REINFORCEMENT LEARNING APPLIED TO MOBILE ROBOT NAVIGATION

A Project Report

submitted by

NAVYATA SANGHVI

(ME11B149)

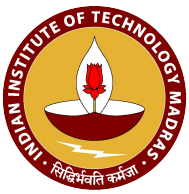
in partial fulfilment of the requirements

for the award of the degree of

BACHELOR OF TECHNOLOGY

in

MECHANICAL ENGINEERING



PRECISION ENGINEERING AND INSTRUMENTATION LABORATORY

DEPARTMENT OF MECHANICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

MAY 2015

PROJECT CERTIFICATE

This is to certify that the project titled **Reinforcement Learning Applied to Mobile Robot Navigation**, submitted by **Navyata Sanghvi (ME11B149)**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology** in **Mechanical Engineering**, is a bonafide record of the project work done by him in the Department of Mechanical Engineering, IIT Madras. The contents of this report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. P.V. Manivannan
Project Guide
Assistant Professor
Dept. of Mechanical Engineering
IIT Madras, Chennai 600 036

Prof. B.V.S.S. Prasad
Head of Department
Dept. of Mechanical Engineering
IIT Madras, Chennai 600 036

Dr. B. Ravindran
Project Guide
Associate Professor
Dept. of Computer Science
IIT Madras, Chennai 600 036

Place: Chennai

Date: May 19, 2015

ACKNOWLEDGEMENTS

First and foremost, thanks to Dr. P.V. Manivannan for his constant guidance and sound advice. The PEIL lab and P3-DX robots were always made accessible to me for my experimental forays into this field, and his encouraging and insightful presence at each of our discussions, reviews and presentations are things I am highly grateful for. He truly enriched my final year project experience to make it substantial and fulfilling.

Next, I would like to thank Dr. Balaraman Ravindran for his constant guidance and teaching in the fields of machine and reinforcement learning (RL). He encouraged me to delve deeper into these fields, giving me confidence in my abilities. I will always be grateful for his open and collaborative nature, which kept me highly motivated while pursuing this RL-based navigation project.

I would like to thank Dr.Manivannan's PhD student Suresh Perumbure, for guiding me throughout, and helping me with all my experimental setups. His imaginative and practical mind called for several refreshing discussions at the PEIL lab, and always left my mind buzzing with new possibilities and ideas to further my project experience. His encouragement, while I was still unsure of pursuing a field outside of my comfort zone, was crucial to my decision of going for it.

I would also like to thank Dr. Ravindran's student, Dhanvin Mehta, for spending countless hours discussing and advising me about my project and this field that I was completely new to. His guidance and encouraging spirit was crucial on every step of the way while I learnt about the novel techniques of learning and decision-making which RL had to offer. With his experienced mind, he was a sound teacher throughout.

Lastly, thanks to my friends and family, whose constant presence and support helped me throughout the project.

ABSTRACT

Reinforcement learning is a sub-branch of machine learning which involves teaching an agent to act rationally in an unknown environment. This is accomplished by learning a policy - mapping situations to actions - so as to maximize a numerical reward signal that it receives from the environment as feedback. With its wide applications in control, optimization and robotics, reinforcement learning has mounting potential in mechanical engineering to design single and multi-agent systems with desired behaviours. Manufacturing processes, propulsion systems, cruise controllers in automobiles and multi-agent swarm robots, among other systems whose optimal states can be derived from learning by interacting with their environment, are potential areas of application of this concept. This is especially useful in situations where the dynamics of the system are unknown, noisy or too complex to input by calculation. In such cases, receiving feedback from the environment in the form of rewards helps an intelligent agent estimate optimal actions and learn the goodness of a state over time.

Navigation tasks involve situations where an agent is put in an environment with or without obstacles and must successfully follow the optimum path to its goal state. Planning algorithms do not involve independent learning on the part of the agent to make decisions in unknown environments. Parameters, situations and tasks are pre-decided and directly input to the robot in the form of commands.

In this work, I apply reinforcement learning to solve real-world navigation problems in unknown environments using the Pioneer 3-DX robot. I aim to address the case where the robot is able, by exploration, to independently gather the knowledge necessary to make decisions in unknown situations. Human input is then unnecessary to make decisions. Once trained, based on this knowledge which it has independently gathered during exploration, it is able to make optimal decisions when put in any situation.

In a problem with discrete state- and action-spaces, by solving three independent tasks of mounting complexity, I successfully trained the robot to go from **any** initial position to **any**

goal position, in an unknown environment with **obstacles**, by engaging its SONAR range device. This algorithm may also be applied in dynamic obstacle navigation.

Each task was performed in three stages:

1. Training on a virtual robot using MATLAB.
2. Testing in simulation, using MobileSim, a simulator for MobileRobots. The arena is set up using ARIA, the higher-level software interface for MobileRobots platform.
3. Testing on the physical robot Pioneer 3-DX. [1]

[1] Video for testing can be found here - https://www.dropbox.com/sh/fd2b2p1xi6wsjnu/AACxvR9695aX-UGdan80Ta3ba?dl=0

TABLE OF CONTENTS

LIST OF FIGURES

NOTATION

$\mathcal{A}$	set of available actions
a	action
B	Bellman operator
e	voltage applied
F	forces applied
L	distance between robot wheels
(m_p, n_p)	previous robot position reference
(m_c, n_c)	current robot position reference
(m_g, n_g)	goal position reference
(m_{rel}, n_{rel})	relative position reference
O_i	Binary variable $\in \{0, 1\}$ indicating obstacle in direction i
$P(s'\|s, a)$	probability that action *a* taken in state *s* will lead to s' at the next time step
$Q(s, a)$	state-action value function at state s
R	reward function
$R_a(s, s')$	expected immediate reward on transition from s to s' on taking action a
R_∞	cumulative discounted sum of rewards
r	sampled reward
$\mathcal{S}$	set of possible states
s	state
T	torque applied
$V(s)$	state value function at state s
V_x	forward velocity
v	tangential velocity
γ	discount factor $\in\ (0, 1)$
$\pi(s)$	policy (action) taken in state s
θ	angular orientation of the robot
ω	rotational velocity

CHAPTER 1

INTRODUCTION

Reinforcement learning (RL) is a rapidly evolving sub-field of machine learning and statistics, whose basic concepts are inspired mainly from behaviorist psychology. Machine learning algorithms operate by building and using a model based on inputs to make predictions or decisions, rather than following only explicitly programmed instructions. When we think about the nature of learning, the foundational idea governing its progress is learning by interacting with the environment. Among several approaches to machine learning, reinforcement learning is much more focused on goal-directed learning from interaction.

Reinforcement learning is learning a policy - mapping situations to actions - so as to maximize a numerical reward signal. Unlike most forms of machine learning, the learner is not told which actions to take, and must instead discover which actions yield the most reward by trying them. Actions may affect not only the immediate reward but also the next situation and thus, all subsequent rewards. Two major characteristics distinguish reinforcement learning from other machine learning sub-fields: trial-and-error search, and delayed reward.

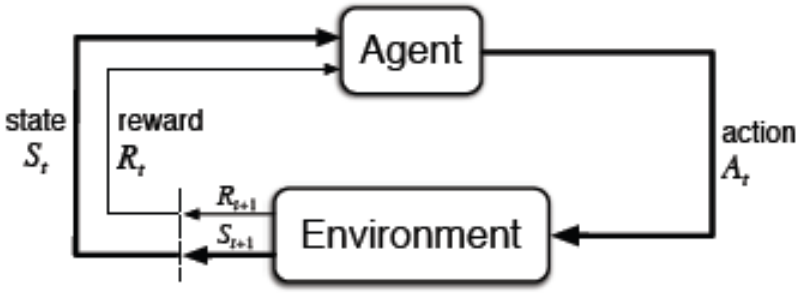


Figure 1.1: Agent-environment interaction in reinforcement learning

With its wide applications in control, optimization and robotics, reinforcement learning has mounting potential in mechanical engineering to design single and multi-agent systems with desired behaviours. Manufacturing processes, propulsion systems, cruise controllers in automobiles and multi-agent swarm robots, among other systems whose optimal states can be

derived from learning by interacting with their environment, are potential areas of application of this concept. We chose to apply reinforcement learning to solve real-world navigation problems in unknown environments using the P3-DX robot.

1.1 The Navigation Problem

Navigation tasks involve situations where an agent is put in an environment with or without obstacles and must successfully follow the optimum path to its goal state. Planning in a navigation task with robots generally involves complete knowledge of the robot environment, and human input of this knowledge while planning. Planning algorithms do not involve independent learning on the part of the agent to make decisions in unknown environments. Parameters, situations and tasks are pre-decided and directly input to the robot in the form of commands. Decision-making is based on human knowledge which is input during programming.

We aim to address the case where the robot is able, by exploration, to independently gather the knowledge necessary to make decisions in unknown situations. Human input is then unnecessary to make decisions and is taken out of the equation. During training, much like the human brain, the robot explores its environment and learns what is right or wrong, good or bad, by receiving rewards (praises or admonishments for its various actions) from the environment it interacts with. These rewards are a property of the environment, and not the agent. Then, based on this knowledge which it has independently gathered during exploration, it is able to make decisions when put in any situation.

This kind of learning-based approach, as opposed to one wherein the agent is directly commanded, makes the problem formulation much more general, as well as harder.

1. **Generality** stems from the fact that the robot may be trained for any task in any situation (environment), given the necessary training period for exploration and reward feedback. This is unlike custom-made commands for specific tasks.
2. **Added complexity** stems from the fact that the agent now assumes no knowledge of the environment.

1.2 Guide Through the Report

1. Chapter 1 deals with an intuitive introduction of the idea of reinforcement learning in navigation tasks. It also provides a guide through the project report.

