MDP-based Adaptive Motion Planning for Autonomous Robot Operations under Degraded Conditions

---

A Thesis

Presented to

the Faculty of the School of Engineering and Applied Science

University of Virginia

In Partial Fulfillment

of the requirements for the Degree

Master of Science (Computer Engineering)

by

Phillip Seaton

December 2021

Approval Sheet

This thesis is submitted in partial fulfillment of the requirements for the degree of

Master of Science (Computer Engineering)

Phillip Seaton

This thesis has been read and approved by the Examining Committee:

Nicola Bezzo, Advisor

Harry Powell, Committee Chair

Joanne Dugan

Accepted for the School of Engineering and Applied Science:

Jennifer West, Dean, School of Engineering and Applied Science

December 2021

Abstract

Autonomous mobile robots (AMR) like ground and aerial vehicles may encounter internal failures and external disturbances when deployed in real-world scenarios compromising the success of a mission. This thesis proposes an online learning method to adapt the motion planner to recover and continue an operation after a change in a robot's dynamics. Our proposed framework builds on the Markov Decision Process (MDP) and leverages the residual – defined in this work as the difference between the predicted and the actual state – to update the transition probabilities online and in turn update the optimal MDP policy. To maintain the system safe during learning, we propose a χ^2-based dynamic learning rate that is event-triggered when the robot approaches an unsafe region of the workspace. Our framework can also distinguish between external disturbances versus internal failures by tracking the robot's state in a local and fixed frame view. We finally propose a state-machine-based resetting procedure to return to a previous MDP model when the problem disappears. This framework for resilient planning of impaired vehicles is validated both in simulations and experiments on unmanned ground vehicles (UGV) in a cluttered environment. Finally, we show an extension of our framework for multitask cooperative missions in which robots need to balance tasks based on the impaired dynamics of the robots in the network.

Acknowledgements

I am thankful for my advisor, Dr. Nicola Bezzo, for mentoring me and taking the time to help solve any research problem I had. He helped push me as a student to where I am today. His genuine care for his students has made us all feel supported.

I extend gratitude to the members of our lab group: Esen, Paul, Rahul, Shijie, Jacob, Lauren, Nick, Will, and Carmelo who all gave me many laughs and assistance when performing research.

I would also like to thank Dr. Harry Powell, Dr. Joanne Dugan, and Natalie Edwards for all the guidance during my time at UVA. Both Professor Dugan and Professor Powell have taught me in three classes throughout and have been great resources. Natalie has always been there for her students; she has been able to solve any issue: funding, classes, and even healthcare.

Finally, I would like to thank my Dad and Mom for their endless love and support over the past 26 years. They shaped me into the person I am today.

Funding: I acknowledge: the Defense Advanced Research Projects Agency (DARPA) for providing financial support for my research under the Assured Autonomy program, Contract FA8750-18-C-0090, the National Science Foundation (NSF) through grant # 1816591, and the Department of Electrical and Computer Engineering at the University of Virginia.

Contents

List of Figures

Chapter 1

Introduction

As humans we have the ability to adapt to changes in our environment and solve what we would consider to be simple problems. This may be just walking over a pile of rocks or going uphill. Robots can only act in ways that they are trained or programmed. In many applications of planning, a robot must navigate an environment even with many changing aspects of its surroundings which include ground or aerial disturbances, and internal failures that can lead to improper actions. Disturbances typically arise from the environment and are usually temporary. For example, a sheet of ice or gust of wind may cause the system to be unable to turn properly. Internal problems including system aging tend to last longer; for example, a tire deflating or a wheel stuck may cause improper wheel rotation. We note that typically a compromised system could continue the operation working with limited capabilities, while maintaining safety (e.g., avoid collisions), and even maintaining liveness conditions (e.g., eventually reaching the desired goal). With this consideration in mind, we propose a Markov Decision Process (MDP)-based approach to adapt the motion planner online as the system encounters changes in dynamics or disturbances. The MDP can find an optimal policy given a system's model represented as transition probabilities. At the core of our framework, we leverage a residual, defined as the difference between the predicted state and the measured state, given an action and an initial state. We then use such measures to update the transition probabilities and recompute at runtime the optimal policy for the robot. To speed up learning and avoid undesired behavior near unsafe states, we propose a χ^2-based learning rate update method that detects changes in the dynamics of the system and based on the surrounding environmental conditions (i.e., the presence of obstacles or other safety-critical states).

Our framework is able to distinguish between internal and external problems by tracking predicted and measured states in a local and fixed-frame view. We also implement a state machine-based approach that allows the system to quickly return to a previously trained model when a degradation is temporary (e.g., a

change of slope or in terrain material). We also provide an approach for multi-vehicle coordination for tasks while one vehicle is experiencing faults. Extensive simulations and experiments are presented throughout the thesis to validate the framework.

The rest of the thesis is organized as follows: in Chapter 2, related work on other failure and recovery techniques is summarized. The preliminaries for the MDP and robot dynamics are defined in Chapter 3. The approach for our adaptive MDP approach to recover the system after failures is defined in Chapter 4. Simulations and experiments are shown in Chapter 5. Multi-vehicle formulation and simulations are covered in Chapter 6. We conclude our work in Chapter 7.

1.1 Contribution

The main contribution of this work is on *the design of a computationally efficient online adaptable optimal policy framework for autonomous systems under recoverable (both persistent and temporary) degradation.* We also validate the scheme with extensive simulations and experiments on a UGV navigating a cluttered environment under different disturbances and failures. This thesis specifically has the following six contributions:

Adaptable Markov Decision Process: In the first part of the thesis, we present an MDP-based method for online failure detection and recovery. The traditional MDP is changed to incorporate a reward and penalty system for its transition probabilities. The MDP is recalculated after every iteration of rewards and penalties to generate a new estimation of the rewards associated with the motion model estimation.

Dynamic Learning Rate: We propose a method to speed up model learning of an impaired system when there is a statistical difference between the experimental and predicted motion model. The threshold for these statistical tests is relaxed when safety is a concern. The quicker learning rate allows for quicker change and the ability to avoid danger since the robot learns the optimal move earlier than the typical model.

Reset Mechanism: A reset state machine is employed so that the robot may return to the original model after a temporary degradation (e.g., due to an external disturbance).

External vs Internal Disturbance Detection: We also design an approach to detect and differentiate between external disturbances and internal faults allowing the robot to select the appropriate motion policy.

Multi-Vehicle Coordination under degraded conditions: We provide a solution for an impaired robot to work alongside a normal robot to complete tasks. Each robot takes on the optimal number and order of tasks so that each reaches the final task in the least amount of time.

Experimental Validation: The different contributions are validated in simulation and real-world experiments on UGVs under degradation. The robot is able to complete the task regardless of the disturbance type or motion model parameter initialization.

Chapter 2

Survey of Related Work and the State of the Art in the Recovery of Faulty Systems

One of the most studied problems in robotics is how to adapt an autonomous system against changes in dynamics and in the surrounding environment. A robot may encounter internal faults or external disturbances. A general approach for solving this issue is fault detection and recovery. A robot must discover what is damaged within the system, and either repair the damage or learn how to act with its defective system. Many different research areas of robotics have attempted to solve similar problems; these topics range from control-based approaches to reinforcement learning-based algorithms.

Classic approaches to fault tolerance rely on updating the model of the robot, either directly with self-diagnosis, or indirectly with machine learning; the model is then used for planning or control [1]. For instance, if a legged robot detects that one of its legs is not working as expected, it can stop using it and adapt the controller to use only the remaining working legs [2]. These approaches may need many sensors to detect all of these faults. Most of the current real-world experiments are on legged or wheeled robots [3], [4].

An alternative approach is to let a damaged robot learn a new policy by trial and error, which bypasses the need for a diagnosis. In this line of work, the biggest challenge is to design algorithms that are as data efficient as possible, because too many trials may damage the robot further, and learning must be rapid enough to be useful in real-world situations.

2.1 Machine Learning and Reinforcement Learning

Many propositions try to learn a robot's optimal action when a fault occurs. Learning algorithms can be combined with controllers and planners to finish the current task. Supervised learning [5] has shown to be more effective when teaching a robot how to adapt to a task. Unlabeled data or random robot movements are unlikely to provide helpful information for the learning algorithm but supervised learning allows for learning where the robot needs it.

In comparison to many of these forward learning models, reinforcement learning allows for the robot to learn how to behave through trial and error. The robot learns how to maximize its reward while being damaged [6]. In traditional reinforcement learning, the robot selects the action at the current state that is projected to maximize the robot's reward [7]. Many reinforcement learning approaches use discrete actions spaces which fail to scale well in real-world experiments since the real world is in continuous space. In [8], an algorithm is proposed that attempts to bridge the reality gap in evolutionary robotics through a "transferability approach", which focuses on finding realistic behaviors in simulation that also work on the physical robot. Newer reinforcement learning methods focus on policy search methods for optimizing the inputs of continuous controllers. Direct policy search is model agnostic but model-based approaches are needed when the dimensionality of the search space increases. These model-based approaches are simultaneously learning the model of the robot and learning the optimal policy for the task at hand [9]. Much of the learning time is wasted when trying to learn parts of the new system model that may not be useful for a given task [10].

Reinforcement learning has several constraints. First, the robot needs to restart at an initial state for every iteration of learning [11]. They also have difficulty adapting when the environment changes from the initial training environment. Since much of reinforcement learning is trial and error, damaging a robot at runtime to test the potential outcome of a policy is not ideal. It takes several hours of iterations for convergence. Reinforcement learning also does not scale well into high dimensionality. Reinforcement learning is a robust technique for the initial training of a robot but is not practical for learning when the system changes during a task. Reinforcement learning also has difficulty when used in real-world experiments. The overhead to train these robots can be hours of training to perform simple tasks.

A meta-learning approach was used in unmanned aerial vehicles [12] to track the performance of the system in faults that the robot has not seen. Reductions in thrust in some of the quadrotors emulated a fault, and the reference tracker is updated based on future state predictions of the robot. The system was able to follow its desired trajectory while remaining safe. A neural network based approach for UGVs in

situations of changing or degraded environment is proposed in [11]. The speed of the robot was optimized to reach the goal as quickly as possible while minimizing speed variability and planning for safety.

2.2 Control Based Methods

Control-based approaches have been used for allowing a system to continue when under degraded conditions. Adaptive control has been used to monitor for significant failures and choose the best action to ensure the success of the mission [13]. Authors of [14] present a method for actuator fault estimation and fault resilient control to a problem of trajectory tracking in a system composed of three connected two-wheeled robots. A nonlinear adaptive observer is used for state and fault estimation, the fault tolerance is created by updating dynamic control law with fault estimations. Adaptive control can also be used in fault detection and recovery in swarms [15].

2.3 Markov Decision Process

The Markov decision process is an integral part of stochastic control in robotics. MPDs can be used to solve the dynamics of the system and the environment from experience [16]. A variant of the MDP called the abstract MDP was used for task planning of a taxi problem where the robot had motion primitives and the ability to pick up passengers and drop them off. Although this work uses similar motion primitives, this is solely focused on task management and is not concerned with faults [17]. Partially observable Markov decision (POMDP) processes can be used for fault detection and direct new high-level actions like "give up on this path" or "request a new path" [18]. An approach for fault detection and recovery in autonomous underwater vehicles (AUV) utilizes partially observable Markov decision processes (POMDP) [19]. The POMDP was used for fault management and identification if it was an internal or environmental fault.

2.4 Trial-and-Error Approaches

Several trial-and-error approaches are used under the topic of fault detection and recovery. While robots have a difficult time thinking outside the box, the Intelligent Trial-and-Error algorithm is designed to work around issues similar to how animals adapt [20]. Their approach works with robot arms and legged robots. Before runtime, their algorithm searches for multiple policies that will solve a given action. After the robot is damaged, a subset of these policies is available with the damaged parts of the robots. In a real-world experiment, a 6-legged robot was damaged and able to recover after two minutes of their algorithm attempting

policies. Another similar approach, the Reset-free Trial-and-Error algorithm [1], also computes hundreds of possible behaviors with the simulated dynamics of a real robot. After the vehicle is impaired at runtime, the optimal action is found through a Monte Carlo Tree Search. Trial and error can be exhaustive and does not scale well as the complexity of the system model increases.

Chapter 3

Preliminaries

Before discussing the approach of the thesis, we will define the Markov Decision Process (MDP) theory and robot dynamics that are integral to our framework.

3.1 Markov Decision Process

Our model builds off the traditional Markov Decision Process [21], which entails four components: states X, actions A, transition probabilities P, and reward functions R. This can be written as the tuple: $\{X, A, P, R, x_0\}$. To be Markovian, the previous actions and visited states should not affect the result of an action, and the current state x_i gives enough information to make an optimal decision. The set of states $X \in \mathbb{R}^2$ is the total state space $\{x_1, \ldots, x_{n_X}\}$ where n_X is the total number of states. The size n_X is determined by the height and width of the two-dimensional grid generated from the local view of the robot. This local view entails all obstacles and a waypoint that points towards the goal. The set of actions A is defined as $\{a_1, \ldots, a_{n_A}\}$ where n_A is the total number of potential actions. These actions are chosen to control the next state x'.

