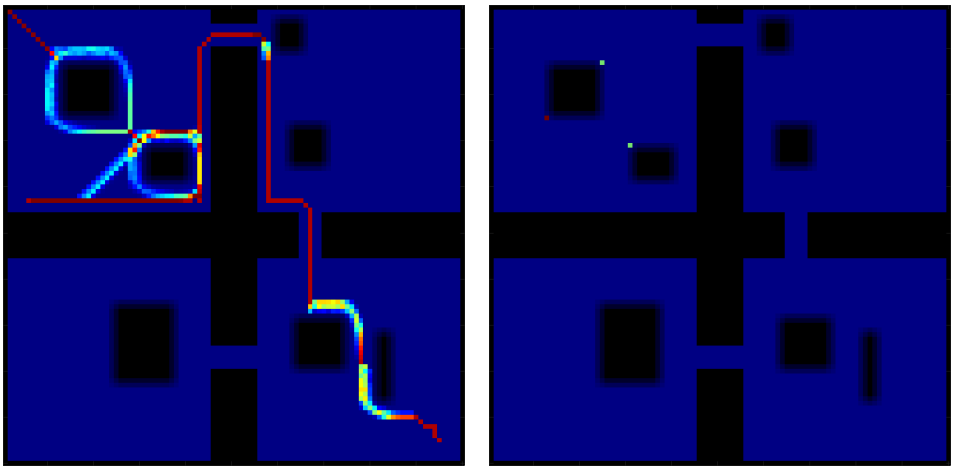


Figure 5.2.6: Simple World - Evaluation 3: Actual World

This evaluation is similar to Evaluation 2 in that the prior known world is empty but in this case the actual world (Figure 5.2.6) has changed. The doorway from the left hand room to the right hand room has been closed off and now contains 3 more dynamic obstacles (7 total). In this world, PI+QMDP success rate was 96% outperforming PI only which had 38%. Shown in Table 5.2.3, PI+QMDP had 4 collisions that occurred closer to the beginning of the trial and are shown in Figure 5.2.7b whereas PI only had 62 collisions mainly occurring at the same obstacles and some closer to the goal.

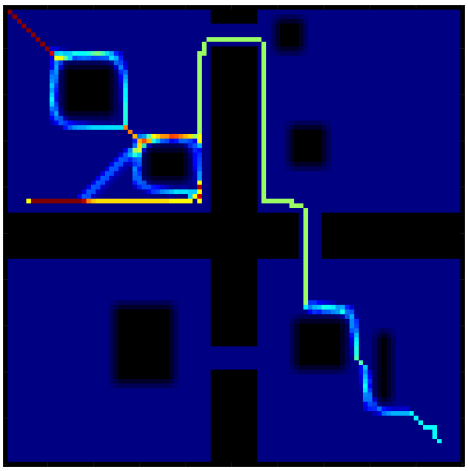
7 Obs.	# Fail.	Avg. FS	Med. FS	Avg. SS	Med. SS
PI+QMDP	4	37	28	308.75	308
PI only	62	110.97	58	301.76	301.5

Table 5.2.3: Simple World - Evaluation 3: Results

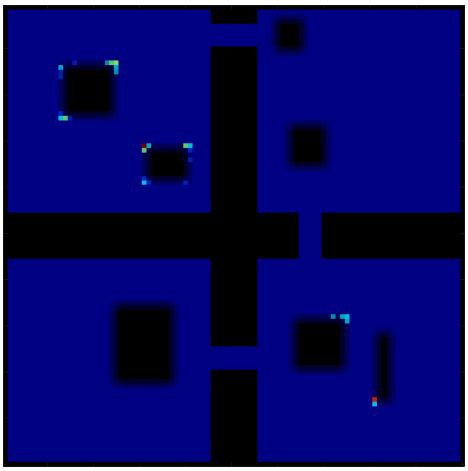


(a) PI+QMDP trajectories

(b) PI+QMDP collisions



(c) PI only trajectories



(d) PI only collisions

Figure 5.2.7: Simple World - Evaluation 3: PI+QMDP versus PI only trajectories and collisions.