2. Chapter 2 provides several concepts of reinforcement learning which were used and built on during the project.

3. Chapter 3 details the kinematics, dynamics and control of the Pioneer robot P3-DX.

4. Chapter 4 details various P3-DX navigation tasks, their formulation, the training phase, testing in simulation as well as physical testing.

5. Chapter 5 discusses possible future work and extensions of this project.

CHAPTER 2

BACKGROUND CONCEPTS

Chapter overview:

Before detailing reinforcement learning (RL), related concepts and algorithms, it is important to understand the problem formulation in the form of a Markov Decision Process (MDP). This is a special kind of control process wherein outcomes are uncertain and, thus, probabilistic. It assumes that an agent, at any time instant, finds itself in a particular state - which may be detailed by any number of state variables - and has a certain set of actions available to it. Reinforcement learning is then introduced as a special class of problems which may be modeled as this decision process. Lastly, in order to find optimal strategies in this problem setting, it is essential to keep track of the knowledge acquired by the agent. Most reinforcement learning algorithms do so by maintaining state and state-action value functions.

2.1 Markov Decision Process

Markov Decision Processes (MDPs) are discrete time stochastic control processes. These provide a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker (agent).

At any time-step, the agent observes its state s and takes an action a. The agent moves into a new state s', governed by a conditional probability distribution influenced by the chosen action and the current state - called $P(s'|s,a)$. Additionally, the agent receives a reward r from the environment that depends on the current state s and the decision maker's action a and the next state s'.

Given the current state, all future dynamics are independent of the history of the system. Thus, it satisfies the Markov property.

Definition 1. *A Markov Decision Process (MDP) is a 5-tuple:*

$$(\mathcal{S}, \mathcal{A}_s, P(s'|s,a), R_a(s,s'), \gamma), \text{ where:}$$

- *S: set of states*
- $\mathcal{A}_s$*: set of actions available at state S*
- $P(s'|s,a)$*: probability that an action a taken in state s will lead to state* s' *at the next time step*
- $R_a(s,s')$*: the expected immediate reward received after transition from state* s *to* s' *upon taking action* a
- γ*: discount factor between (0,1) representing the difference in importance between future and present rewards.*

Definition 2. *A policy* π *for an MDP is a mapping* $\pi : \mathcal{S} \mapsto \mathcal{A}$ *from states to actions;* $\pi(s)$ *denotes the action choice in state* s*.*

Upon observing its state, the agent acts in the environment according to a policy. The goal of the problem is to choose a policy π which maps each state to the corresponding best action. This policy must result in the maximization of the cumulative discounted sum of rewards.

Definition 3. *The cumulative discounted sum of rewards* R_∞ *is as follows:*

$$R_\infty \stackrel{def}{=} \sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1}),$$
$$\text{where } a_t = \pi(s_t)$$

Definition 4. *The optimal policy* π^* *is as follows:*

$$\pi^* \stackrel{def}{=} \arg\max_{\pi} \mathbf{E}\big[R_\infty\big]$$

2.1.1 The mouse in the maze problem

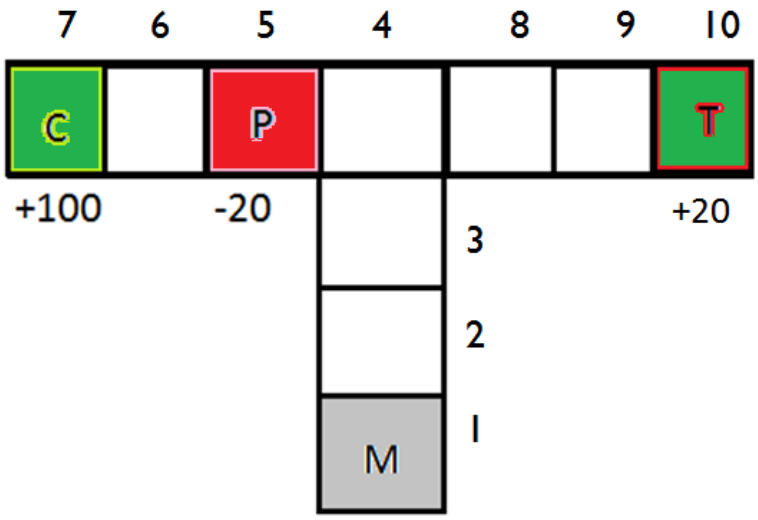


Figure 2.1: Mouse in the maze problem - the mouse starts in state 1.

Here is an example to show how decision-making is carried out in the MDP framework. The following are details of symbols in the above figure:

- Numbers 1 to 10 denote various states possible in the maze.
- Our agent is the mouse, which starts in state 1, denoted by the letter M.
- State 7 contains cheese (C). Here, the reward received from the environment is +100.
- State 10 contains toffee (T). Here, the reward received from the environment is +20.
- State 5 contains a puddle (P). Here, the reward received from the environment is -20.
- All other states receive reward = 0.

The discount factor γ can be seen to impact the far-sightedness of the agent. From Definition 3 it is easy to see that a small value of γ would result in lower emphasis placed on future rewards as compared to immediate rewards. A large value of γ, on the other hand, would result in a far-sighted agent, resulting in substantial weights being placed on subsequent rewards, and not just the immediate reward.

At state 4, the mouse has the option of turning right or left. During exploration, it will try both and, based on the cumulative reward it receives, choose one action to follow in that state.

- On turning left, the mouse receives $R_\infty = -20 + \gamma(0) + \gamma^2(100)$.
- On turning right, the mouse receives $R_\infty = 0 + \gamma(0) + \gamma^2(20)$.

Let us assume different cases of γ:

1. $\gamma = \mathbf{0.1}$

   - On turning left, $R_\infty = -20 + 0.1(0) + 0.1^2(100) = -19$.
   - On turning right, $R_\infty = 0.1^2(20) = 0.2$.

   Therefore, it will turn **right** to maximize its cumulative sum of rewards.

2. $\gamma = \mathbf{0.9}$

   - On turning left, $R_\infty = -20 + 0.9(0) + 0.9^2(100) = 61$.
   - On turning right, $R_\infty = 0.9^2(20) = 16.2$.

   Therefore, it will turn **left** to maximize its cumulative sum of rewards.

Thus we see that, by employing a short-sighted policy, the mouse avoids the puddle and opts for the path to the toffee. If it is far-sighted, however, it opts to brave the puddle to get to the cheese beyond. Thus, depending on the kind of learner, different outcomes may be expected. Thus we have seen a working example of a navigation task solved using an MDP.

2.2 Reinforcement Learning

Reinforcement learning (RL) problems are modeled as Markov Decision Processes (MDPs), as defined above. Consider the simple example of a child learning to sing a song. He has several options at his disposal, like changing pitch, volume, modulation and pronunciation. He receives feedback from his environment in the form of reprimands or praise from his teacher or listener. This constitutes his reward r. So, at any time step, he is in a state s with respect to his current pitch, volume, etc., and takes an action a to change these values and come to a state s', and receives reward r from his teacher. At any time step, it is unknown to him what reward he will receive, as well as what state he might end up in while trying to change his modulation,

pronunciation, etc. From this example, thus, we see that RL problems belong to the special case wherein transition probabilities $P(s'|s,a)$ and/or reward function $R_a(s,s')$ are unknown and must be estimated by exploration. This estimation is either done implicitly, as in the case of model-free RL, or explicitly, as in the case of model-based RL.

Because of this sort of 'live' learning from the environment, the following are a few challenges which come up in reinforcement learning, and must be dealt with carefully:

- **Sequential Decision Making**: The decisions of the agent at any time must be dependent not only on immediate reward, but also on the next state it transitions to using the action it performs. Therefore, since its action impacts the future rewards it receives, actions must thus be chosen keeping this in mind. For instance, an agent might choose to act so that it receives a low immediate reward but transitions into a good state after which it can obtain high rewards in future transitions. This makes decision making sequential, and not just immediate.

- **Exploration v/s Exploitation**: At any time step, a learner has the knowledge it has gathered until then at his disposal. It can therefore exploit this knowledge in making its next decision. However, much like a curious human learner, it also has the option of randomly taking an action to improve its knowledge of the environment. This trade-off - exploration vs. exploitation - must be appropriately chosen so neither one is in excess. If it always exploits, it may lose out on knowledge it may have gained by exploring further, whereas if it never exploits its knowledge, it may get stuck with very bad decisions.

2.3 Value Functions

After gathering knowledge through its explorations, a learner must be able to quantify the goodness of a state, using a 'value function'. The value of a state is derived from the discounted sum of rewards, with the amount of emphasis placed on immediate reward as compared to future reward determined by the discount factor - γ, as explained above. A large vale of γ indicates a learner which is far-sighted, whereas a small value indicates short-sightedness.

A state's value is the return the agent *expects* to receive, should it land in that state. The policy

the learner chooses to follow will then be based on the estimated values of the states around it. These values are updated as the learner receives new information.

Definition 5. *A state value function is a mapping $V : \mathcal{S} \mapsto \mathbb{R}$ from states to the set of real numbers. The value $V^\pi(s)$ of a state s under a policy π is defined as the expected, total, discounted reward when the process begins in state s and all decisions are made according to policy π.*

$$\begin{aligned} V^\pi(s) &\stackrel{def}{=} \mathbf{E}_\pi\big[R_\infty | s_1 = s\big] \\ &= \sum_{s'} P(s'|s, \pi(s))\Big(R_{\pi(s)}(s, s') + \gamma \mathbf{E}_\pi\big[R_\infty | s_1 = s'\big]\Big) \\ &= \sum_{s'} P(s'|s, \pi(s))\Big(R_{\pi(s)}(s, s') + \gamma V^\pi(s')\Big) \end{aligned}$$

The value function may be updated using the above definitions. The policy chosen here would thus be updated as follows:

$$\pi(s) \leftarrow \arg\max_a \sum_{s'} P_a(s, s')(R_a(s, s') + \gamma V(s'))$$

Assigning a value to a state directly masks the action which gives the state its value making it difficult to obtain the policy from the value function. Instead, each action available in each state may be assigned a value, so as to decide the best action which can be taken in any state. The following defines a state-action value function.