Transition functions (P) give the probabilities of entering new states through potential actions. The representation $P(x'|a, x)$ denotes a probability distribution that current state x will transition to the next state x' through action a. This $P(x'|a, x) \rightarrow [0, 1]$ gives the resulting distribution of transition values.

The reward for the MDP is state dependent. The policy will try to navigate through states and get higher rewards until it reaches the maximum reward at the goal state. There are goal states (x_g), loss or obstacle states (x_o), and penalties for movement. These will all be combined through value iteration.

Given states, actions, transitions, and rewards, the optimal policy can be calculated through the value function. These value functions associate optimal criteria to policies. Since rewards are only dependant

on the state, the value function has to associate which action generates the best value depending on the probability distributions.

$$V^{\pi}(x) = \max_{a}[(R(x,a) + \gamma \sum_{x'} P(x'|x,a)V^{\pi}(x'))] \tag{3.1}$$

In (3.1), the widely known Bellman equation is shown. The value of a given action can be determined based on the transition probabilities multiplied by the initial reward and the discounted value of the state. As this function is run over many iterations on the system of states and actions, each state is given a value that is propagated from the initial reward values. The best action corresponds to the optimal policy. Given a state, the best action is chosen through its probability distribution leading it to other states of higher value.

3.2 Action Space and the Robot's Field Of View

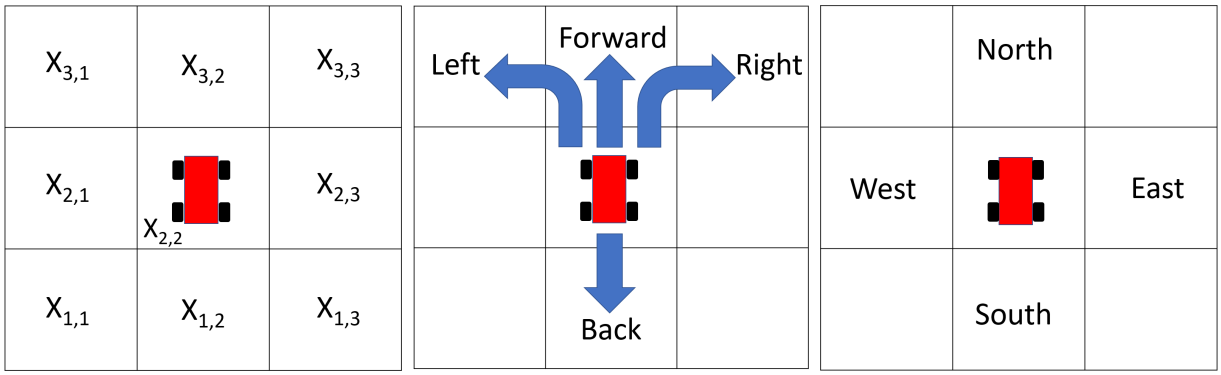
While the Markov decision process involves discrete states and actions, our system is continuous and must be converted to a discrete system for policy optimization. Then the discrete policy must be converted back to continuous through controllers for primitive actions. For ease of discussion, in this work, the action space is divided into four rudimentary actions. The two action spaces are in (3.2). Two different action spaces depend on the frame of the robot as seen in Fig. 3.2.

$$\begin{aligned} A &= \{Forward(a_{\mathrm{f}}), Left(a_l), Right(a_r), Back(a_b)\} \\ A &= \{North(a_n), South(a_s), East(a_e), West(a_w)\} \end{aligned} \tag{3.2}$$

In Fig. 3.1(b) and (c), the local action space considers primitive actions related to the local frame of the robot while the global actions are used for a given fixed-frame and provide cardinal directions of motion. These primitive actions allow the robot to move to reachable states as seen in Fig. 3.1(a). As we will see these two action spaces are necessary to differentiate between internal and external problems on the robot.

Both open-loop and closed-loop controllers were tested. Low-level control is kept simple to embody these primitive actions. Open-loop control can be summarized as going a fixed linear and angular velocity for a given amount of time. Closed-loop control involves a go-to-goal control implementation where the linear and angular velocities are based on a proportional-integral-derivative (PID) controller.

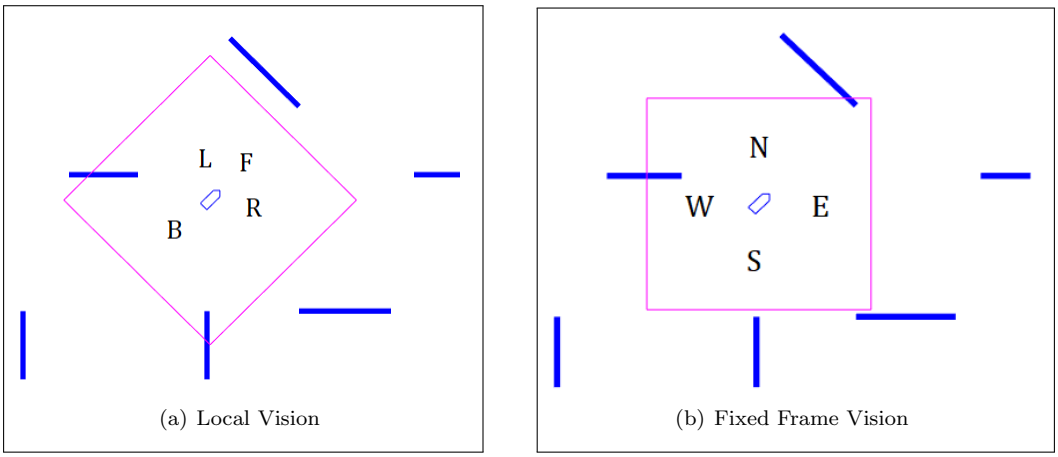
In robotics, a field of view (FOV) must be defined for the robot; this is used to identify objects or obstacles in nearby potential states. X_L is the set of nearby states that are reachable within the robot's view. In the traditional MDP, the FOV is fixed around the robot and does not depend on the direction of the robot. After analyzing the states in view after every iteration, the MDP determines which of the action primitives are optimal for the robot. The action primitives for the local robot do not revolve around the angle orientation



(a) Reachable states (b) Local actions (c) Fixed-frame actions

Figure 3.1: Primitive actions

of the robot. They are strictly action to get a robot from its current state to a surrounding local state. These actions are defined in (3.2) as (North, South, East, West). This fixed FOV frame can be seen in Fig. 3.2(b) where the robot is angled but the frame does not change based on its orientation.



(a) Local Vision (b) Fixed Frame Vision

Figure 3.2: FOV frames with primitive actions

Our variation of the Markov Decision Process varies through its local approach. Instead of the typical global variation, we orient the MDP state space around the robot. Then local model generates policies that are local actions and are entirely dependent on the pose of the robot. These actions would include (Forward, Left, Right, Back). The actions to get to surrounding desired states would change as the orientation changes. The blue lines in Fig. 3.2 denote obstacles while the pink box represents the frame around the robot. This local frame can be seen in Fig. 3.2(a); the local frame is shown to pivot with the orientation of the robot.

3.3 Ground Vehicle Dynamics

For the extent of the real-world experiments in this thesis, we will use a ROSbot [22]. The ground vehicle dynamics of a similar robot can be represented with a non-holonomic skid-steering model as:

$$\begin{aligned} \dot{x_R} &= \frac{r}{2}(\omega_r + \omega_l)\cos\theta_R \\ \dot{y_R} &= \frac{r}{2}(\omega_r + \omega_l)\sin\theta_R \\ \dot{\theta_R} &= \frac{r}{L}(\omega_r - \omega_l) \end{aligned} \tag{3.3}$$

where x_R, y_R, and θ_R are the position and orientation of the robot and ω_r and ω_l are the angular velocities of the right and left wheels, respectively. L is the wheel base length, and r is the radius of the wheel.

For our real-world experiments, the robot was limited via software not allowing to move backwards and/or right. For example, a right turn may be limited to going straight if the $\omega_r \geq \omega_l$. The left wheel angular velocity would not be able to exceed the right wheel angular velocity resulting in the right action being unavailable. If the angular velocity of the wheel was limited to being zero or positive, then the robot would not be able to go in reverse.

Chapter 4

Adapting MDP Framework

4.1 Problem Formulation and Hypotheses

The goal of the work is to find the optimal policy for motion planning in an environment while having some of the vehicle's action space A limited. Due to system failures or perturbations, these impaired dynamics $f(x, u, d)$ differ from the original dynamics $f(x, u)$. Our system has an environment with the set of states $\{x_{goal}, x_{start}, x_{obstacle}\} = \{x_g, x_s, x_o\} \in X$. Given these states, the robot must choose the best series of actions to navigate through the environment to the goal state while the system dynamics are changing. This equates to the robot reaching the goal state $x(t) = x_g$ as $t \to \infty$. Failures may arise and the system must adapt while avoiding unsafe states.

PROBLEM. **Trajectory Planning in Environments with Disturbances** *Given a vehicle with state x and the dynamics $\dot{x} = f(x, u)$, the dynamics become $\dot{x} = f(x, u, d)$ due to external disturbances $d = d_e$ or internal problems $d = d_i$. Based on these assumptions, the robot will need to generate a motion model estimation and detect which actions are affected. Using this model, the robot must find the optimal policy π that leads the vehicle to the goal while avoiding all the obstacles.*

The system will have to adapt dynamically as the system failures or disturbances are added and removed. The robot will have a possible combination of actions to get to the goal regardless of disturbances and obstacles. The environment's obstacle density will be limited to allow for a possible trajectory from start to finish. The robot may encounter one or a series of tasks based on the environmental setup.

4.2 The Adaptable Markov Decision Process

The overall system architecture is summarized in Fig. 4.1. The current state x_{ref} of the robot is passed into the planner where the MDP is calculated. From the MDP, an optimal action is chosen which is then translated into control inputs for the robot. Various disturbances d affect the dynamics around the plant. On the feedback side, an original matrix of the transition probabilities and measurement of the current state is sent to the reset state machine. The transition probability matrix P is then potentially reset to the original model if the problem had disappeared. A transition matrix estimate is used for non-reset conditions and goes through the adaptation process.

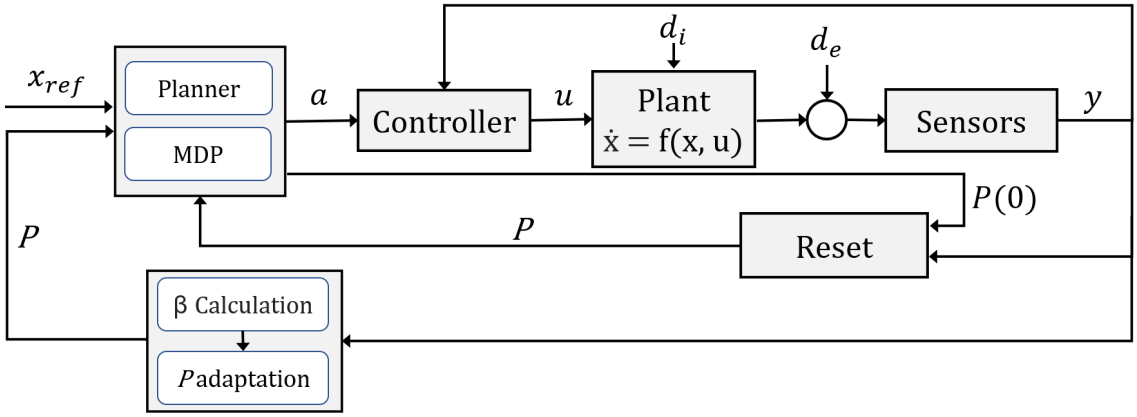


Figure 4.1: System architecture

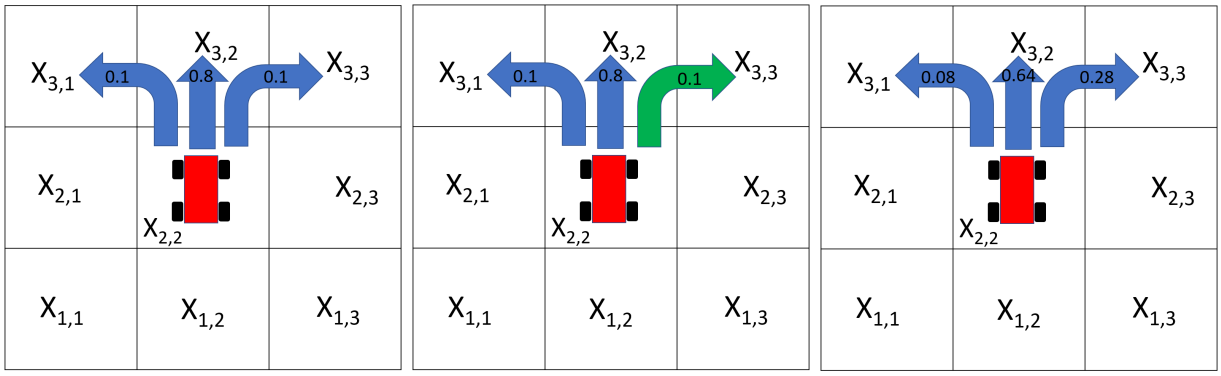
We propose a variation of a typical Markov decision process. The focus of this adapting Markov decision process is to have the transition probabilities adjust to the true values of the underlying transition dynamics. $P(x'|x, a)$ changes depending on the dynamics of each primitive action. We assume that our dynamics change $f(x, u, d)$ from the original dynamics $f(x, u)$. As the dynamics change, the actions reach different states than the original dynamics. Hence, the need to update $P(x'|x, a)$. The adapting Markov decision process uses the same interaction of states, actions, rewards, and transitions probabilities to the traditional MDP. Bellman equations are used to calculate the reward of the MDP. The difference is that these transition probabilities change depending on what state was chosen and performed. Consider P where $P \in \mathbb{R}^{A \times X_L}$, each action is associated with all local states in the transition probability matrix. After each iteration, a new actual state x'_{a} is visited given an action and previous state. $P(x'_{\mathrm{a}}|x, a)$ is then rewarded using (4.1). In (4.2), the probabilities of the unvisited states x'_{u} are normalized.

$$P(x'_{\mathrm{a}}|x, a) = (P(x'_{\mathrm{a}}|x, a) + \beta)/(1 + \beta) \tag{4.1}$$

$$P(x'_{\mathrm{u}}|x,a) = P(x'_{\mathrm{u}}|x,a)/(1+\beta) \tag{4.2}$$

The framework uses a learning rate β to increase the probability estimation that corresponds to the given optimal action and actual state (a_o, x'_{a}). This process rewards and normalizes all of the probabilities corresponding to the desired action. The probabilities that are rewarded are updated using the first equation (4.1) which increases the associated probability while having smaller and smaller increases as this given probability gets closer to one. The rest of the probabilities of the unvisited states x'_{u} in that action row are updated using the second equation (4.2). The penalties become smaller as the probability gets closer to zero. The denominator of both equations ensures that the sum of the probabilities of a desired action is still one.