5.2.2 Maze World Results

Evaluation 1

The first evaluation for a world that resembles a maze, the prior known world used for offline training was empty and the actual world used during online execution is shown in Figure 5.2.8. There were 5 dynamic obstacles but online the agent encountered only 4. Table 5.2.4 summarize the results from this world with PI+QMDP having fewer collisions (7% failure rate) compared to PI only (47% failure rate) and in PI only most failures occurred when encountering the first dynamic obstacle (Figure 5.2.9d). Figure 5.2.9a depicts the PI+QMDP agent staying further away from the walls compared to the PI only agent shown in Figure 5.2.9c.

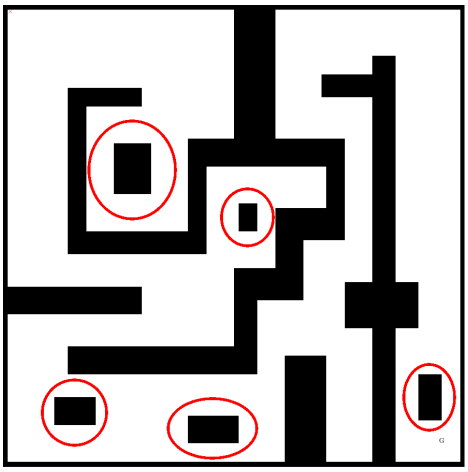
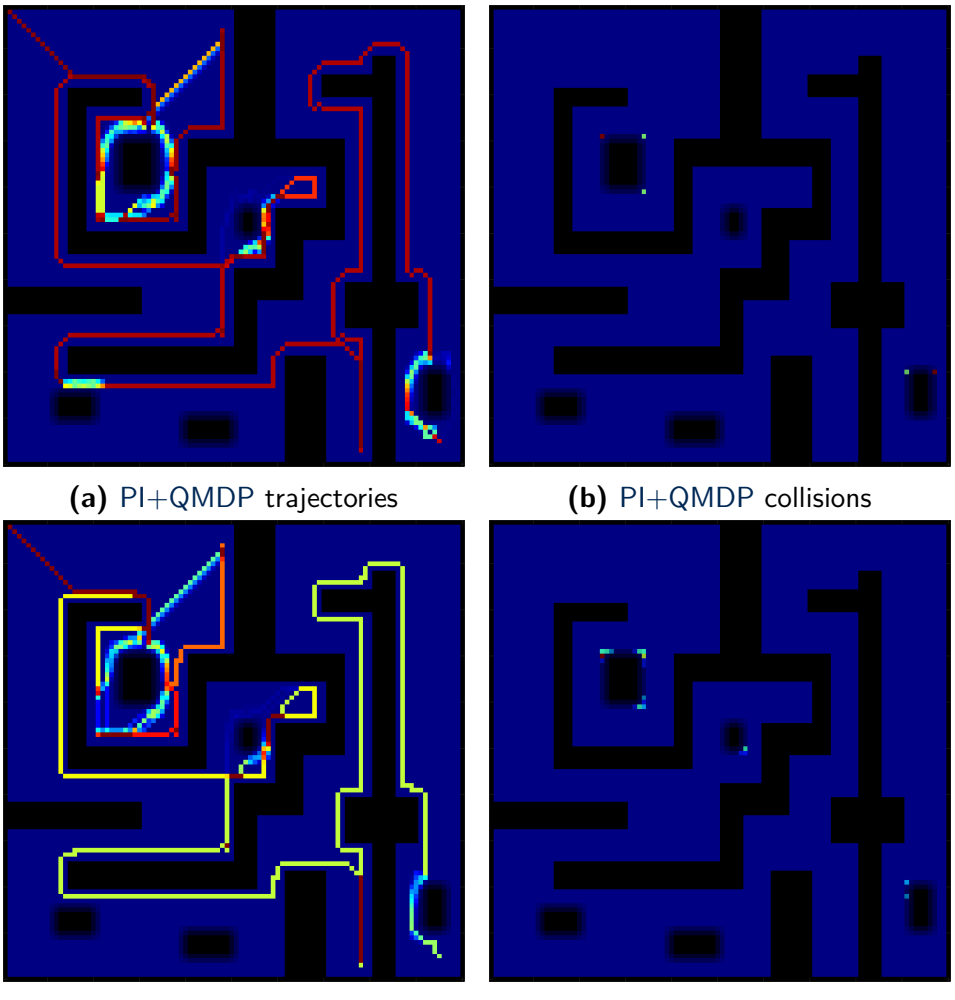


Figure 5.2.8: Maze World - Evaluation 1: Actual world where dynamic obstacles are circled in red.