Definition 6. *A state-action value function is a mapping $Q : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ from state-action pairs to the set of real numbers. The value $Q^\pi(s, a)$ of a state-action pair (s, a) under a policy π is defined as the expected, total, discounted reward when the process begins in state s, the agent takes action a in the first time-step and all decisions (actions) are made according to*

policy π.

$$
\begin{aligned}
Q^{\pi}(s,a) &\stackrel{def}{=} \mathbf{E}_{\pi}\big[R_{\infty}|s_1 = s, a_1 = a\big] \\
&= \sum_{s'} P(s'|s,a)\Big(R_a(s,s') + \gamma \mathbf{E}_{\pi}\big[R_{\infty}|s_1 = s'\big]\Big) \\
&= \sum_{s'} P(s'|s,a)\Big(R_a(s,s') + \gamma V^{\pi}(s')\Big) \\
&\quad where\ V^{\pi}(s') = Q^{\pi}(s', \pi(s'))
\end{aligned}
$$

CHAPTER 3

PIONEER 3-DX ROBOT SPECIFICATIONS

Figure 3.1: The mobile robot Pioneer 3-DX robot

The efficacy of reinforcement learning for the robot navigation problem was tested on the standard Pioneer P3-DX robot. In this chapter, we introduce the kinematics and dynamics of this robot. Figure 3.2 shows the design of the robot.

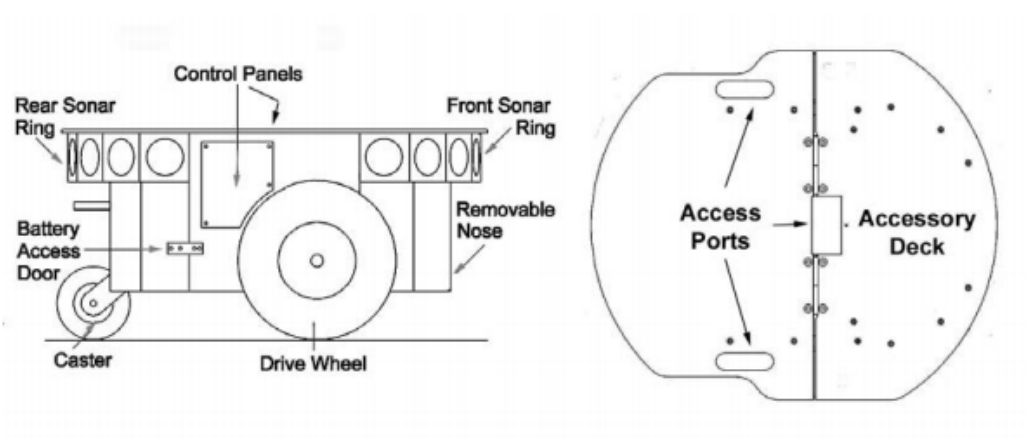


Figure 3.2: Views of mobile robot P3-DX and sensors

Pioneer robot P3-DX is a two wheel, two motor, differential drive robot that adheres to the unicycle robot model. The base platform can reach speeds of 1.6 meters per second and carry a payload of up to 23 kg. The robot is powered by three hot-swappable 9Ah sealed batteries. At

any instant, it has zero lateral motion, i.e., it cannot move perpendicular to the forward direction of motion (the robot's heading).

3.1 Kinematics

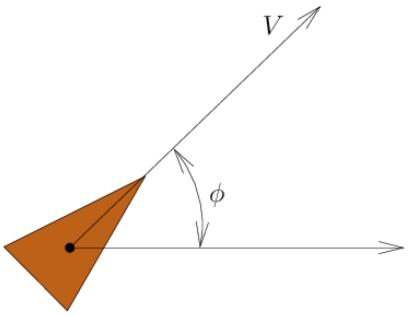


Figure 3.3: The simplified unicycle model

Figure 3.3 represents the unicycle model. As can be seen from this figure, the model assumes a set speed in its forward direction, as well as an angular (rotational) velocity. The following Jacobian encompasses the velocities in the robot's local frame of reference:

$$\mathcal{J} = \begin{bmatrix} V_x \\ V_p \\ \omega \end{bmatrix} = \begin{bmatrix} V_x \\ 0 \\ \omega \end{bmatrix} \tag{3.1}$$

V_x is the velocity in the direction of the robot heading. V_P is the velocity perpendicular to its heading, and it is $= 0$ in a unicycle type model. ω is the angular velocity of the robot. The differential drive robot can be modeled as a unicycle type machine, using conversions of the wheel velocities. Let ω_r be the angular velocity of the robot's right wheel. Let ω_l be the angular velocity of its left wheel. These velocities are angular, in rad/sec, and therefore, converting them to V_x involves multiplying with the wheel radius R, since $V_{tangential} = \omega R$.

Also, $V_{trans} = V_{tangential}$ in pure rotation. Therefore,

$$V_x = \omega_{avg} R \tag{3.2}$$

$$= \frac{(\omega_r + \omega_l)}{2} R \tag{3.3}$$

Let $v_r = \omega_r R$ and $v_l = \omega_l R$, representing wheel velocities of the robot. Figure 3.4 shows the top view of the robot. L is the distance between its wheels, and the arrow ω shows its direction of rotation about the instantaneous center of curvature (ICC).

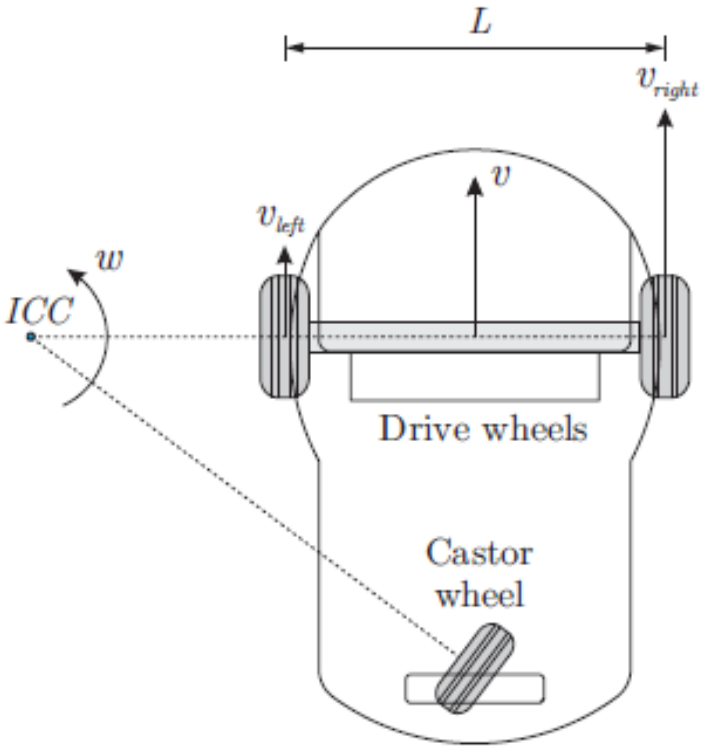


Figure 3.4: Top view of mobile robot P3-DX

From the figure, we see,

$$\omega = \frac{(v_r - v_l)}{L} \tag{3.4}$$

$$= \frac{(\omega_r - \omega_l)}{L} R \tag{3.5}$$

The angular velocity of the robot in the local frame (which is the same as in the global frame),

has thus been derived above. It is the resultant velocity of rotation ($v_r - v_l$) divided by the distance between the wheels (L = radius of rotation). Recall, since wheel velocities are controllable, they must be derived from the input V_x and ω.

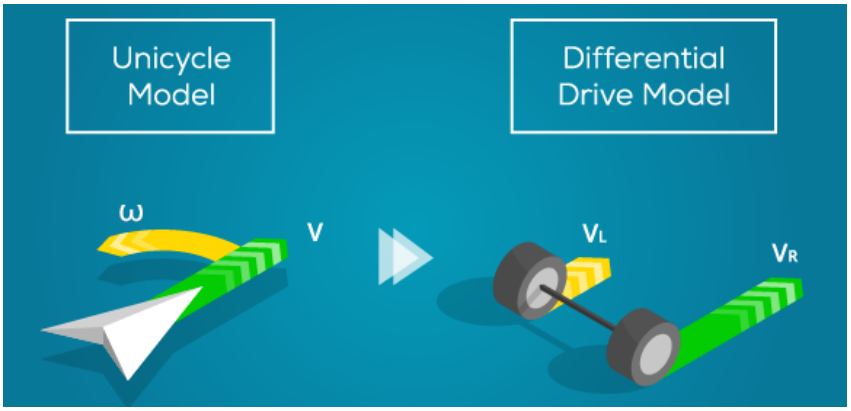


Figure 3.5: Modelling a differential drive robot as a unicycle model

From equations 3.3 and 3.5, the wheel angular velocities may be derived from a desired forward and robot angular velocity. Therefore, given a desired velocity and rotation angle, the kinematics of the robot may be defined by simultaneously solving the equations, resulting in the following:

$$v_r = \frac{(2V_x + \omega L)}{2R} \tag{3.6}$$

$$v_l = \frac{(2V_x - \omega L)}{2R} \tag{3.7}$$

In the global frame of reference, the kinematic equations would be as follows:

$$\dot{x} = V_x \, cos(\theta) \tag{3.8}$$

$$\dot{y} = V_x \, sin(\theta) \tag{3.9}$$

$$\dot{\theta} = \omega \tag{3.10}$$

3.2 Dynamics

The translational and rotational dynamics of the unicycle robot can be described, using Newton's 2^{nd} law of motion, by:

$$M\dot{V_x} = F - B_v V_x \tag{3.11}$$

$$J\dot{\omega} = T - B_\omega \omega \tag{3.12}$$

Here, M is the robot's mass, J is the Inertia Moment, F are the forces applied on the system, T is the wheel axis binary, B_v is the translational friction coefficient and B is rotational friction coefficient. We assume that these values are constant for the typical values of velocities of a moving robot.