In Fig. 4.2(a), a robot starts in x_0 and achievable states are surrounding our robot. From the vision of the robot, there are nine states $(x_{1,1}, x_{1,2}, x_{1,3}, x_{2,1}, x_{2,2}, x_{2,3}, x_{3,1}, x_{3,2}, x_{3,3})$ from $x_{2,2}$. There are four actions as mentioned in (3.2): Forward, Left, Right, and Backwards. In Fig. 4.2, the probabilities for a forward action are listed for the relevant achievable states.



(a) Example reachable states (b) Right turn chosen (t=0) (c) Resultant state (t=1)

Figure 4.2: Attempted forward action and probability update

Assume that the robot's reward is maximized by choosing a_{f}. Although the forward action was predicted to be most likely, a right turn occurred shown in green in Fig. 4.2(b). The vehicle reached the local state $x_{3,3}$ but the local frame was recentered so $x_{3,3}$ becomes $x_{2,2}$ since the local frame shifts as the robot moves. The local frame also pivots with the robot, keeping the local states with the same relative orientation at each position. In Fig. 4.2, the probabilities to get to each reachable state given a forward action are listed on the arrows.

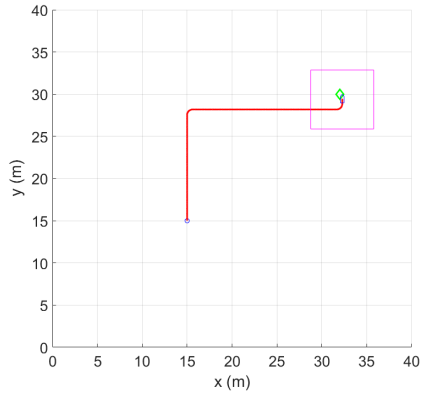
The matrix relating to the example in Fig. 4.2 has the action and state pairing (a_k, x'_i) where k and i represent a single action or state from their respective sets. The optimal action a_o is the action that was computed from the MDP and has the maximum expected reward over the next iteration. The actual state x'_{a} is the measured state after the movement update is performed on the robot. Assume the transition

probabilities in the first matrix of (4.3). With the desired action of a_{f} and actual state $x_{3,3}$, reward equation (4.1) is performed on $(a_o, x'_{\mathrm{a}}) = (a_{\mathrm{f}}, x_{3,3})$. The remainder of the row a_{f} is then normalized by (4.2).

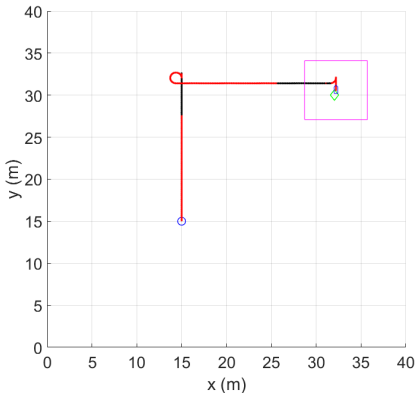
The example in (4.3) is a smaller scale of what is implemented in the simulations. While four actions are still used in the simulations, the local states are in a 9x9 grid which increases the action state matrix size.

$$
\begin{array}{c}
\begin{array}{c} \\ a_{\mathrm{f}} \\ a_l \\ a_r \\ a_b \end{array}
\begin{array}{c}
\begin{array}{ccccccccc} x_{1,1} & x_{1,2} & x_{1,3} & x_{2,1} & x_{2,2} & x_{2,3} & x_{3,1} & x_{3,2} & x_{3,3} \end{array} \\
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0.8 & 0.1 \\
0 & 0 & 0 & 0.1 & 0 & 0 & 0.8 & 0.1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.1 & 0 & 0.1 & 0.8 \\
0.1 & 0.8 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\end{array}
\Longrightarrow \text{Movement Update} \\
\\
\begin{array}{c} \\ a_{\mathrm{f}} \\ a_l \\ a_r \\ a_b \end{array}
\begin{array}{c}
\begin{array}{ccccccccc} x_{1,1} & x_{1,2} & x_{1,3} & x_{2,1} & x_{2,2} & x_{2,3} & x_{3,1} & x_{3,2} & x_{3,3} \end{array} \\
\begin{pmatrix}
\frac{0}{1+0.25} & \frac{0}{1+0.25} & \frac{0}{1+0.25} & \frac{0}{1+0.25} & \frac{0}{1+0.25} & \frac{0}{1+0.25} & \frac{0.1}{1+0.25} & \frac{0.8}{1+0.25} & \frac{0.1+0.25}{1+0.25} \\
0 & 0 & 0 & 0.1 & 0 & 0 & 0.8 & 0.1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.1 & 0 & 0.1 & 0.8 \\
0.1 & 0.8 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\end{array}
\Longrightarrow \\
\\
\begin{array}{c} \\ a_{\mathrm{f}} \\ a_l \\ a_r \\ a_b \end{array}
\begin{array}{c}
\begin{array}{ccccccccc} x_{1,1} & x_{1,2} & x_{1,3} & x_{2,1} & x_{2,2} & x_{2,3} & x_{3,1} & x_{3,2} & x_{3,3} \end{array} \\
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0.08 & 0.64 & 0.28 \\
0 & 0 & 0 & 0.1 & 0 & 0 & 0.8 & 0.1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.1 & 0 & 0.1 & 0.8 \\
0.1 & 0.8 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\end{array}
\end{array}
\tag{4.3}
$$

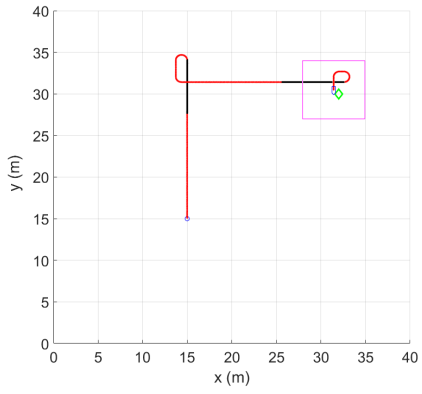
In Fig. 4.3 there are four sets of dynamics, one normal and three impaired trials. A start position in the bottom right is denoted as a blue circle and a goal as a green diamond. The first simulation shows the vehicle taking a normal right turn and proceeding to the goal. The second simulation in Fig. 4.3(b) has an internal failure with a restricted action space. The robot must solve the problem with no right action. The third in Fig. 4.3(c) also has a restricted action space with no right or back actions and must solve the problem with no right or back actions. In the robot's path, there are dark segments where the robot is attempting to go right or back but is forced to go forward; this occurrence is defined as a conflicted action. In Fig. 4.3(d), the dynamics are the same as in Fig. 4.3(c) but the learning rate β is reduced from 0.3 to 0.1. The model takes longer to determine that a left turn is the correct action; the first black line is longer for the smaller learning rate. Faster learning rates lead to quicker initial convergence to the true model estimations.



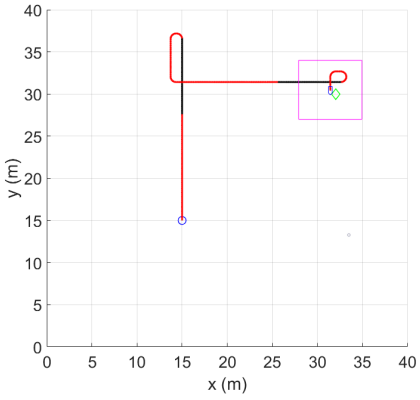
(a) Normal dynamics



(b) Impaired dynamics (no right) β=0.3



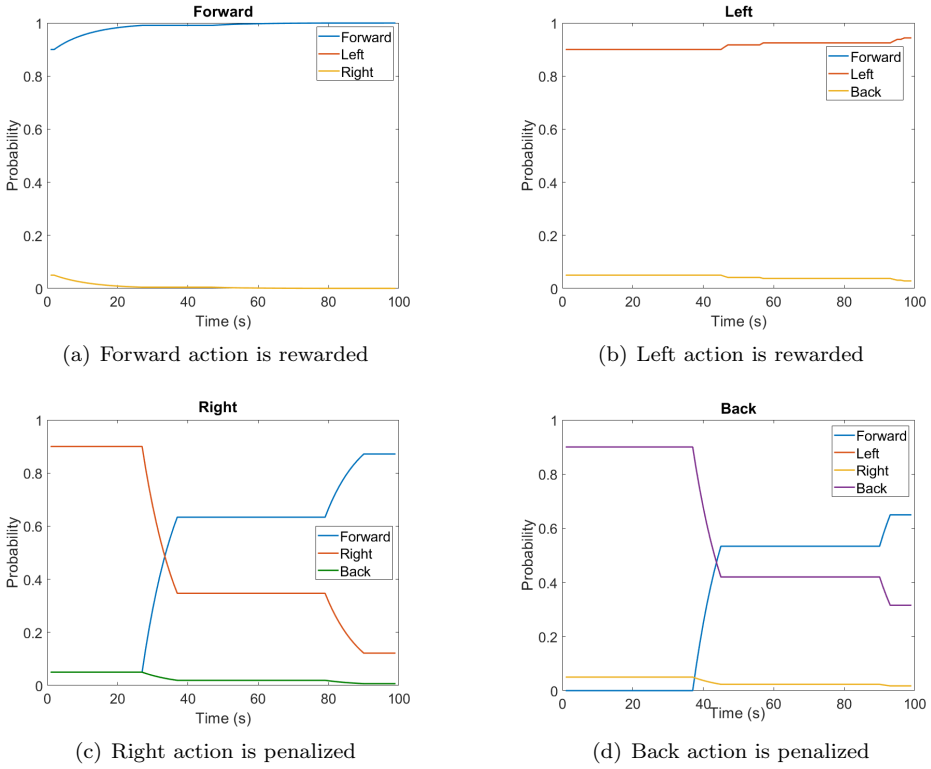
(c) Impaired dynamics (no right or back) with higher learning rate β=0.3



(d) Impaired dynamics (no right or back) with lower learning rate β=0.1

Figure 4.3: Simple dynamics comparison

The probability results are shown for each action in Fig. 4.4 for the third example in Fig. 4.3(c). The two declines in the forward probability correspond to these segments. We see that the first decline penalizes the probability more than the second due to the asymptotic nature of the equations in (4.1) and (4.2). The probability corresponding to the forward action is rewarded as the right probability is penalized in 4.4(c). The forward and left actions are rewarded while the back action also is shown to be compromised. As seen in Fig. 4.4, only one action's probability is updated at any given time, while the probabilities of the other actions remain constant.



(a) Forward action is rewarded

(b) Left action is rewarded

(c) Right action is penalized

(d) Back action is penalized

Figure 4.4: Transition probabilities for a system with no back or right actions as shown in Fig. 4.3

While we are testing the probabilities of an action by state matrix, the results are displayed in the probability distribution graphs by showing the probability that one action is performing like another. This decision was done to not oversaturate the probability graphs with every single potential state.