5 Obs.	# Fail.	Avg. FS	Med. FS	Avg. SS	Med. SS
PI+QMDP	7	380.29	170	733.53	737
PI only	47	188.32	167	734.21	733

Table 5.2.4: Maze World - Evaluation 1: Results

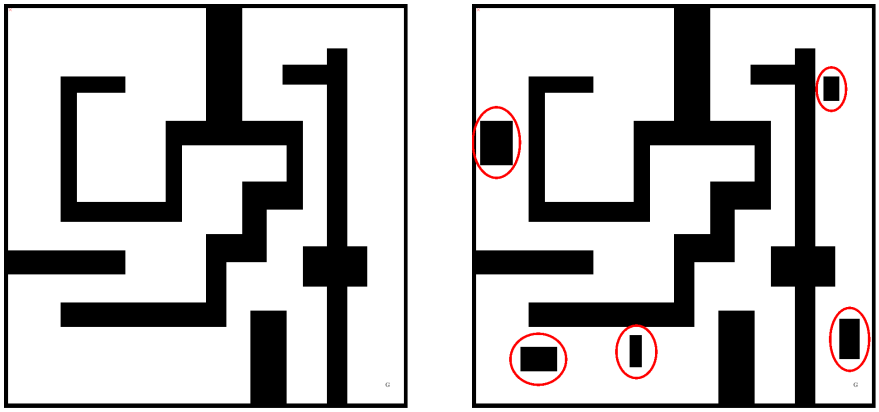


(a) PI+QMDP trajectories **(b)** PI+QMDP collisions

(c) PI only trajectories **(d)** PI only collisions

Figure 5.2.9: Maze World - Evaluation 1: PI+QMDP versus PI only trajectories and collisions.

Evaluation 2



(a) Known world used for offline training.

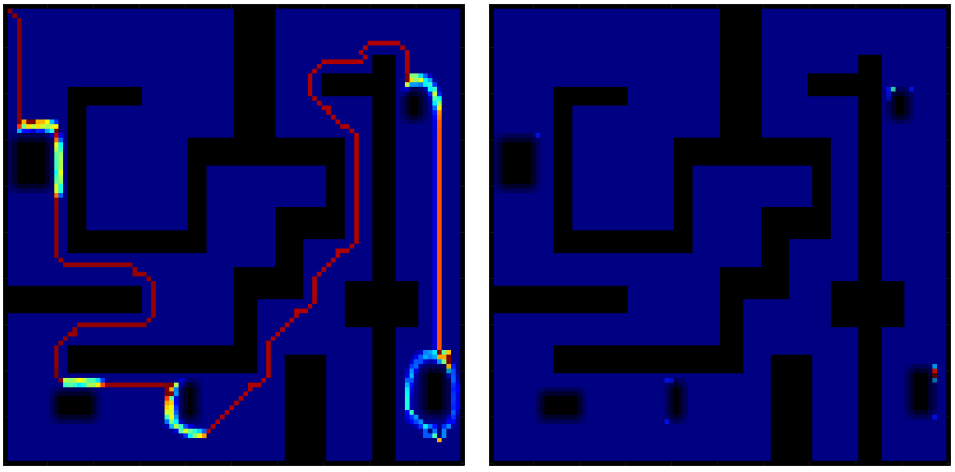
(b) Actual world with dynamic obstacles are circled in red.