Consider the forces acting on the left and right wheels, F_L and F_R. These can be replaced by an equivalent force and moment:

$$F = F_R + F_L \tag{3.13}$$

$$T = \frac{L(F_R - F_L)}{2} \tag{3.14}$$

Now, considering the voltages of the left and right wheel motors, if e_{am} is the mean voltage and e_{ad} is the differential voltage, the relationship between voltages and forces is as follows:

$$F = K_m e_{am} - K_v V_x \tag{3.15}$$

$$T = K_d e_{ad} - K_\omega \omega \tag{3.16}$$

Here K_m, K_v, K_d, K_ω are constants corresponding to relative coefficients of mean voltage, forward velocity, differential voltage and rotational velocity respectively. Therefore, we have

the following dynamic model for the unicycle:

$$M\frac{dV_x}{dt} = -K_v V_x + K_m e_{am} \tag{3.17}$$

$$J\frac{d\omega}{dt} = -K_\omega \omega + K_d e_{ad} \tag{3.18}$$

3.3 Control

Figure 3.6 shows the flow diagram for control [Carona *et al.*, 2008] in the Pioneer P3-DX robot. The four major blocks are shown in the figure.

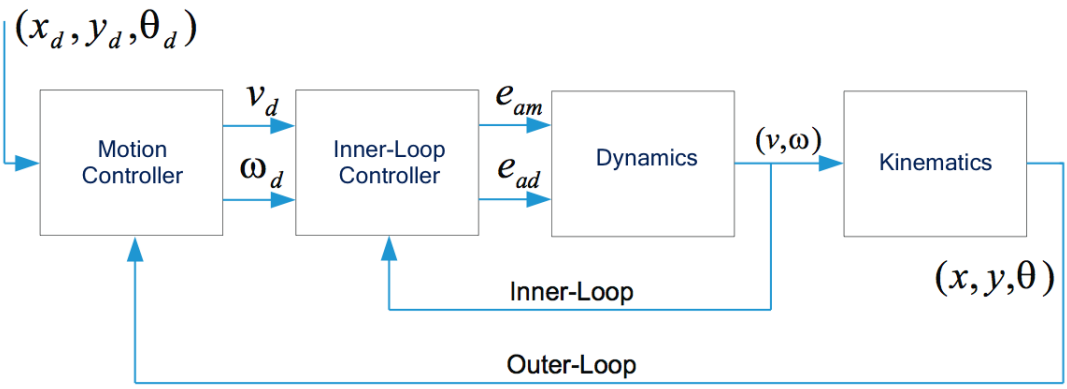


Figure 3.6: Flow diagram for mobile robot P3-DX control

3.3.1 Motion Controller

The motion controller describes the outer-loop control of the system.

1. **Inputs:**

   x_d, y_d, θ_d : the desired position and orientation of the robot.
   x, y, θ : the current position and orientation of the robot.

2. **Outputs:**

   v_d, ω_d: the desired forward and rotational velocity of the robot.

Here, the desired velocities are computed given the desired (next) and current positions. This may be done by forcing the robot to perform according to a pre-defined tracking law, with time-parametrized reference points along the path to its goal. Therefore, according to this inbuilt law, using the reference points and relative error between current and next states, the output desired forward and rotational velocities are computed, and input to the inner-loop controller.

3.3.2 Inner-Loop Controller

This describes inner-loop control of the system.

1. **Inputs:**

v_d, ω_d : the desired forward and rotational velocity of the robot.
v, ω : the observed forward and rotational velocity of the robot.

2. **Outputs:**

e_{am}, e_{ad} : mean and differential voltages applied to the robot wheels.

To accomplish the goal of driving the robot to a desired linear velocity v_d and angular velocity ω_d, the first step is to compute the error between the true and desired velocities. Let linear velocity error be e_v and angular velocity error be e_ω.

$$e_v = v - v_d \tag{3.19}$$

$$e_\omega = \omega - \omega_d \tag{3.20}$$

If the dynamic of the robot has small static gains then the error neighborhood may be significant. In that case one can enforce the steady state error convergence to zero (with a constant input) by using potential-integral control:

$$e_{am} = -K_{P1} e_v - K_{I1} \int_0^t e_v(\tau) d\tau \tag{3.21}$$

$$e_{ad} = -K_{P2} e_\omega - K_{I2} \int_0^t e_\omega(\tau) d\tau \tag{3.22}$$

Here, K_{P1} and K_{P2} are potential gains, while K_{I1} and K_{I2} are integral gains. Thus voltages are computed and output to the dynamics block.

3.3.3 Dynamics Block

Dynamics block applies forces required for changing velocities toward desired velocities.

1. **Inputs:**

   e_{am}, e_{ad} : mean and differential voltages applied to the robot wheels.

2. **Outputs:**

   v, ω : the observed forward and rotational velocity of the robot.

Forces needed to be applied on the wheels are thus calculated from the equations 3.17 and 3.18 in section 3.2. The resultant wheel linear and rotational velocities are then sensed and output to the inner-loop control block as well as the Kinematics block.

3.3.4 Kinematics Block

Kinematics block applies equations to calculate current position of the robot.

1. **Inputs:**

   v, ω : the observed forward and rotational velocity of the robot.

2. **Outputs:**

   x, y, θ : the current position and orientation of the robot.

Current position and orientation of the robot are calculated from equations 3.8, 3.9 and 3.10, given the forward and rotational velocities of the robot, and output to the motion controller block. This concludes the control section of this chapter.

CHAPTER 4

IMPLEMENTATION OF REINFORCEMENT LEARNING IN MOBILE ROBOT P3-DX NAVIGATION TASKS

As explained in the introduction, robot navigation tasks involve situations where the agent is put in an environment with or without obstacles and must successfully follow the optimum path to its goal state. However, the Markov Decision Process (Definition 1) needs to be formally defined for various tasks performed by the robot. In a nutshell, I have explored the following situations:

1. No-obstacle shortest path navigation
2. Fixed obstacle optimum path navigation, without engaging sensors
3. Dynamic obstacle optimum path sensor-engaged navigation

Before detailing the complexities involved in these experiments, I will first define how the learning environment, variables and parameters were set up.

4.1 Discrete State Space

From Definition 1, we have $\mathcal{S}$ as the set of all possible states in the problem.

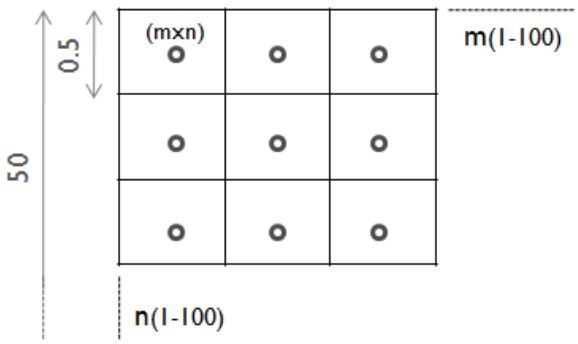


Figure 4.1: Robot exploration stage: Discrete grid for training

Each state is defined by a set of state variables. In our environment, a 2-D world is chosen. It is discretized into squares, and each square is referenced with a horizontal and vertical component. Dicretization means that, for defining the position of a robot, instead of actual x-y coordinates (which would be in a continuous space, with infinite possibilities), the coordinates of the center of the square which it is positioned in (a finite number of possibilities), are used. The reference horizontal or vertical component may also be used to indicate a particular square.

Consider Figure 4.1, which shows the robot exploration grid for training. m and n represent the vertical and horizontal components respectively. For example, the robot might start in $(m, n) = (1, 1)$, which according to the figure, would correspond to the upper-left corner of the grid. State variables may be derived from these definitions of components. In Figure 4.1, it is assumed that:

1. the size of each grid = $gSize$ = 500 mm = 0.5 m
2. the number of cells along each axis (horizontal and vertical) = $gNum$ = 100

The parameters $gSize$ and $gNum$ may be changed according to the desired exploration area and accuracy. The total area for exploration:

$$\text{width (m)} \times \text{length (m)} = \big(gSize \times gNum\big)_{horz} \times \big(gSize \times gNum\big)_{vert} \tag{4.1}$$

$$= (0.5 \times 100) \times (0.5 \times 100) \tag{4.2}$$

$$= 50 \text{ m} \times 50 \text{ m} \tag{4.3}$$

Lower limit on discretization

The grid size should not be arbitrarily lowered in a quest for accuracy. The robot's center of gravity is the point tracked during exploration. Since the robot's width is $\approx$ 300 mm, a grid size of around 500 mm would be a good choice. Inaccuracies in positioning also indicate this kind of sizing to be a better choice. Also, lowering the grid size would make the problem more complex, and the exploration needed would be far more intensive. Therefore, 500 mm is the ideal choice for grid size.

4.2 Discrete Action Space

From Definition 1, $\mathcal{A}_s$ is the set of actions available at state $s \in \mathcal{S}$. At every step, the robot can move into its 8 neighbouring squares as detailed in Figure 4.2. The coordinates in each neighbouring grid square represent the change in values of m and n, i.e., (m_{change}, n_{change}).