4.3 Bounding the Learning Rate

Due to the reward and normalization equations, repetitive rewards to the same probability for (a_o, x'_a) will result in asymptotic convergence to one. In (4.4), we analyze the probability as the same number of steps where n is the number of repeated rewards to the same probability. Equations (4.5) and (4.7) represent the

recursive sequence solutions to the limits in (4.4) and (4.6). As n approaches infinity, the second term goes to 0 and the output becomes 1 regardless of the initial condition h(0). Equation (4.7) shows the convergence to 0 as a probability is repetitively penalized. $h(n)$ is defined to represent the limit of $P(x'|x,a)_n$ as $n \to \infty$.

$$\lim_{n\to\infty} P(x'|x,a)_n = (P(x'|x,a)_{n-1} + \beta)/(1+\beta) \tag{4.4}$$

$$h(n) = 1 + (h(0) - 1)(1+\beta)^{-n} \tag{4.5}$$

$$\lim_{n\to\infty} P(x'|x,a)_n = (P(x'|x,a)_{n-1})/(1+\beta) \tag{4.6}$$

$$h(n) = h(0)(1+\beta)^{-n} \tag{4.7}$$

From the solutions in (4.5) and (4.7), the probabilities are bounded between 0 and 1. For ease of discussion let's simplify (4.1) and (4.2) with the following:

$$R(p) = (p+\beta)/(1+\beta) \tag{4.8}$$

$$N(p) = p/(1+\beta) \tag{4.9}$$

Let us consider the following notation that represents repetitive recursive functions for c total occurrences for a specific function.

$$f^{\circ c} = f \circ f^{\circ c-1} = f \circ \cdots \circ f \tag{4.10}$$

Consider a recursive series that has a constant pattern of rewards and penalties as $t \to \infty$. This pattern will create oscillations around the true transition probability estimates. K_r represents the number of rewards, and K_n represents the number of penalties. The total pattern length is $K_n + K_r$. As time approaches infinity, the minimum bound is calculated by $N^{\circ K_n}(R^{\circ K_r}(p))$, and the maximum bound is $R^{\circ K_r}(N^{\circ K_n}(p))$. For example, a pattern is approximating the probability of 2/3. This pattern can be simplified to two rewards and one penalty. The maximum of the oscillating series is found at the peak of the oscillations and the minimum is found at the trough. We find that the minimum is obtained by $N^{\circ 1}(R^{\circ 2}(p))$ because the value is checked after the penalty. The maximum is $R^{\circ 2}(N^{\circ 1}(p))$. To find the bounds of this recursive series, we take the limit to get it in terms of β, shown by:

$$\lim_{n\to\infty} p_{n+1} = \lim_{n\to\infty} R^{\circ 2}(N(p_n)) = \frac{\frac{p_n}{1+\beta} + \beta}{1+\beta} \tag{4.11}$$

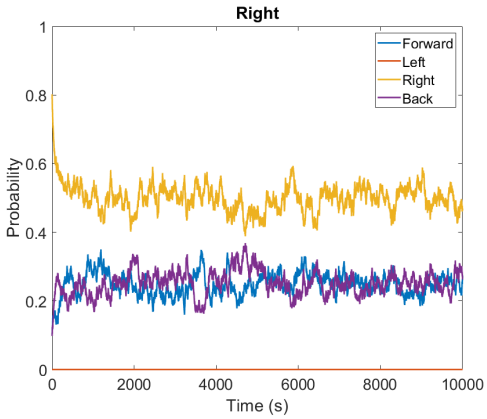
Since $\lim p_{n+1} = \lim p_n$, these variables can be substituted for each other resulting in the following for the maximum.

$$\lim_{n\to\infty} p_{n+1} = \frac{\beta^2 + 3\beta + 2}{\beta^2 + 3\beta + 3} \tag{4.12}$$

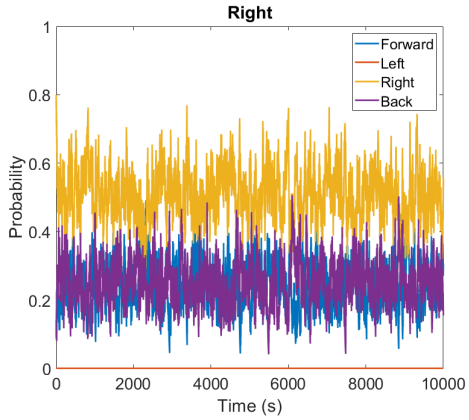
In (4.13), the minimum bound is found for the alternating recursive series.

$$\lim_{n\to\infty} p_{n+1} = \frac{\beta + 2}{\beta^2 + 3\beta + 3} \tag{4.13}$$

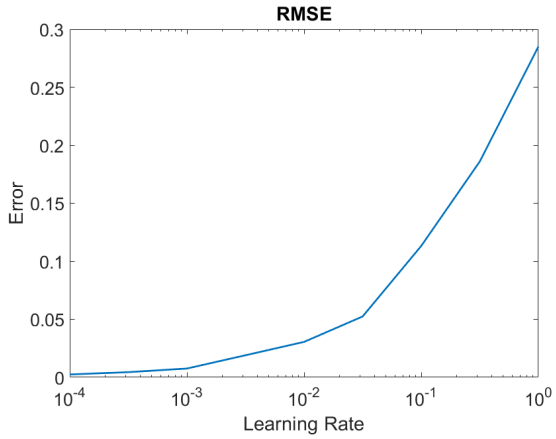
From the minimum and maximum equations, the dependence on beta is shown. At small β values, the maximum and minimum approach 2/3. With large β values, the max and minimum approach 1 and 0 respectively. An appropriate β value is chosen to balance quick probability adjustments and having low variance for the motion model estimation. This becomes a safety vs accuracy trade-off.



(a) Low variance $\beta = 0.01$



(b) High variance $\beta = 0.05$



(c) RMSE of different learning rates

Figure 4.5: β Testing

In Fig. 4.5(a) and (b), we evaluate the effects of different values of learning rates. The right action probability is set to a probability of 0.5 and the left and back actions are set to 0.25. A random action generator is used that picks actions based on those probabilities. The learning rate in (a) is set to 0.01, while β is set to 0.05 in (b). There is a considerable difference in noisy oscillations across the two simulations. In Fig. 4.5(c), the following learning rates are tested (0.0001, 0.000316, 0.001, 0.00316, 0.01, 0.0316, 0.1, 0.316, 1). A considerable increase in RMSE error increases as the learning rate increases. This further confirms the safety vs error trade-off for the transition probabilities of our motion model.

4.4 Dynamic Learning Rate

One purpose of a dynamic learning rate is to have a higher learning rate when our transition probabilities are very different from the recent actions. A faster learning rate will lead to recovering from a fault quicker as seen in Fig. 4.3. The more different two models are increases the utility of a higher learning rate. As the models converge, the model benefits from a lower learning rate because the model varies less as the estimation oscillates around the true model of the system. High static learning rates would result in oscillations around the optimal model parameters. Low learning rate parameters result in longer convergence time to changes within the action space

This leads to the second reason for the dynamic learning rate; the robot must be able to adapt the model quicker in potentially dangerous systems. Knowing when to make a turn earlier may prevent a collision with an obstacle. A higher dynamic learning rate is activated for these situations. We propose a χ^2-based dynamic learning rate for our framework [23].

$$\tilde{\chi}^2 = \frac{1}{d}\sum_{i=1}^{m}\frac{(O_i - E_i)^2}{E_i} \tag{4.14}$$

The χ^2 statistic is used for comparing the categorical data of the expected distributions to the observed distribution. As the χ^2 statistic is taken, this metric is used for comparing if the observed and expected distributions are different. O_i and E_i are the observed and expected data points, and d is the degrees of freedom. m represents the total samples. This is compared to a threshold H. The H threshold changes depending on safety constraints. The reduction of this threshold when safety is a concern allows for more sensitive detectors of a changing distribution. When the χ^2 term is greater than the threshold H, the learning rate β is increased by a scalar of χ^2.

In cases where the sample size is smaller, the Fisher's exact test [24] performs better than χ^2. Consider two groups that are being tested to be statistically different across two categories.

The Fisher's exact test is performed on smaller sample sizes for the system. In (4.15), Fisher's exact test is calculated using the variables in Table 4.1. Although, Fisher's exact test is valid for samples greater than 50, our framework switches to the χ^2 test when the sample size criterion is met. The χ^2 statistic is not used when the sample size is less than 50. This becomes relevant when the robot begins a simulation and has few samples for the current motion model estimation. Fisher's exact test is then used to test the difference between these models in safe and unsafe scenarios.

$$p = \frac{(c+d)!(e+\mathrm{f})!(c+e)!(d+\mathrm{f})!}{c!d!e!\mathrm{f}!n!} \tag{4.15}$$

Table 4.1: Fisher's Exact Test Table

	Group 1	Group 2
Category 1	c	d
Category 2	e	f

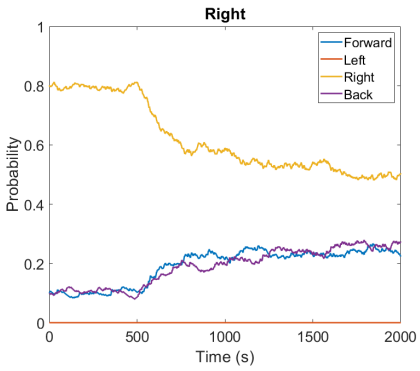
In Table 4.2, an example model based on training data and a model based on test data is shown. The χ^2 and Fisher's exact tests are computed on similar data to determine if a dynamic learning rate is needed. If the statistical test determines that the experimental and expected models are different, then the dynamic learning rate is activated.

Table 4.2: Testing Two Models

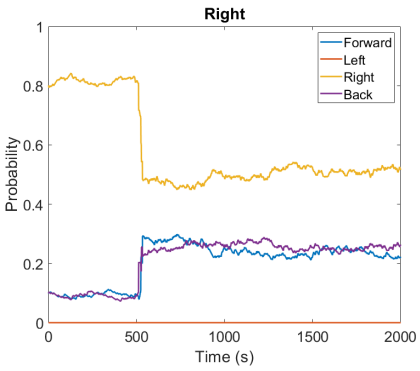
	Forward	Left	Right	Back
Training Model	995	252	251	2
Experimental Model	52	30	22	1

In Fig. 4.6, The effect of a dynamic learning rate is shown between (a) and (b). Much faster convergence happens with the dynamic learning rate. The results of the χ^2 test and the learning rate change rapidly as the internal model changes at timestep 500. The new probability of the right action changes from 0.8 to 0.5. The χ^2 statistic can quickly guide the model into changing quicker to avoid longer convergence times to the true model estimation.

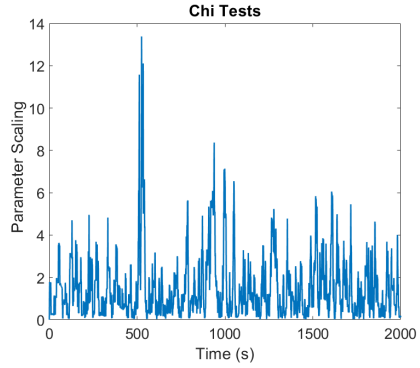
For the learning rate in (4.16), β is divided into two terms: the initialized learning rate β_I and the dynamic learning rate β_D. The initialized learning rate is the base on our learning rate and this generates the lower bound of our learning rate. The dynamic portion of the learning rate is calculated by a scalar multiplied by the χ^2 statistic. The unsafe criterion is met by getting within a certain distance of an obstacle.



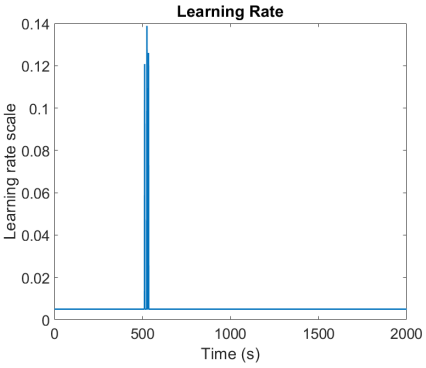
(a) β=0.005



(b) Dynamic learning rate



(c) Difference in probability distributions detected at t=500



(d) β increased at t=500

Figure 4.6: Effects of the dynamic learning rate

$$\begin{aligned} \beta &= \beta_I + \zeta * \chi^2 \textbf{ if } \chi^2 > H \\ \beta &= \beta_I \textbf{ if } \chi^2 < H \\ H &= H/2 \textbf{ if } \text{ unsafe} \end{aligned} \tag{4.16}$$

As the model converges to the recent actions through rewards and penalties, the learning rate will decrease. This creates a smoother convergence to the true transition probabilities. The dynamic learning rate will be deactivated when the experimental and expected models are not statistically different.

4.5 Reset Formulation from Active Perturbations

When a system's dynamics are only impaired temporarily, a state machine is proposed to test the initial and learned transition probabilities within the Markov decision process. This allows for the system to return to the old transition probabilities when it has discovered that the impaired dynamics no longer exist. Although

the robot can learn using the adapting framework, the robot may deem an action no longer viable and may no longer attempt it regardless if the problem has subsided. Therefore, a reset framework is needed to recover this old action space.

As depicted in Fig. 4.7, a state machine is used to determine how to reset these probabilities. τ is used as a trust factor for our state machine; this heuristic is increased when the robot begins trusting its model more. For this state machine, the robot has two MDPs running during every iteration of the simulation. Several conditions need to happen to trigger the reset to the old Markov decision process. A counter is used that increments up to τ. When the counter is greater than τ, the robot attempts an action if the optimal action prediction is different from the current model's optimal action. δ is used as the margin metric for comparing the difference between two probability distributions.