Figure 5.2.10: Maze World - Evaluation 2: Global Worlds

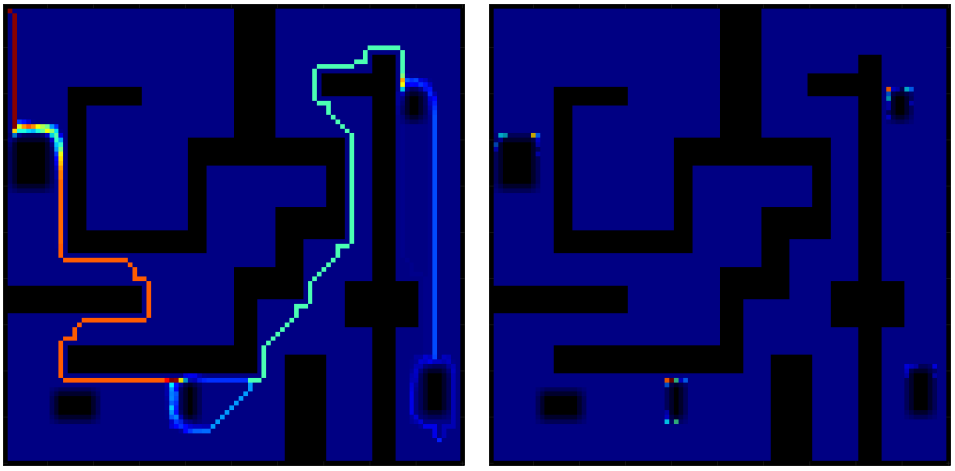
In this assessment, a provided known world containing static obstacles shown in Figure 5.2.10a was used for the offline procedure and the actual world displayed in Figure 5.2.10b was utilized for online operation. The actual world contained 5 dynamic obstacles along the agent's anticipated path. The results in Table 5.2.5 present PI+QMDP performing better (71% success rate) than PI only (15% success rate). Looking at the paths from Figure 5.2.11a and Figure 5.2.11c it can be seen that PI+QMDP took safer paths to the goal compared to PI only. Safer path selection can explicitly be seen at the 3rd encountered obstacle at the bottom. PI only would attempt to go through the narrow space between the wall and the obstacle, which is where most collisions in this scenario occurred, but PI+QMDP used uncertainty of the obstacle's motion to go around the obstacle. This evaluation also shows how performance is increased with the more prior information given. In the previous evaluation (Maze World - Evaluation 1) where the prior known world is unknown, the agent needed to take more than 700 steps to reach the goal whereas in this case the agent took less than 400 steps.

5 Obs.	# Fail.	Avg. FS	Med. FS	Avg. SS	Med. SS
PI+QMDP	29	313.97	359	383.69	382
PI only	85	173.75	152	362.67	359

Table 5.2.5: Maze World - Evaluation 2: Results



(a) PI+QMDP trajectories **(b)** PI+QMDP collisions



(c) PI only trajectories **(d)** PI only collisions

Figure 5.2.11: Maze World - Evaluation 2: PI+QMDP versus PI only trajectories and collisions.

Evaluation 3

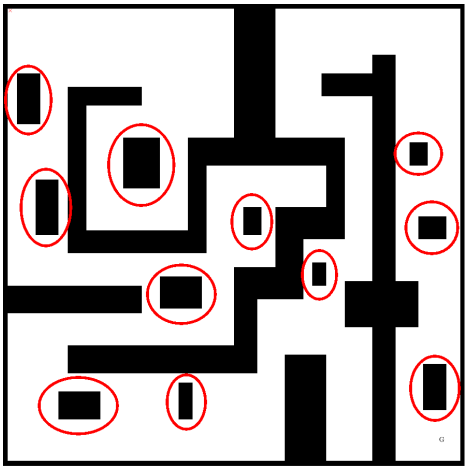
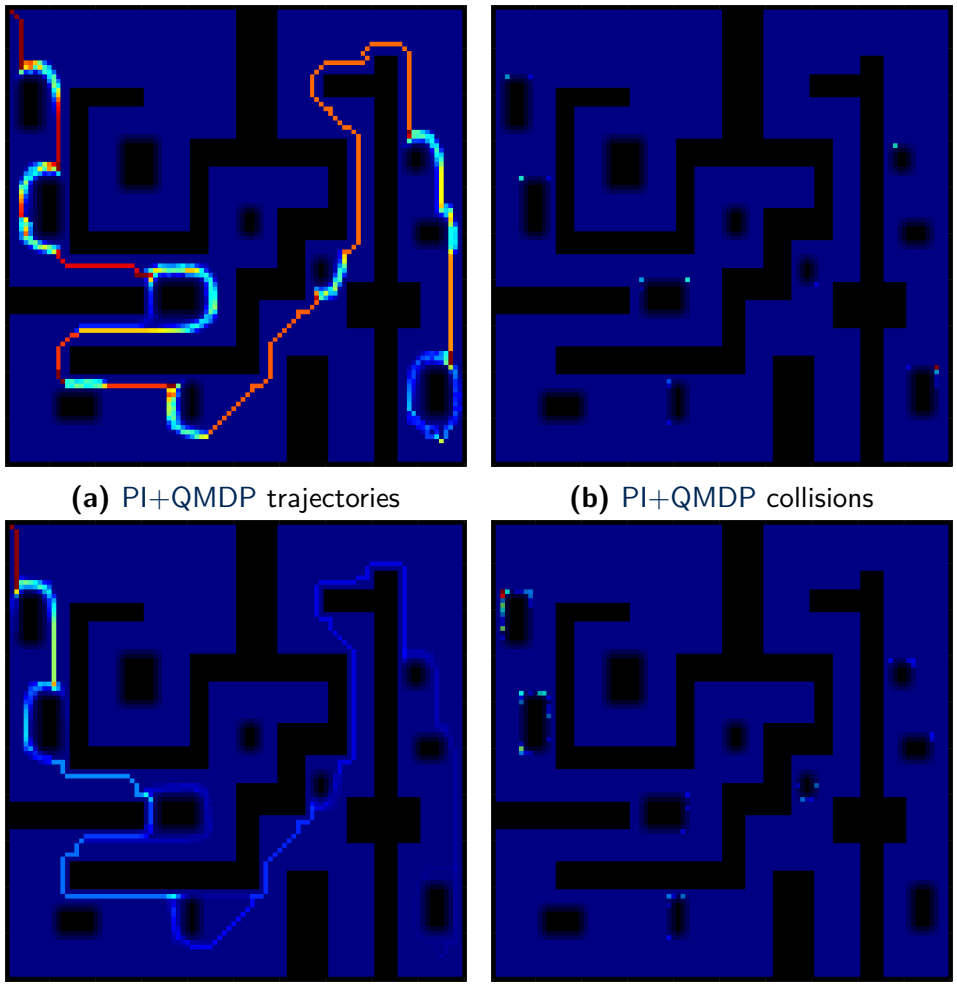