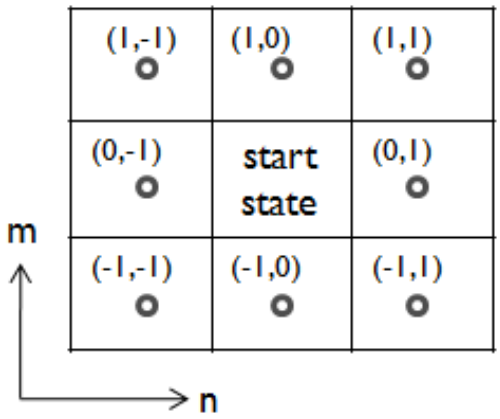


Figure 4.2: Actions possible according to start state

Therefore, there are 8 actions (directions) available to the robot, instead of a continuous case wherein it may move in one of an infinite number of directions away from its current state. This action space is thus discrete.

4.3 The Reward Function

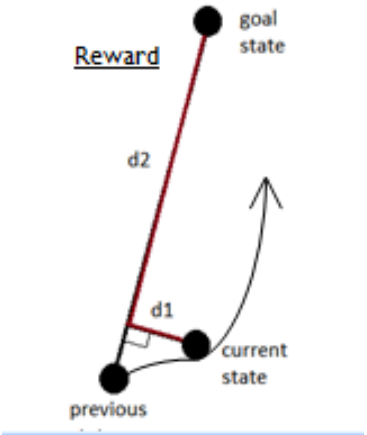


Figure 4.3: Reward function parameters

From Definition 1, R is the reward function fed back to the agent from its environment. The reward function is a **property of the environment**, and is unknown to the learner. In model-free reinforcement learning, it is estimated implicitly while exploring.

Consider Figure 4.3. The previous, current and goal points are shown. Let the squares be referenced as follows:

$$\begin{aligned} \text{Point 1. } (m_p, n_p) &= \text{Previous position reference} \\ \text{Point 2. } (m_c, n_c) &= \text{Current position reference} \\ \text{Point 3. } (m_g, n_g) &= \text{Goal position reference} \end{aligned}$$

The reward function is a function of $d1$ and $d2$:

$$\begin{aligned} d1 = &\text{ Perpendicular distance of Point 2 from the} \\ &\text{line joining Points 1 and 3} \\ d2 = &\text{ Distance from the foot of the perpendicular} \\ &\text{to Point 3, along the line joining Points 1 and 3} \end{aligned}$$

From straight-line equations and calculations, distances $d1$ and $d2$ are as follows:

$$d1 = \frac{|(m_g - m_p)(n_c - n_p) - (n_g - n_p)(m_c - m_p)|}{\sqrt{(m_g - m_p)^2 + (n_g - n_p)^2}} \tag{4.4}$$

$$d2 = 1 - \frac{|(m_g - m_p)(m_c - m_p) + (n_g - n_p)(n_c - n_p)|}{\sqrt{(m_g - m_p)^2 + (n_g - n_p)^2}} \tag{4.5}$$

For straight-line shortest-path navigation, both deviation from straight-line path as well as distance from the goal must be penalized, through the reward function. Since $d1$ is generally smaller than $d2$, a simple reward function like $-(d1 + d2)$ would lead to masking of the effect of $d1$ on the function. Since both must have nearly equal impact, the reward function chosen is:

$$R = -(1 + d1)(1 + d2) \tag{4.6}$$

Algorithm 1: Q Learning

Input: Bellman operator B for the MDP, and allowable error ϵ
Initialize $Q(s, a) \leftarrow 0$
Initialize $converged = false$
repeat
  $Q_{new} = BQ_{old}$
  if $\max(|Q_{new} - Q_{old}|) < \epsilon$ **then**
    $converged = true$
  end if
  $Q_{new} = Q_{old}$
until $converged = true$
return Q_{new}

4.4 Q-learning

For the purpose of training, the Q-learning algorithm is employed, where the $Q^{\pi}(s, a)$ state-action value function is updated during exploration. Recall, from Definition 6,

$$Q^{\pi}(s, a) = \sum_{s'} P(s'|s, a)\Big(R_a(s, s') + \gamma Q^{\pi}(s', \pi(s'))\Big) \tag{4.7}$$

Let us now introduce the Bellman operator B. This operates on the state-action value function and is defined as follows:

$$BQ(s, a) \stackrel{\text{def}}{=} \sum_{s'} P(s'|s, a)\Big(R_a(s, s') + \gamma \max_a Q(s', a)\Big) \tag{4.8}$$

These definitions utilizes the Markovian property of the system. The value of a state-action can be completely defined using properties of the next state. The goodness of a state is thus a weighted sum of the reward it may receive by transitioning to the next state and the maximum value it may receive in the next state. Therefore, from equations 4.7 and 4.8, we see that, for the optimal state-action value function Q^*, the policy will **not** change on update, and will be called the optimal policy π^*. Therefore:

$$Q^{\pi^*}(s', \pi^*(s')) = \max_a Q(s', a) \tag{4.9}$$

$$\Leftrightarrow BQ^* = Q^* \tag{4.10}$$

The algorithm pseudo-code is written as shown in Algorithm 1.

4.5 The software-hardware interface

The P3-DX is a robot platform by MobileRobots Inc.(ActivMedia Robotics) and comes equipped with a 32-bit Renesas SH2-7144 RISC microprocessor, including the P3-SH microcontroller with the ARCOS (Advanced Robot Control and Operations Software), an embedded software (firmware) that provides a client-server interface.

The Advanced Robotics Interface for Applications (ARIA) software is a higher level software interface for MobileRobots platforms. ARIA is a programming library (SDK) for the C++ language with wrappers in Python and Java providing both high and low level access to MobileRobots/ActivMedia platforms and accessories. In addition to providing a robot and accessory API to developers, ARIA also serves as a foundation for other libraries providing additional capabilities: For general communication over the network, ArNetworking is used.

4.6 P3-DX Robot Task 1

The first task the robot must learn to perform is to navigate the shortest straight-line path to its goal position, in an environment with no obstacles. It must learn to do this given any start and end point in a pre-defined exploration area. For this task, each time the robot is trained, the following hold true:

1. The robot is trained to go from **any** initial position to a **any** goal position, and
2. It learns to do this in an environment with **no obstacles**.

For example, Figure 4.4 shows navigation results of an experiment on the Pioneer 3-DX robot at the PEIL Lab. The discretized grid as detailed in 4.1 is also shown.

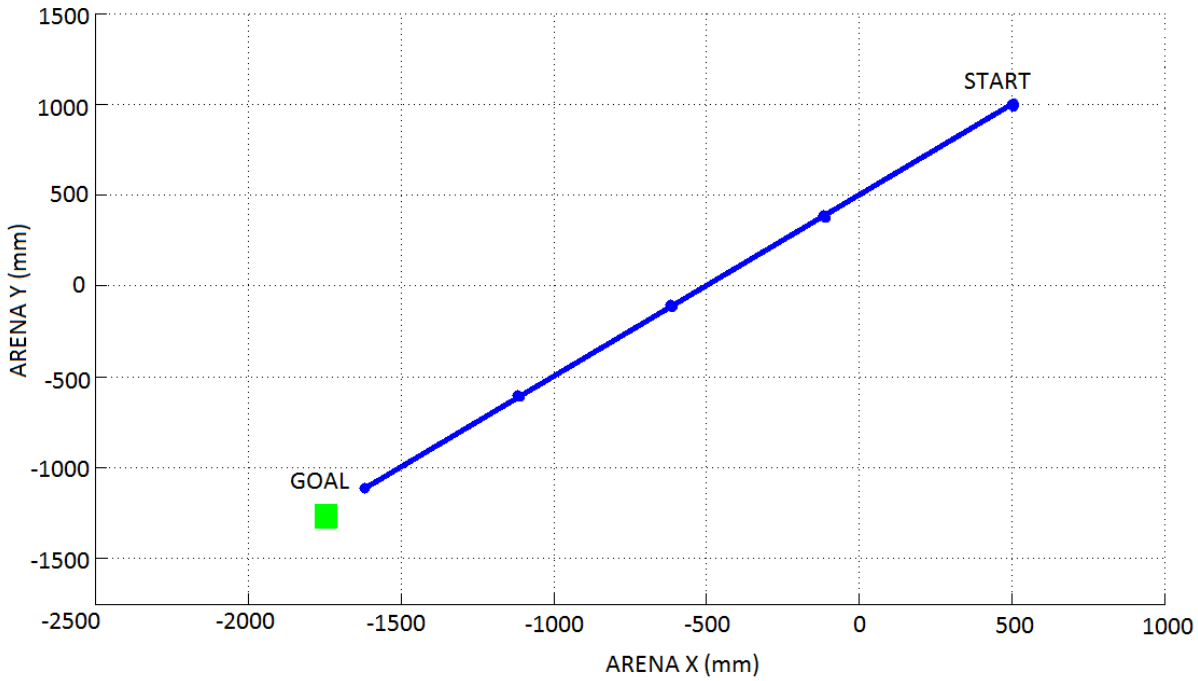


Figure 4.4: Experimental results - testing of mobile robot P3-DX

4.6.1 Training

The problem is posed as an MDP ($\mathcal{S}, \mathcal{A}_s, P(s'|s,a), R_a(s,s'), \gamma$), and the setting up of all parts of the 5-tuple is explained below.