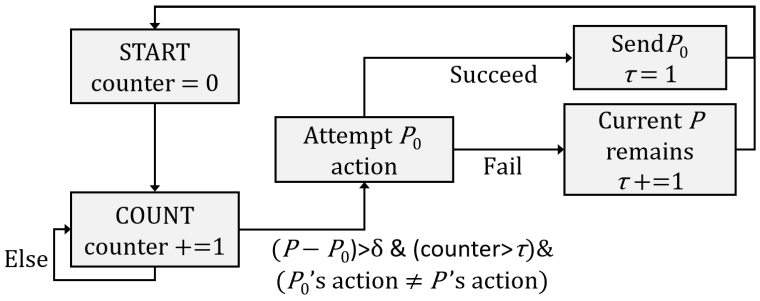


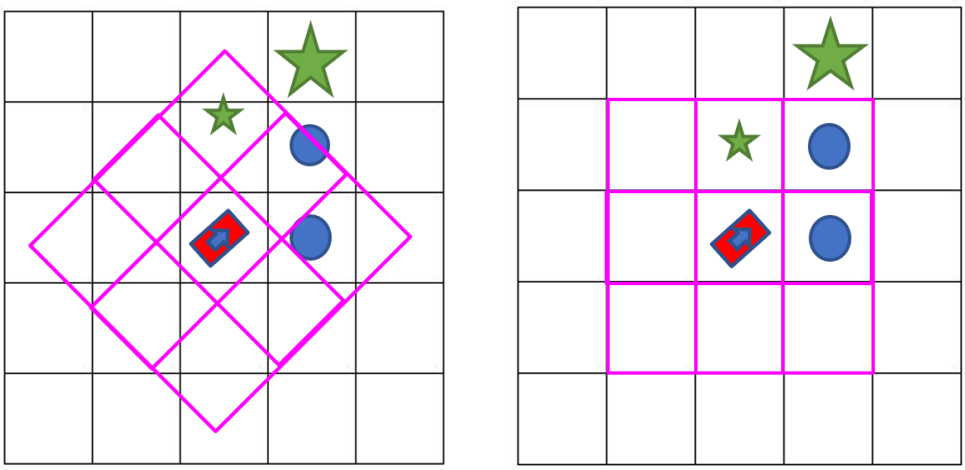
Figure 4.7: Reset state machine

Given an action, a series of transition probabilities correspond to each of the possible resulting states from that action. The original model generates the desired action; the probability distributions are then compared between the current and original models. If the difference is greater than δ, then the system tries to take the original model's action to check if the system is still impaired. If the original model's desired action is the same as the actual action, then the original model's probability distribution is replaced for that action using the distribution from the old model. When the action fails, τ is increased because the robot is trusting the current model and does not need to attempt the reset as frequently.

4.6 Detection of Local and Global Problems

Two possible types of disturbances are possible in real world applications: i) internal due to onboard failures like the loss of power in one motor and ii) external due for example to external forces like wind or change in terrain slope. In other cases the dynamics of a system can be cast as $\dot{x} = f(x, u, d)$ with $d \in \{d_i, d_e\}$ where d_i is the internal disturbance and d_e is the external disturbance.

To deal with both disturbances we introduce a dual MDP framework where the system has both a local and a fixed frame MDP. The purpose of having this dual MDP system is to be able to differentiate between internal and external problems. The combined system's goal is to accomplish the task regardless of the disturbances that act on the system. The local formulation solves internal problems. The system must also solve disturbances that result from the environment using the fixed MDP. In our approach, the local and fixed frame MDPs give the respective actions (Forward, Left, Right, Back) and (North, West, South, East). In Fig. 4.8, the local and fixed frame implementations are shown. The goal is shown outside of the view of the robot with a sub-goal placed inside of the MDP grid; the sub-goal's reward is propagated across the MDP grid. The robot makes movements typically toward this sub-goal using the respective actions from the local and fixed-frame approaches. The local frame pivots with the robot while the fixed frame does not.



(a) Local MDP (b) Fixed Frame MDP

Figure 4.8: MDP differences when detecting internal vs external problems

The fixed frame variation creates an MDP of size $A \times X_L \times O$ where O is the set of orientation of the system; n_O defines the total number of orientations. These orientations would be defined as different angular poses. The local variation is smaller with a size of $A \times X_L$. Adding higher resolution or more depth to the state or orientation matrices increases the size and computation complexity.

The dual MDP system must decide which policy to give to the controller of the robot between the local and fixed frame MDP. The $P(x'|x,a)$ matrix is stored by using buffers of size B where B << T (total iterations). The variance for each (a_k, x_i) pair in $P(x'|x,a)$ is calculated over the total stored iterations in the buffer. All of these variances are summed into one variance statistic. The total variance for the local and fixed frame models are defined as σ_L^2 and σ_F^2 respectively. These calculations can be seen in (4.17):

$$\begin{aligned} \sigma^2_{a_k,x_i} &= \frac{\sum_{l=1}^{B}\left(P_{(a_k,x_i)_l} - \overline{P_{(a_k,x_i)}}\right)^2}{B-1} \\ \sigma_L^2 &= \sum_{i=1}^{n_X}\sum_{k=1}^{n_A}\sigma^2_{a_k,x_i} \\ \sigma^2_{a_k,x_i,o_j} &= \frac{\sum_{l=1}^{B}\left(P_{(a_k,x_i,o_j)_l} - \overline{P_{(a_k,x_i,o_j)}}\right)^2}{B-1} \\ \sigma_F^2 &= \sum_{j=1}^{n_O}\sum_{i=1}^{n_X}\sum_{k=1}^{n_A}\sigma^2_{a_k,x_i,o_j} \end{aligned} \tag{4.17}$$

The system with the smaller total variance gives the policy. This method of determining which MDP is correct is based on the idea that the system with converging transition probabilities will have a smaller total variance due to reduced oscillations around model parameters. If the fixed frame MDP was used for an internal problem, the transition probabilities would continue changing as the actual state is different each time. This would cause an elevated variance for the remainder of the mission.

Chapter 5

Simulations and Experiments

5.1 Ground Robot Implementation

We apply the adaptive MDP to a real-world situation of a ground vehicle navigating around obstacles to reach a goal state. The environment is discretized into the state space X where there are n_X different states, and X_L is the local environment with n_L total local states within view, each cell representing a different state. The robot has n_A motion primitives where n_A is set to four in our simulations. The primitive actions for local MDP formulation are (Forward, Left, Right, and Back). The MDP calculates the expected reward of performing each action and chooses the optimal action. We utilize this adaptive MDP to account for when the system is impaired. Our MDP-based approach provides the framework to navigate its environment with a sub-optimal action space.

We present two possibilities for impair dynamics for our system. The first group of impaired system dynamics are caused by internal system problems and the second group are external or environmental problems. Some of the problems may be permanent or temporary. Our adaptive Markov decision process will learn how these dynamics change the transition probabilities.

The robot's dynamics are characterized as a non-holonomic vehicle with linear and angular velocity inputs defined as v and w. Linear and angular disturbance are defined as d_v and d_w. In (5.1), the following equations are used to represent the simulated robot.

$$\begin{bmatrix} \dot{x_R} \\ \dot{y_R} \\ \dot{\theta_R} \end{bmatrix} = \begin{bmatrix} \cos(\theta_R) & 0 \\ \sin(\theta_R) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v + d_v \\ w + d_w \end{bmatrix} \tag{5.1}$$

5.2 Software Stack

In Fig. 5.1, a summary of the implementation of our approach is shown. First, the goal state and obstacles are sensed within the local frame. The MDP is calculated and gives the optimal action to avoid obstacles while getting closer to the goal. The reset state machine is executed, and the robot is given a movement command. Based on the experimental state and action, $P(x'|x,a)$ is updated.

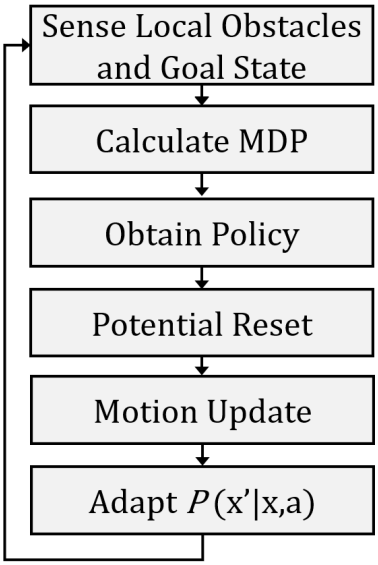


Figure 5.1: Software flow

5.3 Environment and Robot Primitives

An environment was created in MATLAB [25] for both simulations and real-world experiments. For most simulations, a (40x40) grid was used for the size of the map; this sets n_X to 1600 total states. Since we use local vision, we have a local view of (7x7) or (9x9) for the more complex dynamics. X_L becomes 49 or 81 total states. The local view is used since robots frequently have limited vision.

To reduce computation time, we use a local environment and have a vision depth value d_{vision} that determines how far out the robot analyzes the environment. This local environment is more practical from an application standpoint because the robot rarely will know the entire environment unless it has previously mapped the area. The robot has no map before the start of online learning. This decreases the computation time by a factor of $(width_{map}/1 + 2 * (d_{vision})^2$ and by 1600/49 for the (7x7) case. If an MDP on the full map was calculated at every iteration, the approach would lose its runtime viability.

The robot senses the obstacles in the environment and uses an occupancy grid for each obstacle when computing the MDP. The local grid is oriented around the robot as the center and the obstacles also occupy states. X_L contains the states within this local grid. We generate a local goal within the MDP. The model knows the location of the goal state, we make the closest point in the local grid to be the temporary goal. This grid point must not be an obstacle or loss point. Open-loop controllers are used and are impaired when disturbances arise.

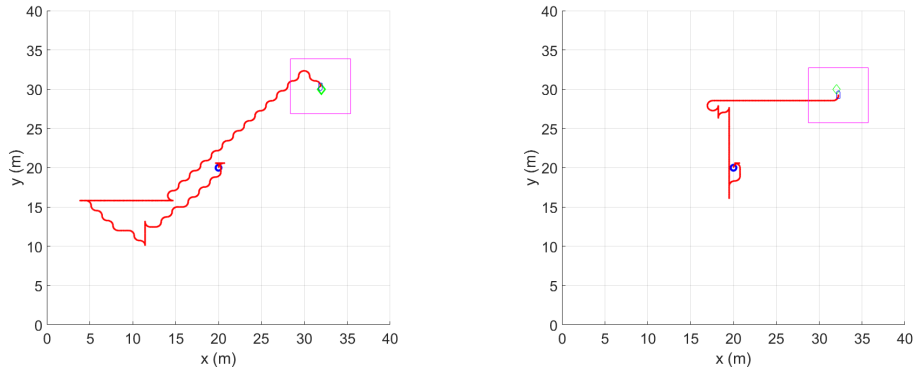
The goal points are given reward values of 1000 and obstacles are given values of 0. These are the only initialized values, the values of the initialized points cannot be changed but the values of the remaining states are updated over many iterations of the value function. The value iteration equation continues updating until the summed difference between the current and previous values changes by less than a predefined threshold. Our MDP is first calculated at the initialization of the experiment but also between every action because we are updating transition probabilities within the MDP.

5.4 Confused Action Convergence

For these simulations, the action primitives are confused. Forward and back are swapped, and left and right are swapped initially. The transition probabilities are initialized to reflect this confusion. As the robot begins moving, it attempts the action that gives the highest reward. We analyze this with two different learning rates $\beta = 0.1$ and $\beta = 0.2$.

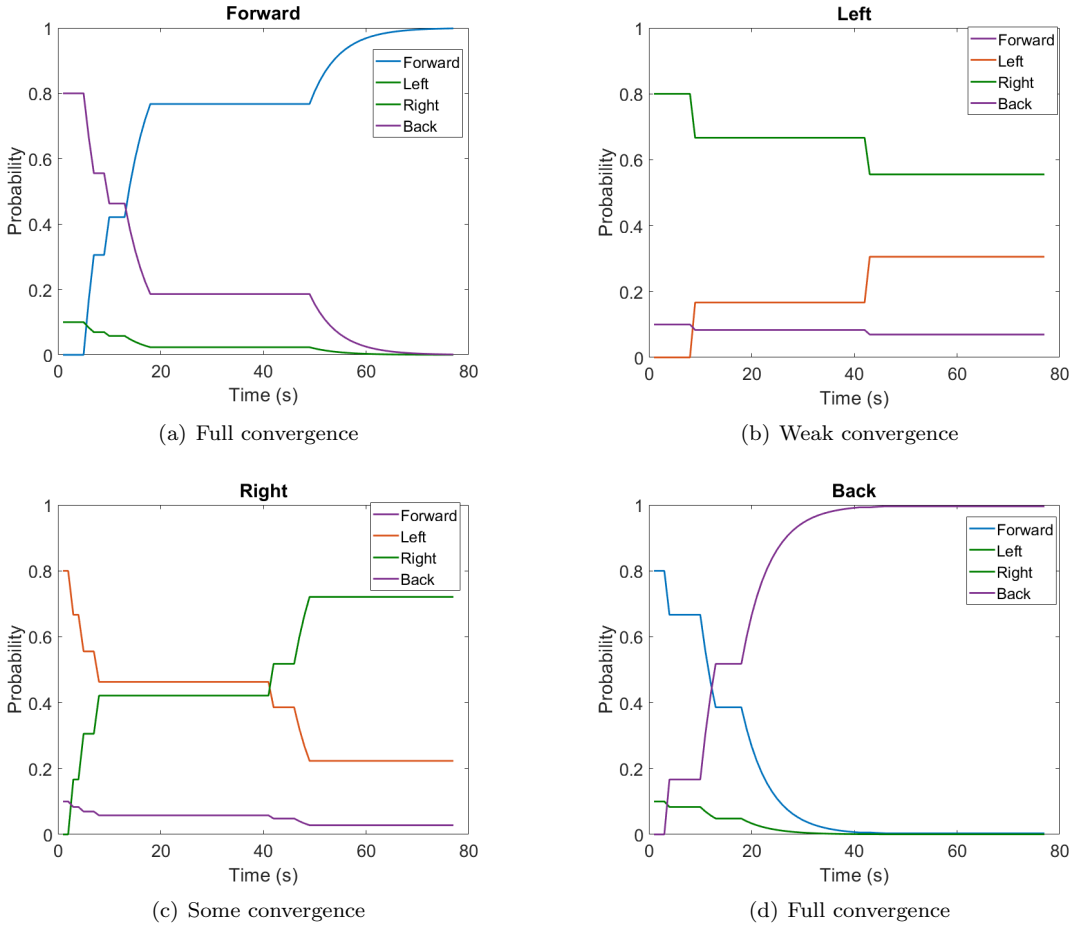
Fig. 5.2(a) contains the result from a confused simulation with $\beta = 0.1$. Due to the lower learning rate, the simulation will take longer to find the goal and for the four transition probabilities to converge closer to the true actions. From the delayed convergence of the forward action, left and right work together to make the forward action and to reach the goal; these actions can be seen converging in Fig. 5.4(b) and (c). The right and left actions are able to switch back and converge to the true probabilities for those actions. The forward probability does not even reach 0.5 of the true probability in Fig. 5.4(a).