Figure 5.2.12: Maze World - Evaluation 3: Actual world where dynamic obstacles are circled in red.

For the final evaluation, the prior known world is the same as in the previous assessment (Figure 5.2.10a) but now the actual world contains a total of 11 dynamic obstacles that are shown in Figure 5.2.12. During execution, the agent encounters 9 dynamic obstacles and the results in Table 5.2.6 show that the implemented PI+QMDP is able to complete more simulations successfully. The failure rate for PI only close to 100% where most collisions occurred at the first encountered obstacle because it does not use motion uncertainty to select actions and only operates on what is observed at that time. The number for collisions in this PI+QMDP evaluation was greater partly because the environment is much more complex and the agent needs to move between narrow spaces. Also, in all cases where collision occurred, the obstacles jumped 2 spaces over instead of one. This will be briefly discussed in Section 5.2.3 and more elaborated in the future work section of the next chapter.

11 Obs.	# Fail.	Avg. FS	Med. FS	Avg. SS	Med. SS
PI+QMDP	43	244.53	255	429.65	433
PI only	97	77.72	48	392.33	392

Table 5.2.6: Maze World - Evaluation 3: Results



(a) PI+QMDP trajectories **(b)** PI+QMDP collisions

(c) PI only trajectories **(d)** PI only collisions

Figure 5.2.13: Maze World - Evaluation 3: PI+QMDP versus PI only trajectories and collisions.

5.2.3 Simulation Results Summary

What has been shown is using limited or inaccurate information of the environment for prior training and incorporating QMDP into PI produces safer trajectories compared to when using only PI. In total, PI+QMDP had approximately an 85% success rate versus PI only's success rate of roughly 37%. Table 5.2.7 compiles all the results from all assessments: for each world (World), each evaluation (Eval.) and number of encountered dynamic obstacles (Obs.; *Note*: not the total number in the environment), the algorithm used (Alg.), the number of collisions, average and median of steps leading up to failure, along with average and median of steps in cases where the agent reached the goal.

More collisions occurred in Maze World evaluation 2 and 3 partly due to the increased complexity of the environment and because the obstacle jumped 2 spaces over. The presented algorithm in chapter 4, assumes that the obstacle will only move to one of the immediate, surrounding spaces. The solutions to mitigate this will be proposed and discussed in the next chapter's future work section. The main takeaway is that the proposed method can work and produce safer trajectories with some tuning and potential upgrades.

In all the presented simulations, the maximum number of obstacles seen locally were 4. The average time QMDP took to determine a policy was around 6 seconds, which is acceptable for real-time operation on some real robots. The simulations were run using an Intel Core i7 CPU @ 4.20GHz with 4 cores in MATLAB, which does not have optimal performance in terms of speed compared to C++.