State space $\mathcal{S}$

The state variables define the relative position of the robot from the goal position as follows:

$$m_{rel} = m_g - m_c$$

$$n_{rel} = n_g - n_c$$

These are the difference between goal and current references of the robot. Therefore, state s is defined as:

$$s = (m_{rel}, n_{rel})$$

The Q (state-action) value function updates are carried out for various start and goal references so as to completely train the robot for decision-making given *any* start and goal point. As the maximum values of m and n (m_{max} and n_{max}) increase, the time taken for complete training increases, due to Q being a state-action value matrix of size:

$$\begin{aligned}
&\Big((m^{rel}_{max} - m^{rel}_{min} + 1) \times (n^{rel}_{max} - n^{rel}_{min} + 1) \times \text{size}(\mathcal{A})\Big) \\
=&\Big((2(m_{max} - 1) + 1) \times (2(n_{max} - 1) + 1) \times \text{size}(\mathcal{A})\Big) \\
=&\Big((2m_{max} - 1) \times (2n_{max} - 1) \times \text{size}(\mathcal{A})\Big)
\end{aligned}$$

These hold true because:

1. m_{rel} are integers $\in \big[(-m_{max} + 1), (m_{max} - 1)\big]$,
2. n_{rel} are integers $\in \big[(-n_{max} + 1), (n_{max} - 1)\big]$.

Action space $\mathcal{A}$

As seen in Figure 4.2, eight actions are possible at each state. Quantitatively, these are represented by changes in the m and n reference of the robot, which is the same as changes in the state variables m_{rel} and n_{rel}. Therefore the action space is:

$$\begin{aligned}
a &= (m_{change}, n_{change}) \\
\mathcal{A}_s &= \big\{(1,1)\ \ (1,0)\ \ (1,-1)\ \ (0,1)\ \ (0,-1)\ \ (-1,1)\ \ (-1,0)\ \ (-1,-1)\big\}
\end{aligned}$$

As the size of the action space increases, the time taken for complete training increases, due to Q being a state-action value matrix of size $\Big((2m_{max} - 1) \times (2n_{max} - 1) \times size(\mathcal{A})\Big)$

Transition probabilities $P(s'|s, a)$

States s' and s and action a are represented as:

$$\begin{aligned} s' &= (m'_{rel}, n'_{rel}) \\ s &= (m_{rel}, n_{rel}) \\ a &= (m_{change}, n_{change}) \end{aligned}$$

The transition probabilities are defined as follows:

1. $P(s'|s,a) = 1 \quad \text{for} \quad s' = (m_{rel} - m_{change}, n_{rel} - n_{change})$
2. $P(s'|s,a) = 0 \quad \text{for all other } s'.$

Reward function $R_a(s,s')$

The reward function chosen is as explained in section 4.3:

$$R = \ -(1 \ + \ d1)(1 \ + \ d2)$$

Discount factor γ

γ was set to a high value of 0.95 to ensure far-sightedness of the learnt policy.

Training the virtual robot

```
while( (m_rel ~= 0 || n_rel ~= 0) && (step <= step_max) )

    [max_cur_act, max_cur_index] = max(state_q_values(:,(m_rel + m_max), (n_rel + n_max)));
    m_rel_next = m_rel + next_rel(max_cur_index, 1);
    n_rel_next = n_rel + next_rel(max_cur_index, 2);

    if((abs(m_rel_next) > (m_max-1)) || (abs(n_rel_next) > (n_max-1)))
        state_q_values(max_cur_index,(m_rel + m_max), (n_rel + n_max)) = -Inf;
    else
        max_next_act = max(state_q_values(:,(m_rel_next + m_max), (n_rel_next + n_max)));
        reward = -((m_rel_next)^2 + (n_rel_next)^2)^0.5;
        state_q_values( max_cur_index,(m_rel + m_max), (n_rel + n_max)) = reward + gamma * max_next_act;
        m_rel = m_rel_next;
        n_rel = n_rel_next;
        step = step + 1;
    end
end
```

Figure 4.5: Training code - no obstacles task

To train the robot, a virtual environment was setup in MATLAB. The virtual robot was made to explore its environment, gather knowledge and make estimates for the goodness of actions in

various states. As can be seen in Figure 4.5, the code runs iteratively, updating 'state_q_values' (the Q state-action value function) at each step.

New random **start and end** points are chosen when any of the **termination conditions** is satisfied:

1. If the robot reaches the goal point.
2. If the maximum number of steps in each trajectory is exceeded.

The number of steps in each trajectory is limited, as is number of trajectories for training. If the robot tries to explore outside of the arena, it is immediately penalized with a reward of $-\infty$, ensuring it remains within the designated exploration area. When the training is complete, the policy is chosen as follows (refer Algorithm 1): $\pi(s) = \arg\max_a Q(s, a)$

4.6.2 Testing

Testing in simulation

Once training is complete and a policy matrix learnt, the robot is tested in a simulated environment using MobileSim, a simulator for MobileRobots. The arena is set up using ARIA, the higher-level software interface for MobileRobots platforms. Figure 4.6 shows various functions defined to accomplish this. The policy the robot learnt during training is accessed and used to make decisions based on the position it senses in its environment.

```
int get_policy_matrix(int policy_mat[][2*(n_max-1)+1]);
void get_position(ArRobot *robot, double *X_current, double *Y_current);
void get_n_m (double X, double Y, int n_m[]);
void get_n_m_rel(int n_m_current[], int n_m_goal[] , int n_m_rel[]);
void get_row_col_pol(int n_m_rel[], int row_col_pol[], int n_range[], int m_range[]);
void get_pol(int policy_mat[][2*(n_max-1)+1], int row_col_pol[], int *pol);
void get_n_m_next_policy(int pol, int next_rel[][2], int n_m_current[], int n_m_next[]);
void get_X_Y_next(int n_m_next[], double *X_next, double *Y_next);
void move_to_next(ArRobot *robot, double *X_current, double *Y_current, double X_next, double Y_next);
void turn_to(ArRobot *robot, double AngleToEnd);
```

Figure 4.6: Functions for testing in simulation and on the physical robot

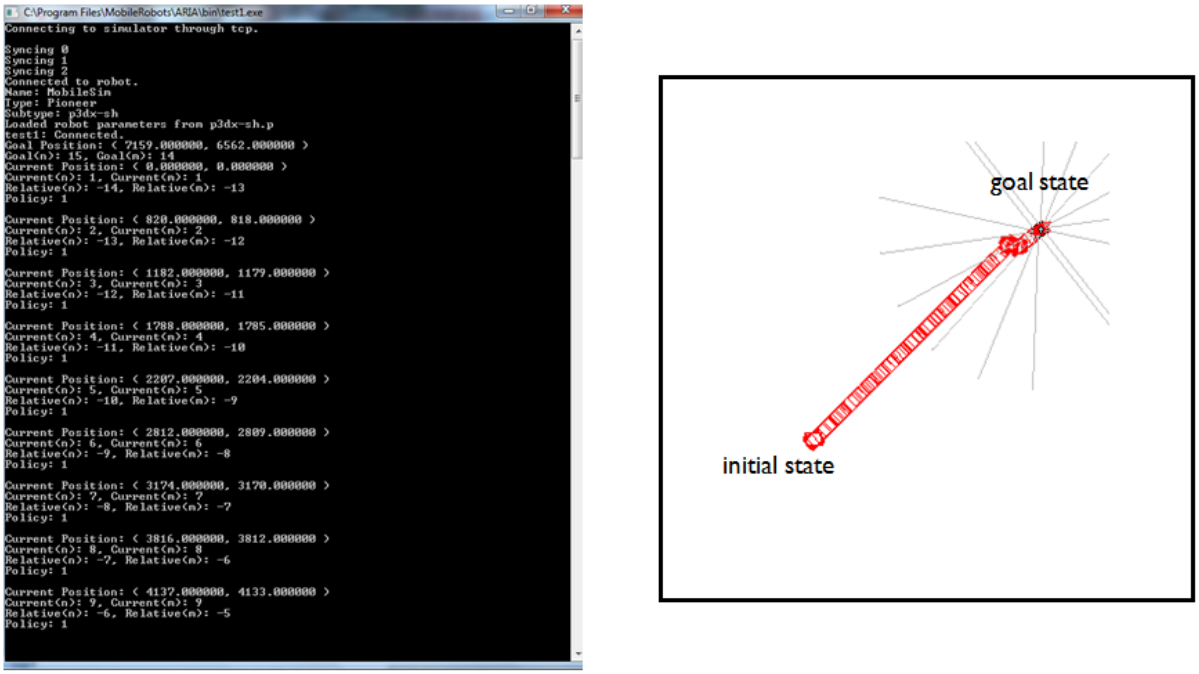


Figure 4.7: Output window and MobileSim simulation

On running code, a goal state is chosen and the robot traverses a path to it according to decisions it makes based on its training (i.e., its optimal policy π^*). The output window shown in Figure 4.7 prints the current position, the relative position and the policy at each step. The robot trail is shown at various intermediate steps along with goal information.