In the second simulation in Fig. 5.2, the MDP has a higher learning rate and allows for a much quicker convergence to the solution. Left is the only action that does not get close to the convergence of the true actions as seen in Fig. 5.3(b).



(a) $\beta = 0.1$ (b) $\beta = 0.2$

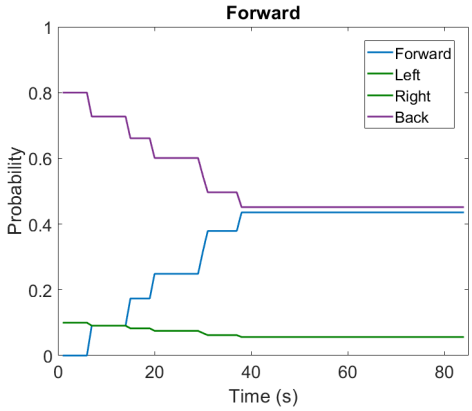
Figure 5.2: Confused action trajectories



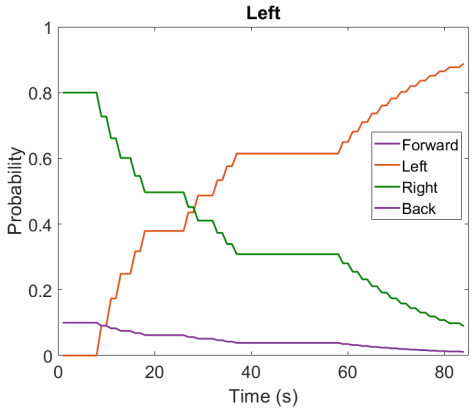
(a) Full convergence (b) Weak convergence

(c) Some convergence (d) Full convergence

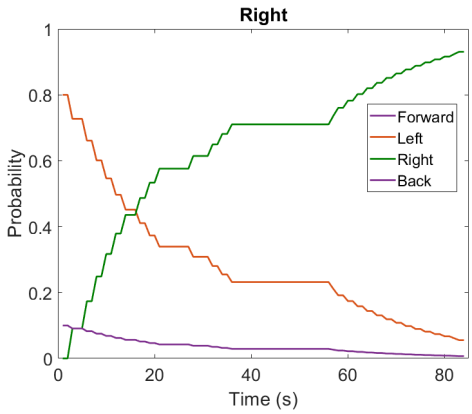
Figure 5.3: Confused action probabilities $\beta = 0.2$



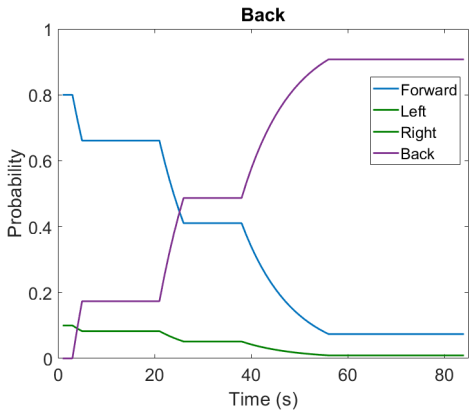
(a) Weak convergence



(b) Almost full convergence



(c) Almost full convergence

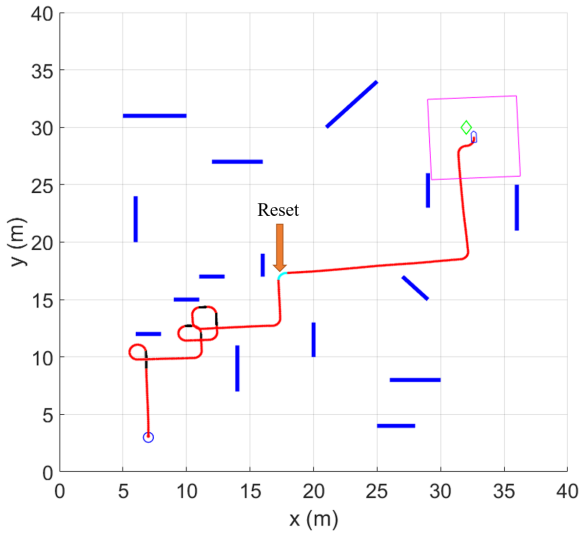


(d) Almost full convergence

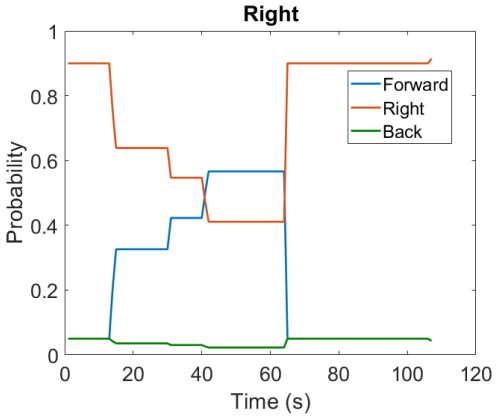
Figure 5.4: Confused action probabilities $\beta = 0.1$

5.5 Testing the Reset State Machine on a Robot with a Temporary Fault

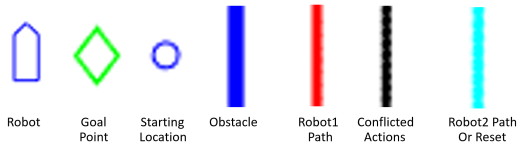
In Fig. 5.5 and Fig. 5.6, a simulation is produced using the simple action primitives with an internal problem and the reset mechanism enabled. Small amounts of noise were added to the linear and angular velocity of the robot. The cyan color represents the segment of the simulation where the robot is reset to the original dynamics. In Fig. 5.5(a), The robot performs several loops to create a left action and is eventually reset roughly halfway to the goal because the disturbance is not present anymore at that stage. In Fig. 5.6(a), the robot takes an alternative path, with a reset when the robot succeeds in its first right turn. The trust factor τ set to 10 in the following simulations. In both simulations, the section colored cyan corresponds with the reset to original dynamics in the probability graph. While several reset attempts were made prior, the only successful one is shown.



(a) Trajectory (reset in cyan)



(b) Right probability

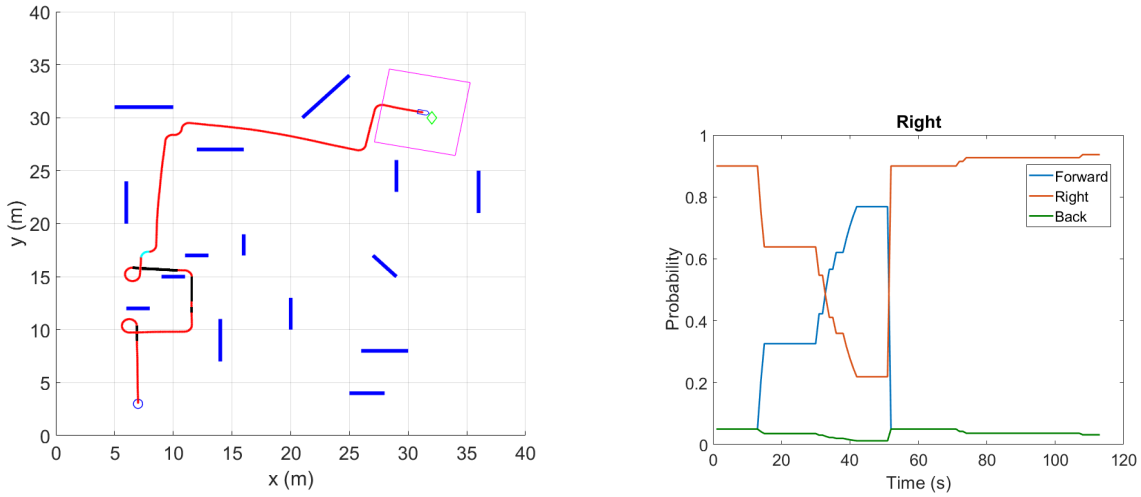


(c) Legend

Figure 5.5: Reset simulation with noise

In Fig. 5.5(c), a universal legend is shown. For this experiment, only one robot is shown. Cyan is used for resets and for the path of additional robots.

In Fig. 5.5(b) and Fig. 5.6(b), both right transition probabilities are reset at the same time step as their respective trajectory graphs. They both return to the original models where the right action was enabled. While the robot could not go right earlier in the simulations, the model was reset, and the right action was utilized.



(a) Trajectory (reset in cyan)

(b) Right probability

Figure 5.6: Additional reset simulation with noise

5.6 Simulations for Detecting Internal and External Problems

In Fig. 5.7, a robot is enabled with enhanced dynamics, a slightly different optimal action calculation, and more complex disturbances and failures. The dynamics differ by having the controller turn in 45° increments compared to the more simple increments of 90°. An open-loop controller is used to make a series of small angled increments that sum up to 45°. In Fig. 5.7(a), a robot is seen completing two tasks with no disturbances. We enact a disturbance on the system by adding to the angular velocity of the system. The two disturbance equations are shown below:

An internal disturbance is modeled by altering the angular velocity term in (5.1) as:

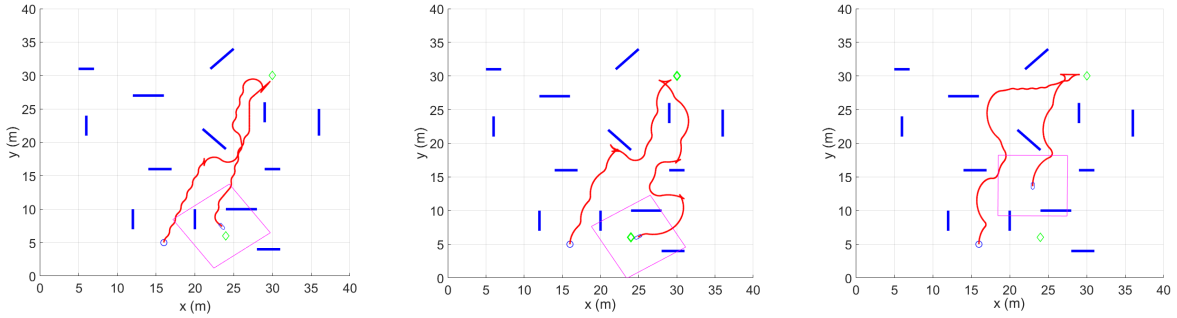
$$\dot{\theta_R} = \omega + d_i \tag{5.2}$$

While an external disturbance is modeled by altering the linear velocity terms in (5.1) as:

$$\begin{aligned} \dot{x_R} &= v\cos(\theta_R) + \gamma d_e \cos(\xi) \\ \dot{y_R} &= v\sin(\theta_R) + \gamma d_e \sin(\xi) \end{aligned} \tag{5.3}$$

In (5.2), the disturbance is created by simply adding a static drift factor d_i to the angular velocity of the robot. As it can be noted, in the trajectory in Fig. 5.7(b), the robot drifts to the left when attempting to go straight from the impaired actions. The robot eventually makes it to both goals but must learn how to navigate while drifting to the left.

(5.3) represents the equation of motion affected by an external disturbance like wind with d_e the external disturbance magnitude, ξ the direction of the disturbance, and γ a weight factor. Fig. 5.7(c) shows an example for a wind external disturbance blowing from east to west (i.e., $\xi = \pi$ rad). As the robot becomes perpendicular to the wind, the angular disturbance is maximized, and the vehicle gains or loses velocity depending on whether it is facing toward or away from the wind. The robot is able to differentiate between internal and external disturbances and reach the goal using the formulation from Section 4.6 as demonstrated in Fig. 5.7(b) and Fig. 5.7(c). In all three simulations, a small amount of noise was added to the linear and angular velocity of the robot to make the simulations more realistic.