World	Eval. \Obs.	Alg.	# Fail.	Avg. FS	Med. FS	Avg. SS	Med. SS
Simple	Eval. 1 \Obs. 2	**PI+QMDP**	3	124.67	120	157.76	152
		PI only	37	104.97	114	160.46	163
	Eval. 2 \Obs. 2	**PI+QMDP**	2	169	169	207.86	203
		PI only	48	100.52	102.50	214.02	215
	Eval. 3 \Obs. 5	**PI+QMDP**	4	37	28	308.75	308
		PI only	62	110.97	58	301.76	301.5
Maze	Eval. 1 \Obs. 4	**PI+QMDP**	7	380.29	170	733.53	737
		PI only	47	188.32	167	734.21	733
	Eval. 2 \Obs. 5	**PI+QMDP**	29	313.97	359	383.69	382
		PI only	85	173.75	152	362.67	359
	Eval. 3 \Obs. 9	**PI+QMDP**	43	244.53	255	429.65	433
		PI only	97	77.72	48	392.33	392

Table 5.2.7: Summary of all results.

6

Conclusion

In this thesis, the problem of navigation in the presence of dynamic obstacles was formulated into a MDP and POMDP. The solution used a combination of PI and QMDP to determine safer policies by accounting for uncertainty of possible next obstacle states. This chapter concludes the completed work by providing an overview of the contributions, a brief discussion on limitations, and considerations for future work.

6.1 Discussion

The major contributions of this work was formulating a solution to the problem of navigation in the presences of dynamic obstacles when their motion is uncertain and leveraging prior information. The following questions, which sparked motivation for this research, were asked in the problem statement:

1. Given what is known prior, which may not be accurate, how can this information used in an offline training process to offload online computation and provide a closed-loop solution at any location?

2. Then during online operation, is there a way to leverage this prior knowledge to **quickly** determine new actions when encountering new information not previously seen and still account for uncertainty of the obstacle motion?

3. How can we generalize the solution so that it can be applicable to other robots that have the same drive-train?

First, the navigation problem is formulated into a MDP in an occupancy grid setting. This grid world expresses the known state of the environment, which may include obstacles. The global prob-

lem is solved using PI, providing general policies for all possible (known) states and a value function that when used locally help propel the agent toward the goal.

Second, during execution, the agent leverages these results and uses locally obtained information of the surrounding environment in the form of a small occupancy grid to provide a short term plan around hazards. When a new obstacle is encountered, it is uncertain where the obstacle will move next and the problem becomes a POMDP. The solution to the local problem was found using a novel application of QMDP. It assumes all obstacles move randomly based on a Gaussian distribution and uses beliefs of all possible obstacle state combinations at the next time step to determine safer actions. Because the local resolution is small and by taking advantage of the global policies in the local context (which helps speed up convergence), the presented approach has the potential to operate in real time, on a real robot. Also, although G-PI is executed again online, this process can be parallelized. Regardless, solutions are found quickly since the updated information retrieved from the local world is small compared to the global and by leveraging the previous global policies, the algorithm is able to converge quickly.

Finally, by using MDPs and POMDPs, the solution should be general enough to be applicable to any holonomic robot. The only case specific requirement is the state transition model, which can be obtained online while executing the proposed algorithm.

Simulation results show that the agent was able to determine safer maneuvers than when only using PI. A limitation in the proposed approach is that it was designed for a holonomic robot and most robots operating in the real-world are non-holonomic. Also, it is locally-short sighted and does not use motion estimates of moving obstacles that could be beneficial in situations such as self-driving cars on a highway. Another limitation is that it only considers the obstacle being in one of the surrounding immediate spaces. If the obstacle moves more than that, then there is a

risk of collision. Although the results show that in much more complex environments there were a relatively large number of collisions, the key take away is that this unique way of using QMDP to approximate POMDPs proves the methodology and shows the potential to work well in terms of navigating dynamic obstacles, providing a foundation for future directions discussed next.

6.2 Future Work

The first future course to implement and verify the proposed algorithm on a real robot, as described in Section 5.1.2, and verify it's application. Then expanding to a multi-agent scenario, where each agent independently runs the PI+QMDP formulation and only sees each other as potential obstacles. It would be interesting to observe how each agent reacts to one another.