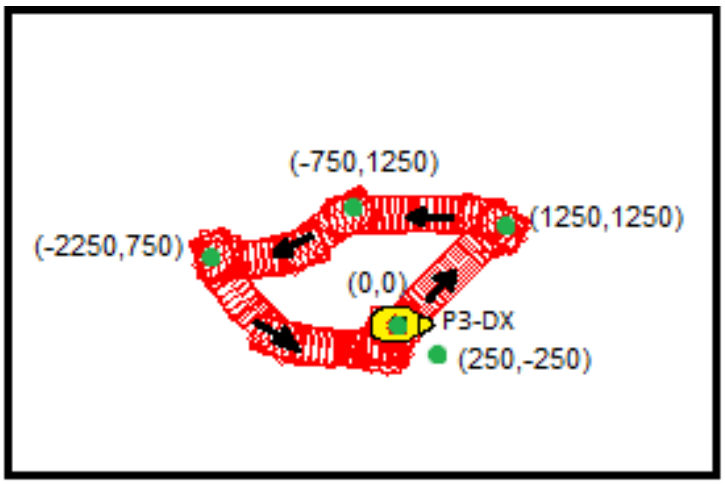


Figure 4.8: MobileSim simulation for multiple-goals experiment

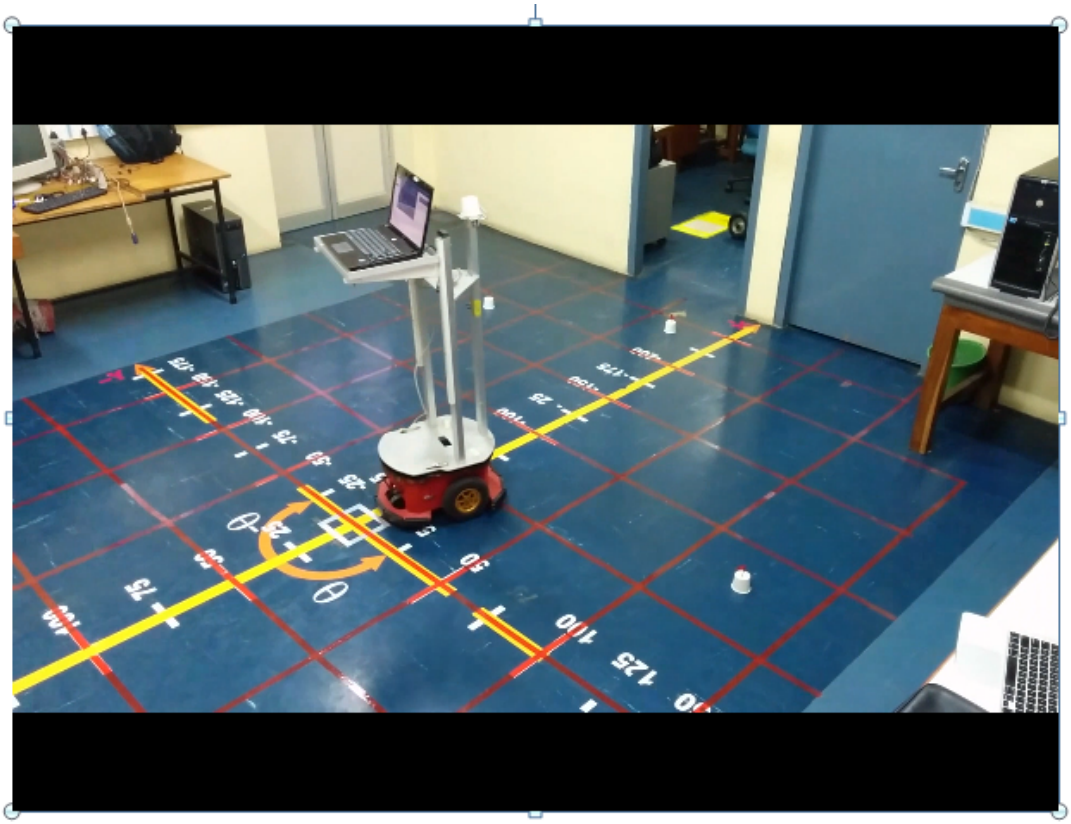


Figure 4.9: Mobile robot P3-DX navigating an environment with no obstacles

The algorithm was also run for multiple end goals across the arena. Figure 4.8 shows the simulation results for this experiment. The paths the robot follows are (in x-y coordinates (mm)):

$$(0, 0) \rightarrow (1250, 1250)$$
$$(1250, 1250) \rightarrow (-750, 1250)$$
$$(-750, 1250) \rightarrow (-2250, 750)$$
$$(-2250, 750) \rightarrow (250, -250)$$

Testing on the robot[1]

[1]Video for testing can be found here - https://www.dropbox.com/sh/fd2b2p1xi6wsjnu/AACxvR9695aX-UGdan80Ta3ba?dl=0

The algorithm was also tested on the physical robot as shown in figure 4.9. The arena was set up using grid squares of $0.5m \times 0.5m$. The grid can be seen in red in the figure. The white markers mark goal points for testing the robot.

Simulation vs. physical P3-DX experimental results

Figures 4.10 and 4.11 compare results of various experiments in simulation vs. on the physical robot.

1. Figure 4.10 shows results of the experiment where the start point of the robot is fixed and goal points are varied. The blue line shows
   simulation results, while the red line shows physical testing results.
2. Figure 4.11 shows results of the experiment where the goal point of the robot is fixed and start points are varied. The blue line shows
   simulation results, while the red line shows physical testing results.

From both these graphs, it is apparent that the robot is still able to learn to follow a nearly-straight path to its goal position despite restricting its movements, i.e., even with reduced dynamics, the agent is able to learn an optimal path.

In general, significant differences between the simulated trajectory and the actual trajectory may occur. The main reason for this deviation is *wheel slippage*. Since the P3-DX maintains its position estimate based on wheel encoders, wheel slippage would cause the robot to estimate a displacement further than it has actually gone. This is a common problem with wheel encoders and is overcome by using additional sensors for localization.

Since our test environment was controlled (at the PEIL laboratory), and the robot was in fairly good condition, the slippage was negligible and the state discretization was sufficient to overcome small errors.

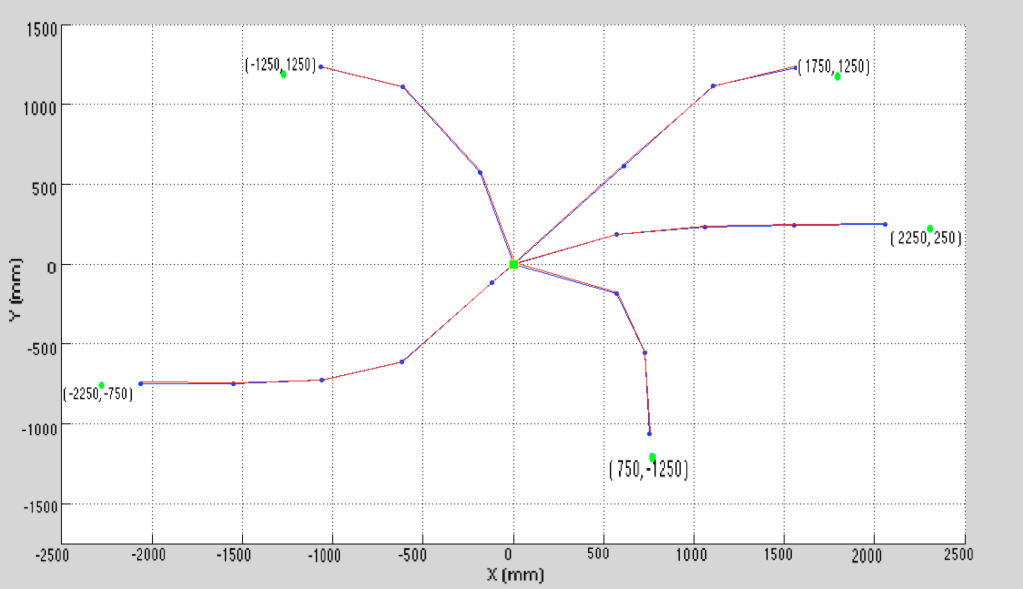


Figure 4.10: Performance in cases with fixed start point, different goal points

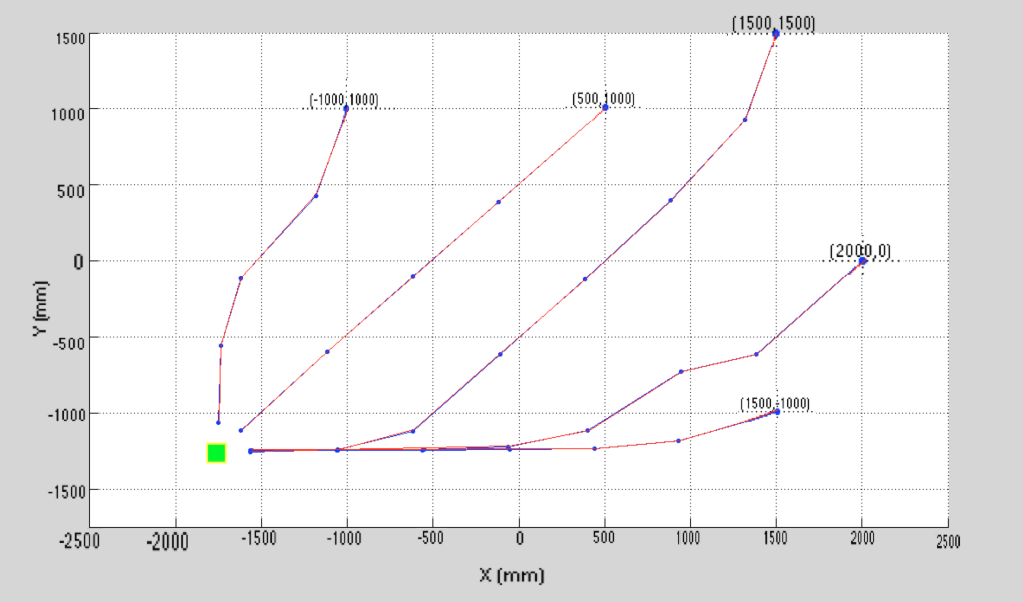


Figure 4.11: Performance in cases with different start points, fixed goal point

4.6.3 Challenges and Road Ahead

Challenges faced during the task formulation, training and testing and subsequent goals:

1. Delving into reinforcement learning (RL) studies from scratch to apply as an alternative approach to decision-making in this robot navigation task was an extensive process,

involving in-depth exploration of various concepts of computer science, an entirely new field.