(a) No problem

(b) Internal fault

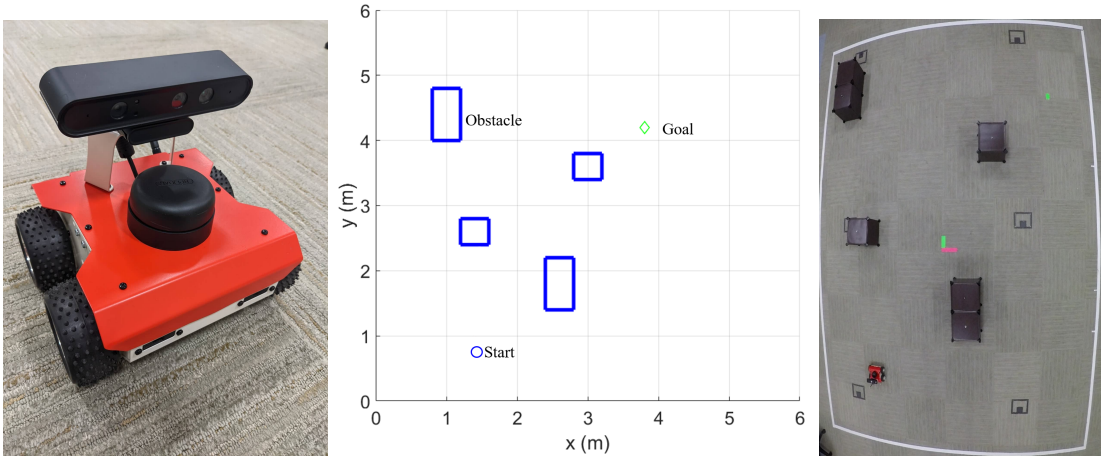
(c) External disturbance (wind blowing from east to west)

Figure 5.7: Robot paths with complex dynamics

5.7 Real-World Experimental Results

In the following section, experimental results on a real vehicle are presented in a lab-controlled environment. For the following experiments, we use the ROSbot which is a skid-steering differential drive unmanned ground vehicle with dynamics from (3.3). Everything was implemented under the robot operating systems (ROS) framework [26]. ROS uses a publisher and subscriber model. Publishers send commands to the robot, and subscribers receive sensor data from the robot. A total of three computers were used to run these experiments: one on the robot, the VICON motion capture system computer for ground truth data, and the base-station computer which sends instructions to the ROSbot. The robot is tracked through the VICON system, which entails eight cameras capable of sub-millimeter positional accuracy. Four reflective markers are placed on the robot to track the position and orientation of the robot; an object is created within VICON software to allow for 3D visualization of the robot. The position and orientation are subscribed from the base computer.

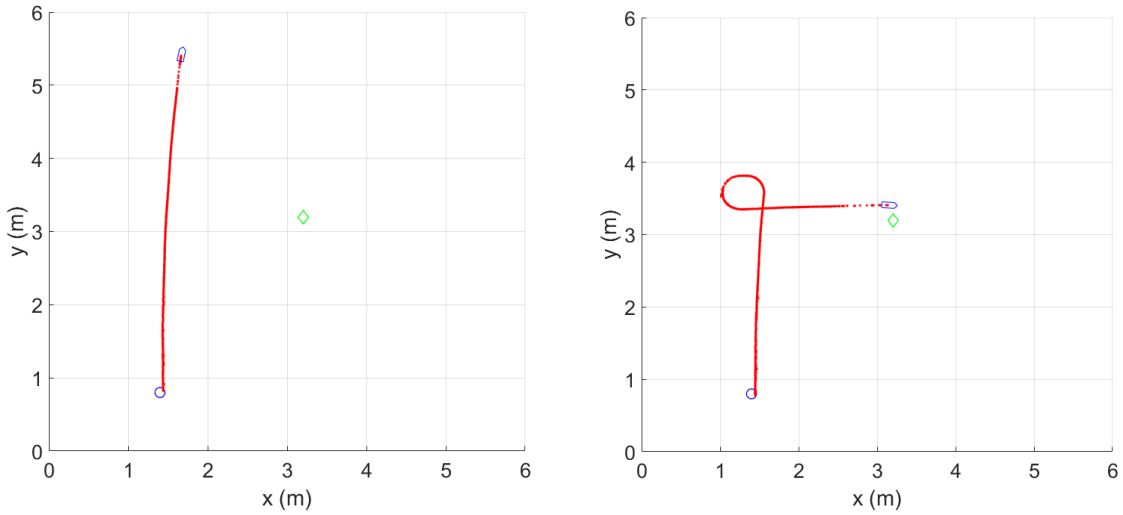
The positions of the real-world obstacles match the positions in the simulated environment. In Fig. 5.8, examples of the simulated and real environments are shown. Four obstacles are used for the robot to navigate around as it goes from the current location of the robot to the green goal point.



(a) Rosbot (b) Simulation Map (c) Camera View

Figure 5.8: Real-world environment

First, an example is shown in Fig. 5.9(a) when a robot is impaired in an environment with no obstacles and does not use the adapting framework but simply uses a regular MDP to find its optimal policy. The right and back actions are removed and result in states achieved by a forward action. Although our robot is impaired by limiting that action via software, a similar real-world condition would occur if the angular velocity of one side of wheels could not exceed the other side. In Fig. 5.9(b), the robot adapts while having impaired dynamics, and the right and back transition probabilities are adapted as these actions are shown to be defective.

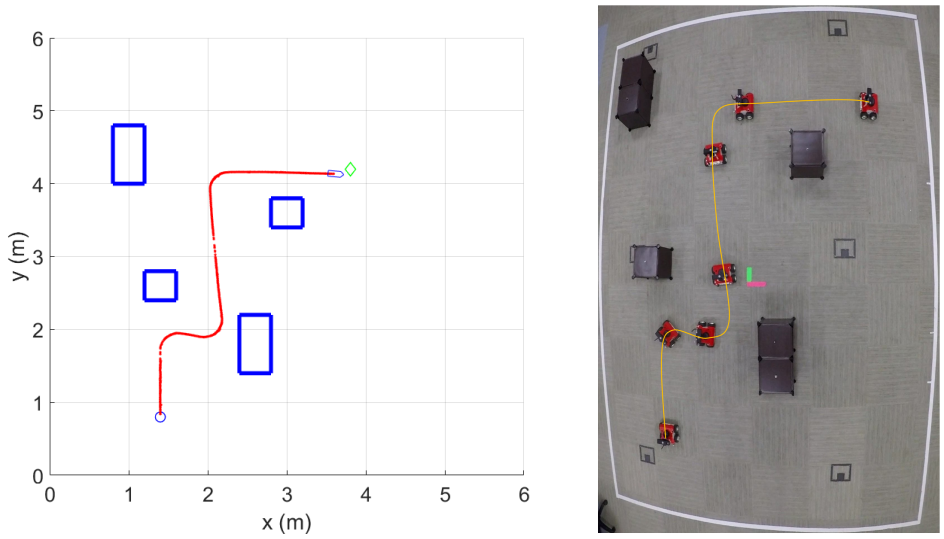


(a) Traditional MDP

(b) Our Approach

Figure 5.9: No obstacles with internal fault

If we consider an environment with obstacles now, a normal robot trajectory is depicted in Fig. 5.10. The robot will use the MDP to plan its path safely through the environment to the goal. Fig. 5.10(b) shows an overlapped sequence of poses of the UGV as it navigates to the goal.

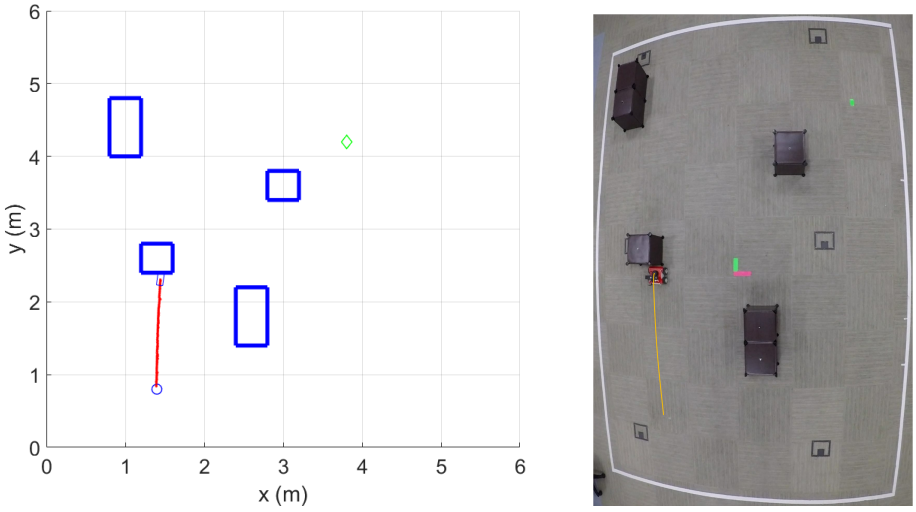


(a) Simulation map

(b) Camera view

Figure 5.10: Nominal path as computed using a traditional MDP policy

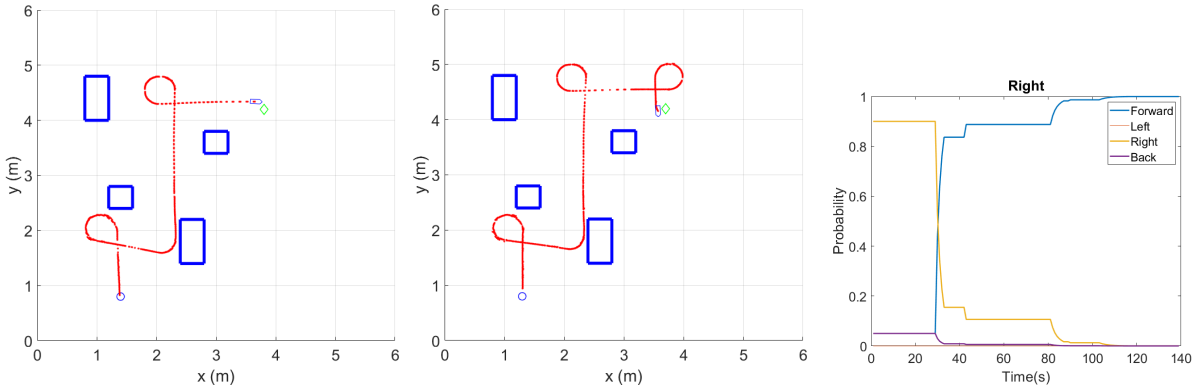
In Fig. 5.11, an experiment is run without the adapting framework while the dynamics are impaired removing the back and right actions. As expected, the robot runs into the first obstacle and fails to reach the goal.



(a) Simulation map (b) Camera view

Figure 5.11: An impaired robot using traditional MDP

Fig. 5.12 shows two runs of the same experiment with our proposed approach with β set to 0.3. The robot is able to adapt robustly in all tests that we run and complete the task, making it to the goal without colliding with any obstacles. The robot can detect and change the transition probabilities of the back and right action. The first initial drop of the right probability is shortly before the first left turning loop. Fig. 5.12(c) is the resulting transition probability graph for Fig. 5.12(a). Additionally, the robot takes a variety of left loops depending on the experiment trial.



(a) Path for impaired system: trial 1 (b) Path for impaired system: trial 2 (c) Right action probabilities from trial 1

Figure 5.12: Real-world experiment with an impaired robot using our framework

Fig. 5.13, shows an image of overlapped snapshots recorded during the experiment presented in Fig. 5.12(a)

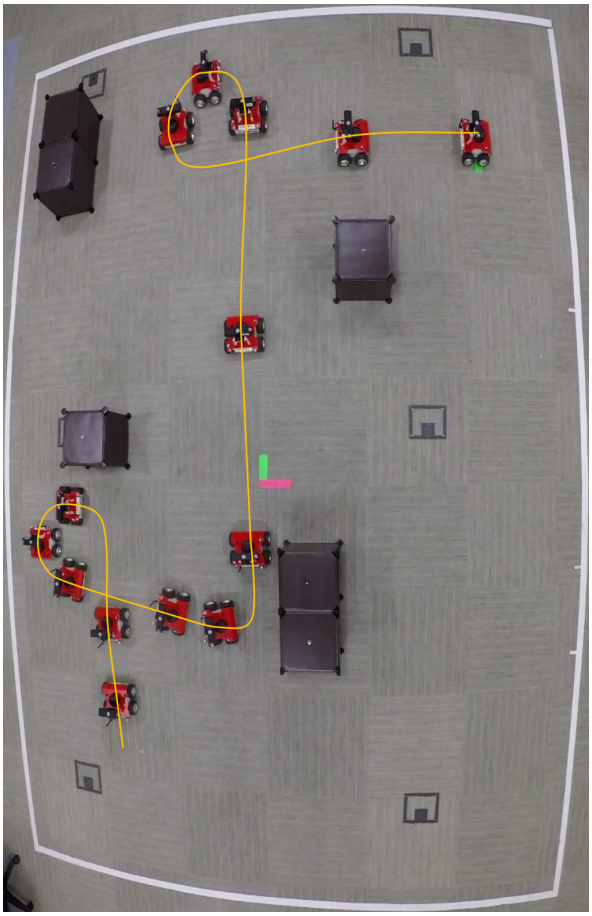


Figure 5.13: Overlapped snapshots for the impaired trial 1 experiment in Fig. 5.12(a)