The most collisions occurred in the Maze World evaluations 2 and 3 because the dynamic obstacles were located in narrow corridors. In all test cases, all the collisions that occurred under PI+QMDP were due to the obstacle jumping 2 states over (e.g., from $[8, 6]$ to $[6, 6]$). The implemented QMDP only considers cases where the obstacle would move to one of the immediate, surrounding states (e.g., from $[8, 6]$ to $[7, 6]$ or $[7, 5]$ and so on). To mitigate this, expanding the local resolution from a 7×7 to a 9×9 or 11×11 to allow expansion of the -10 reward border around obstacles, would help deter the agent from getting too close to an obstacle. The most likely best solution, which should resolve the issue, is to consider dynamic obstacle uncertainty for 2-3 surrounding states in the QMDP formulation. If the uncertainty were to expand to 2 surrounding cells, then the number of potential Q states would be 25^n where n is the number of detected obstacles. Depending on the local resolution, this may still be possible to execute online especially if the calculation of the value function for several Q states can be done in parallel. Expanding to 3 surrounding cells would increase the number of possible Q states to 49^n, which could be infeasible to solve online for

a large number of obstacles. In these large domains, using NN would be beneficial such as QMDP-net [40] to learn all possible Q state combinations offline and use the learned model online. For example, in a 7×7 local world setting and with the robot in the center, for scenario of 24 independent obstacles (worst case) would mean approximately 49^{24} Q states. All combinations for all number of obstacles, potential sizes, and potential next states can be pre-determined and trained offline. Then during online execution, the agent could use the learned model for determining the policy to execute.

References

[1] Jamie Snape, Jur Van Den Berg, Stephen J Guy, and Dinesh Manocha. The hybrid reciprocal velocity obstacle. *IEEE Transactions on Robotics*, 27(4):696–706, 2011.

[2] Jur Van Den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *Robotics research*, pages 3–19. Springer, 2011.

[3] Jur Van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *2008 IEEE International Conference on Robotics and Automation*, pages 1928–1935. IEEE, 2008.

[4] Daman Bareiss and Jur van den Berg. Generalized reciprocal collision avoidance. *The International Journal of Robotics Research*, 34(12):1501–1514, 2015.

[5] Georges S Aoude, Brandon D Luders, Joshua M Joseph, Nicholas Roy, and Jonathan P How. Probabilistically safe motion planning to avoid dynamic obstacles with uncertain motion patterns. *Autonomous Robots*, 35(1):51–76, 2013.

[6] Pete Trautman, Jeremy Ma, Richard M Murray, and Andreas Krause. Robot navigation in dense human crowds: Statistical models and experimental studies of human–robot cooperation. *The International Journal of Robotics Research*, 34(3):335–356, 2015.

[7] Yu Fan Chen, Miao Liu, Michael Everett, and Jonathan P How. Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 285–292. IEEE, 2017.

[8] Yu Fan Chen, Michael Everett, Miao Liu, and Jonathan P How. Socially aware motion planning with deep reinforcement learning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1343–1350. IEEE, 2017.

[9] Johann Borenstein, Yoram Koren, et al. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE transactions on robotics and automation*, 7(3):278–288, 1991.

[10] Yunfeng Wang and Gregory S Chirikjian. A new potential field method for robot path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 2, pages 977–982. IEEE, 2000.

[11] Gonzalo Ferrer, Anais Garrell, and Alberto Sanfeliu. Social-aware robot navigation in urban environments. In *2013 European Conference on Mobile Robots*, pages 331–336. IEEE, 2013.

[12] Min Cheol Lee and Min Gyu Park. Artificial potential field based path planning for mobile robots using a virtual obstacle concept. In *Proceedings 2003 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM 2003)*, volume 2, pages 735–740. IEEE, 2003.

[13] Charles W Warren. Global path planning using artificial potential fields. In *Proceedings, 1989 International Conference on Robotics and Automation*, pages 316–321. Ieee, 1989.

[14] Michael Everett, Yu Fan Chen, and Jonathan P How. Motion planning among dynamic, decision-making agents with deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3052–3059. IEEE, 2018.

[15] Eric R Mueller and Mykel Kochenderfer. Multi-rotor aircraft collision avoidance using partially observable markov decision processes. In *AIAA Modeling and Simulation Technologies Conference*, page 3673, 2016.

[16] Steven M LaValle, James J Kuffner, BR Donald, et al. Rapidly-exploring random trees: Progress and prospects. *Algorithmic and computational robotics: new directions*, (5):293–308, 2001.

[17] Huijuan Wang, Yuan Yu, and Quanbo Yuan. Application of dijkstra algorithm in robot path-planning. In *2011 second international conference on mechanic automation and control engineering*, pages 1067–1069. IEEE, 2011.

[18] Patrick Reignier. Fuzzy logic techniques for mobile robot obstacle avoidance. *Robotics and Autonomous Systems*, 12(3-4):143–153, 1994.

[19] Mohammed Faisal, Ramdane Hedjar, Mansour Al Sulaiman, and Khalid Al-Mutib. Fuzzy logic navigation and obstacle avoidance by a mobile robot in an unknown dynamic environment. *International Journal of Advanced Robotic Systems*, 10(1):37, 2013.

[20] Mohammed Faisal, Mohammed Algabri, Bencherif Mohamed Abdelkader, Habib Dhahri, and Mohamad Mahmoud Al Rahhal. Human expertise in mobile robot navigation. *IEEE Access*, 6:1694–1705, 2017.

[21] Kimberly McGuire, Guido De Croon, Christophe De Wagter, Karl Tuyls, and Hilbert Kappen. Efficient optical flow and stereo vision for velocity estimation and obstacle avoidance on an autonomous pocket drone. *IEEE Robotics and Automation Letters*, 2(2):1070–1076, 2017.

[22] Stanislav Mikhel, Dmitry Popov, and Alexandr Klimchik. Collision driven multi scenario approach for human collaboration with industrial robot. In *Proceedings of the 2018 4th International Conference on Mechatronics and Robotics Engineering*, pages 78–84, 2018.

[23] Lei Tai, Giuseppe Paolo, and Ming Liu. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 31–36. IEEE, 2017.

[24] Gregory Kahn, Adam Villaflor, Vitchyr Pong, Pieter Abbeel, and Sergey Levine. Uncertainty-aware reinforcement learning for collision avoidance. *arXiv preprint arXiv:1702.01182*, 2017.

[25] Antoine Bautin, Luis Martinez-Gomez, and Thierry Fraichard. Inevitable collision states: a probabilistic perspective. In *2010 IEEE international conference on robotics and automation*, pages 4022–4027. IEEE, 2010.

[26] Daniel Althoff, James J Kuffner, Dirk Wollherr, and Martin Buss. Safety assessment of robot trajectories for navigation in uncertain and dynamic environments. *Autonomous Robots*, 32(3):285–302, 2012.

[27] Thomas M Howard, Colin J Green, Alonzo Kelly, and Dave Ferguson. State space sampling of feasible motions for high-performance mobile robot navigation in complex environments. *Journal of Field Robotics*, 25(6-7):325–345, 2008.

[28] Mihail Pivtoraiko, Ross A Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009.

[29] Pinxin Long, Wenxi Liu, and Jia Pan. Deep-learned collision avoidance policy for distributed multiagent navigation. *IEEE Robotics and Automation Letters*, 2(2):656–663, 2017.

[30] Haoyu Bai, Shaojun Cai, Nan Ye, David Hsu, and Wee Sun Lee. Intention-aware online pomdp planning for autonomous driving in a crowd. In *2015 ieee international conference on robotics and automation (icra)*, pages 454–460. IEEE, 2015.

[31] Josep M Porta, Nikos Vlassis, Matthijs TJ Spaan, and Pascal Poupart. Point-based value iteration for continuous pomdps. *Journal of Machine Learning Research*, 7(Nov):2329–2367, 2006.

[32] Alex Brooks, Alexei Makarenko, Stefan Williams, and Hugh Durrant-Whyte. Parametric pomdps for planning in continuous state spaces. *Robotics and Autonomous Systems*, 54(11):887–897, 2006.

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

[34] Mykel J Kochenderfer. *Decision making under uncertainty: theory and application*. MIT press, 2015.

[35] Michael L Littman, Anthony R Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In *Machine Learning Proceedings 1995*, pages 362–370. Elsevier, 1995.

[36] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2005.

[37] ROBOTIS. Turtlebot3 burger. http://www.robotis.us/turtlebot-3-burger-us/, 2020.

[38] Conner Todd Castle. Virtual morphology as a method of robotic control. 2019.

[39] Kevin M Lynch and Frank C Park. *Modern Robotics*. Cambridge University Press, 2017.

[40] Peter Karkus, David Hsu, and Wee Sun Lee. Qmdp-net: Deep learning for planning under partial observability. In *Advances in Neural Information Processing Systems*, pages 4694–4704, 2017.