Chapter 6

Multi-vehicle Cooperative Task Management

6.1 Formulation

In the previous chapters, all the framework and simulations were with a single robot. In this section, we show an extension of our proposed work to deal with cooperative missions where one or more vehicles are impaired. Let's consider a set Q of n_Q robots q with $q = 1 \dots n_Q$. The problem we are interested to solve here is how to best coordinate multiple robots with different dynamics to accomplish a series of tasks that in our case consist of reaching different goal states. This is very similar to the traveling salesman problem where a single agent must visit a given number of vertices in an optimal order [27]. These goal states are defined as $X_G = \{x_{g_1}, \dots, x_{g_{n_g}}\} \in X$ where n_G is the total number of goal states. When solving this problem, the robots must coordinate with each robot to accomplish the set of tasks X_G from their starting states x_{s_q}. Each robot is then given a precomputed efficiency metric E_q that is defined as the rate *travelled distance/time* for each robot navigating through a cluttered environment. This metric is calculated from several simulations using a normal and impaired robot; their average E is calculated by the *travelled distance/time* over each simulation. A higher rating allows that robots to take on more tasks. A set of goal states X_{G_q}, will be generated for each robot q based on the starting states, goal states, and efficiency rating to find the minimum time t_c to complete the tasks. The sets of the states in each X_{G_q} are ordered sets for the purpose of finding the optimal sequence of tasks. The total completion time depends on the worst case scenario that is the longest completion time for any given robot's X_{G_q} assignment. $dist(X_{G_q})$ is the total distance over the

ordered set X_{G_q}. The problem is defined as follows:

$$\begin{aligned} \min_{X_{G_1}, X_{G_2}, \ldots, X_{G_{n_Q}}} \quad & t_c = \max\left(\frac{dist(x_{s_1}, X_{G_1})}{E_1}, \frac{dist(x_{s_2}, X_{G_2})}{E_2}, \ldots, \frac{dist(x_{s_{n_Q}}, X_{G_{n_Q}})}{E_{n_Q}}\right) \\ \text{s.t.} \quad & X_G = X_{G_1} + X_{G_2} + \ldots + X_{G_{n_Q}} \\ & X_{G_1} \cap X_{G_2} \cap \ldots \cap X_{G_{n_Q}} = \varnothing \end{aligned} \tag{6.1}$$

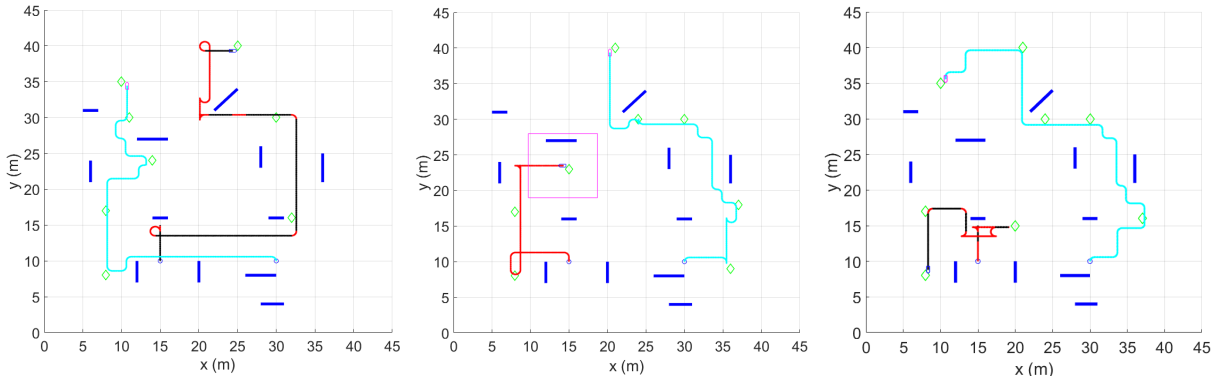
Our algorithm iterates over $\sum_{i=0}^{n_G} \frac{n_G!}{(n_G - i)!}$ permutations of the possible ordered sets and pairs these with the starting states to find the optimal completion time given the distance over each trajectory and each efficiency rating. Note that this algorithm becomes computationally expensive as the number of goals and vehicles grows since it is an exhaustive search. For simplicity, consider an example with two robots and three goals. In Table 6.1, all the combinations are shown for each robot task assignment. From these combinations, every permutation is used to calculate the minimal distance from the starting point. When the minimum t_c is found, the tasks for each robot are assigned from the corresponding permutation.

Table 6.1: Task Assignment Combinations

Combinations of Goals	Robot 1	Robot 2
0 and 3	None	$(x_{g_1}, x_{g_2}, x_{g_3})$
1 and 2	(x_{g_1})	(x_{g_2}, x_{g_3})
1 and 2	(x_{g_2})	(x_{g_1}, x_{g_3})
1 and 2	(x_{g_3})	(x_{g_1}, x_{g_2})
2 and 1	(x_{g_1}, x_{g_2})	(x_{g_3})
2 and 1	(x_{g_1}, x_{g_3})	(x_{g_2})
2 and 1	(x_{g_2}, x_{g_3})	(x_{g_1})
3 and 0	$(x_{g_1}, x_{g_2}, x_{g_3})$	None

6.2 Simulations

In the following simulation, we consider two robots in the set Q, in which one experiences some failure. The normal robot has full access to its action space while the impaired robot cannot turn back or right. The impaired robot has a red trajectory while the robot with normal dynamics is in cyan. Between two robots, eight goal states must be completed. In Fig. 6.1(a), all 8 goal states are completed while giving the impaired robots only 3 tasks to perform. The impaired robot takes longer due to the initial learning in front of the first obstacle. This robot will take longer to travel distance due to the need to perform a non-optimal series of actions to emulate lost actions. The normal robot takes 84 iterations while the impaired robot takes 125 iterations to complete its tasks.



(a) (b) (c)

Figure 6.1: Independent multi-vehicle trials with an impaired robot

In Fig. 6.1(b) and (c), the impaired robot is assigned only three tasks but these tasks are much closer to the starting position than in Fig. 6.1(a). In Fig. 6.1(b), the impaired robot finishes in 66 iterations while the normal robot finishes in 82 iterations. The results in Fig. 6.1(c) are similar: 61 iterations for the impaired robot and 96 iterations for the normal robot. The proximity of the goal states near the impaired robot allows for much faster completion times than in Fig. 6.1(a).

In this implementation we have considered two vehicles tasked to visit a small number of goals. Future work involves finding a faster heuristic to find optimal trajectories considering larger number of vehicles and goals.

Chapter 7

Conclusions and Future Work

7.1 Conclusion

In this thesis, we have demonstrated an adaptive online approach for trajectory planning when disturbances and internal failures arise. Our solution utilizes the MDP formulation to estimate optimal actions for the robot while discovering faults and disturbances. The basic principle behind our approach is that MDP leverages transition probabilities to compute an optimal policy and thus updating these probabilities at runtime will enable the discovery of new policies for changing dynamical systems. As shown in the results, the framework is versatile for internal and external problems and confused action situations. Our approach utilizes a dynamic learning rate for quicker learning in motion model estimates while smoothly converging to the true transition probabilities. We validated our approach through single robot simulations and experiments. A solution for multi-robot coordination was displayed through several simulations.

In comparison to many reinforcement learning implementations, our method works quickly at runtime and requires little overhead for the initialization. Over just a few iterations, the robot is able to detect faults and recover to still be able to complete its task. The robot is also able to have quick adaptations through our local approach.

7.2 Future Work

Future work will involve more multi-robot environments by increasing the number of robots and tasks to solve. A better solution to multitask management will also be explored, for example, considering Hungarian algorithm-based strategies [28] or other heurstics like the Lin–Kernighan (LK) approach [29] to solve the

traveling salesman problem. This adaptable framework can also be applied to higher dimensional state and action spaces targeting for example unmanned aerial vehicles for which much quicker reaction times are needed. Finally, other reinforcement learning or trial-and-error implementations can be tested against our approach. Our approach could work in coordination with a reinforcement learning algorithm for quicker recovery while learning more complex dynamical models. A final future direction would be to include heterogeneous robotic systems to actively adjust task assignments based on disturbances like changes in dynamics and environmental conditions.

Bibliography

[1] K. Chatzilygeroudis, V. Vassiliades, and J. Mouret, "Reset-free trialand-error learning for data-efficient robot damage recovery," *arXiv preprint arXiv:1610.04213*, 2016.

[2] J. Lee, J. Hwangbo, and M. Hutter, "Robust recovery controller for a quadrupedal robot using deep reinforcement learning," *arXiv preprint arXiv:1901.07517*, 2019.

[3] K. Chatzilygeroudis, V. Vassiliades, and J.-B. Mouret, "Reset-free trial-and-error learning for robot damage recovery," *Robotics and Autonomous Systems*, vol. 100, pp. 236–250, 2018.

[4] M. Focchi, V. Barasuol, M. Frigerio, D. G. Caldwell, and C. Semini, "Slip detection and recovery for quadruped robots," in *Robotics Research*, Springer, 2018, pp. 185–199.

[5] A. Mosavi and A. Varkonyi, "Learning in robotics," *International Journal of Computer Applications*, vol. 157, no. 1, pp. 8–11, 2017.

[6] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[7] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[8] S. Koos, J.-B. Mouret, and S. Doncieux, "The transferability approach: Crossing the reality gap in evolutionary robotics," *Evolutionary Computation, IEEE Transactions on*, vol. 17, pp. 122–145, Feb. 2013.

[9] K. Chatzilygeroudis, R. Rama, R. Kaushik, D. Goepp, V. Vassiliades, and J.-B. Mouret, "Black-box data-efficient policy search for robotics," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017, pp. 51–58.

[10] S. Koos, A. Cully, and J. Mouret, "Fast damage recovery in robotics with the t-resilience algorithm," *CoRR*, vol. abs/1302.0386, 2013.

[11] G. Glaubit, K. Kleeman, N. Law, *et al.*, "Fast, safe, and proactive runtime planning and control of autonomous ground vehicles in changing environments," in *2021 Systems and Information Engineering Design Symposium (SIEDS)*, IEEE, 2021, pp. 1–6.

[12] E. Yel and N. Bezzo, "A meta-learning-based trajectory tracking framework for uavs under degraded conditions," *arXiv preprint arXiv:2104.15081*, 2021.

[13] B. Durand, K. Godary-Dejean, L. Lapierre, R. Passama, and D. Crestani, "Using adaptive control architecture to enhance mobile robot reliability," in *TAROS: Towards Autonomous Robotic Systems*, 2010, pp. 54–61.

[14] D. A. Pereira, A. Al-Dujaili, M. E. B. El Najjar, V. Cocquempot, and Y. Ma, "Actuator fault estimation and fault tolerant control in three physically-linked 2wd mobile robots," *IFAC-PapersOnLine*, vol. 51, no. 24, pp. 709–716, 2018.

[15] J. O'Keeffe, D. Tarapore, A. G. Millard, and J. Timmis, "Towards fault diagnosis in robot swarms: An online behaviour characterisation approach," in *Annual Conference Towards Autonomous Robotic Systems*, Springer, 2017, pp. 393–407.

[16] K. Kinjo, E. Uchibe, and K. Doya, "Evaluation of linearly solvable markov decision process with dynamic model learning in a mobile robot navigation task," *Frontiers in neurorobotics*, vol. 7, p. 7, 2013.

[17] N. Gopalan, M. L. Littman, J. MacGlashan, *et al.*, "Planning with abstract markov decision processes," in *Twenty-Seventh International Conference on Automated Planning and Scheduling*, 2017.

[18] V. Verma, J. Fernandez, R. Simmons, and R. Chatila, "Probabilistic models for monitoring and fault diagnosis," in *The Second IARP and IEEE/RAS Joint Workshop on Technical Challenges for Dependable Robots in Human Environments. Ed. Raja Chatila*, 2002.

[19] K. A. Svendsen and M. L. Seto, "Partially observable markov decision processes for fault management in autonomous underwater vehicles," in *2020 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, IEEE, 2020, pp. 1–7.

[20] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, no. 7553, pp. 503–507, 2015.

[21] M. Van Otterlo, "Markov decision processes: Concepts and algorithms," *Course on 'Learning and Reasoning*, 2009.

[22] G. Fu and X. Zhang, "Rosbot: A low-cost autonomous social robot," in *2015 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, IEEE, 2015, pp. 1789–1794.

[23] P. E. Greenwood and M. S. Nikulin, *A guide to chi-squared testing*. John Wiley & Sons, 1996, vol. 280.

[24] D. B. Rubin, "Randomization analysis of experimental data: The fisher randomization test comment," *Journal of the American Statistical Association*, vol. 75, no. 371, pp. 591–593, 1980.

[25] MATLAB, *Version 7.10.0 (r2018a)*, Natick, Massachusetts, 2018.

[26] Stanford Artificial Intelligence Laboratory et al., *Robotic operating system*, version ROS Melodic Morenia, May 23, 2018.

[27] C. Dahiya and S. Sangwan, "Literature review on travelling salesman problem," *International Journal of Research*, vol. 5, pp. 1152–1155, 2018.

[28] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[29] C. Rego, D. Gamboa, F. Glover, and C. Osterman, "Traveling salesman problem heuristics: Leading methods, implementations and latest advances," *European Journal of Operational Research*, vol. 211, no. 3, pp. 427–441, 2011.