2. Formulating the appropriate reward function for this task was a challenge. Refer to section 4.3 Initially, the reward function $R = -\sqrt{(m_g - m_c)^2 + (n_g - n_c)^2}$ was considered, to quantify absolute distance from the current point to the goal point. The results were as shown in Figure 4.12. This choice was inappropriate due to the chosen action space $\mathcal{A}_s$. The diagonal movements lead to a farther distance traversed than vertical and horizontal movements. Therefore, a better reward function was necessary to not only penalize distance from the goal, but also straight-line path deviation.
3. Real environments are cluttered with obstacles. For optimum path navigation, obstacle-avoidance must be incorporated in this framework. The following tasks deal with environments with obstacles.

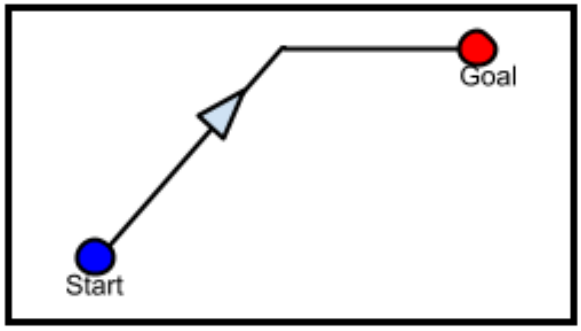


Figure 4.12: Performance for a particular reward function

4.7 P3-DX Robot Task 2

The second task the robot must learn to perform is to navigate the shortest path to its goal position, in an environment with **fixed obstacles**. For this task, each time the robot is trained, the following hold true:

1. The robot is trained to go from **any** initial position to a **specific** goal position, and
2. It learns to do this in an environment with a **specific pre-defined obstacle configuration**.

An example obstacle map is as shown in Figure 4.13. Here, the agent starts in the bottom left

corner of the grid (shown here in green) and must learn an optimal policy to get to any location in the map in the 'best' possible manner, while avoiding the obstacles (shown here in orange).

Added complexity

The task becomes more complex because the robot must now learn to safely navigate an environment with obstacles. Note that a lot of symmetry is lost in the presence of obstacles. In Task 1 (section 4.6), every combination of start and end states could be used for training and the policy could be defined for the goal state relative to the current position. Thus, learning $\Big((2m_{max}-1)\times(2n_{max}-1)\times size(\mathcal{A})\Big)$ values was all that was required to navigate optimally from any start state to any end state. In the presence of obstacles, the state of the robot is now defined relative to the obstacles i.e. in the global reference frame. In this setup, thus, we cannot generalized across different obstacle configurations.

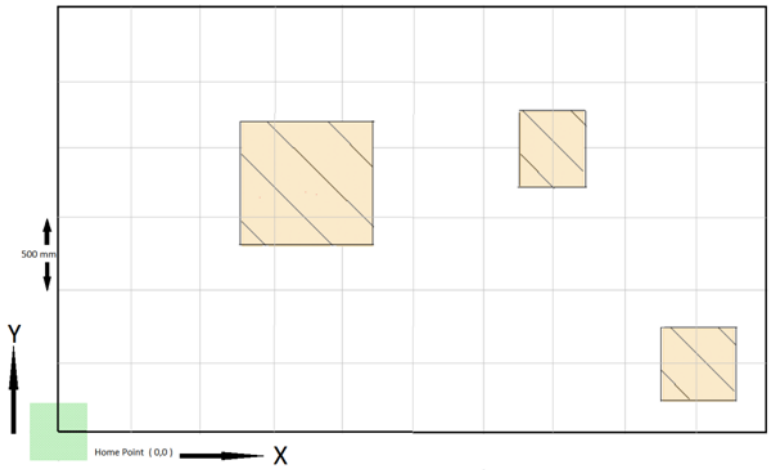


Figure 4.13: Example obstacle map

4.7.1 Training

The MDP for this task is similar to that in section 4.6.1 with an alteration made in the state specification to accommodate the lost symmetry.

State space $\mathcal{S}$

The state of the robot is now defined in absolute (global) coordinates due to the presence of fixed obstacles, and hence, the lost symmetry. The state variables are now the reference of the

square the robot is in:

$$s \;=\; (\; m_c,\; n_c\;)$$

The Q (state-action) value function updates are carried out for various start and goal references so as to completely train the robot for decision-making given *any* start point, given a *specific* goal point. As the maximum values of m_c and n_c (m_{max} and n_{max}) increase, the time taken for complete training increases, due to Q being a state-action value matrix of size:

$$\begin{aligned}&\Big((m_{max} - m_{min}) \times (n_{max} - n_{min}) \times \text{size}(\mathcal{A})\Big)\\ =&\Big(m_{max} \times n_{max} \times \text{size}(\mathcal{A})\Big)\end{aligned}$$

These hold true because:

1. m_c are integers $\in \big[0, m_{max}\big]$,
2. n_c are integers $\in \big[0, n_{max}\big]$.

Action space $\mathcal{A}$

The action space remains the same (Refer 4.2):

$$\begin{aligned}a &= (m_{change}, n_{change})\\ \mathcal{A}_s &= \big\{(1,1)\;\;(1,0)\;\;(1,-1)\;\;(0,1)\;\;(0,-1)\;\;(-1,1)\;\;(-1,0)\;\;(-1,-1)\big\}\end{aligned}$$

Probabilities, Rewards and Discount: $P(s'|s,a)$, $R_a(s,s')$, γ

These are the same as shown in section 4.6.1.

Training the virtual robot

Using the discrete state space and action framework, employing Q-learning, the virtual robot was trained on MATLAB for various end points using the obstacle map in Figure 4.13. The algorithm can be used to train the bot for a new given map or a new end point in about **10**

seconds for an arena of reference size 10×10. Figure 4.14 is a snapshot of the algorithm.

```
while( (m ~= m_goal || n ~= n_goal) && (step <= step_max) )

    [max_cur_act, max_cur_index] = max(state_q_values(:,m, n));
    m_change = next_rel(max_cur_index, 1);
    n_change = next_rel(max_cur_index, 2);
    m_next = m + m_change;
    n_next = n + n_change;
    m_rel = m_goal - m;
    n_rel = n_goal - n;

    if((m_next > m_max) || (n_next > n_max) || (m_next <= 0) || (n_next <= 0) || map(m_next,n_next) == 1)
        state_q_values(max_cur_index, m, n) = -Inf;
    else
        max_next_act = max(state_q_values(:, m_next, n_next));
        dist_line = (abs(m_rel*n_change - n_rel*m_change)/(m_rel^2 + n_rel^2)^0.5);
        dist_goal = ((m_rel^2 + n_rel^2)^0.5) - abs(m_rel*m_change + n_rel*n_change)/((m_rel^2 + n_rel^2)^0.5);
        reward = -( ( 1 + dist_line)*( 1 + dist_goal) );
        state_q_values( max_cur_index, m, n) = reward + gamma * max_next_act;

        m = m_next;
        n = n_next;
        step = step + 1;
    end
end
```

Figure 4.14: Training code - fixed obstacle task

A new random **start** point is chosen when any of the **termination conditions** is satisfied:

1. If the robot reaches the goal point.
2. If the maximum number of steps in each trajectory is exceeded.

The number of steps in each trajectory is limited, as is number of trajectories for training. The robot immediately penalized with a **reward of** $-\infty$:

1. If it tries to explore outside the arena.
2. If it tries to move into a state where an obstacle is present.

These ensure it learns to remain within the designated exploration area, and does not hit any obstacle. Once training is complete, the policy chosen is as follows (refer Algorithm 1): $\pi(s) = \arg\max_a Q(s, a)$

4.7.2 Testing

Testing in simulation

Once training is complete and a policy matrix learnt, the robot is tested in a simulated environment using MobileSim, the MobileRobots simulator. As before, the arena is set up using ARIA. Figure 4.6 shows various functions defined to accomplish this. The policy the robot learnt during training is accessed and used to make decisions based on the position it senses in its environment.

Figure 4.15 shows simulation performance for two example end points: $(2750, 2250)$ and $(4750, 2750)$. The obstacle map is as shown in Figure 4.13. The obstacles can be seen in orange in both figures.

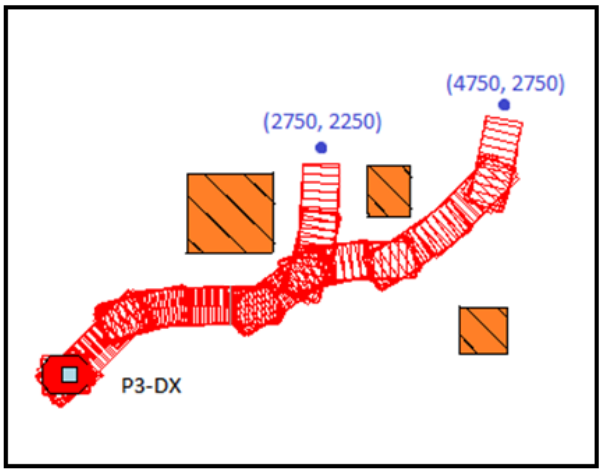


Figure 4.15: Simulation results - fixed obstacle navigation task

Testing on the robot[2]

The algorithm was also tested on the physical robot as shown in figure 4.16. The arena was set up using grid squares of $0.5m \times 0.5m$. The grid can be seen in red in the figure. The white areas represent environmental obstacles. The white markers mark goal points for testing the robot. The figure shows the robot in action, learning to avoid obstacles while making decisions to take the optimum path to $(2750, 2250)$.

[2]Video for testing can be found here - https://www.dropbox.com/sh/fd2b2p1xi6wsjnu/AACxvR9695aX-UGdan80Ta3ba?dl=0

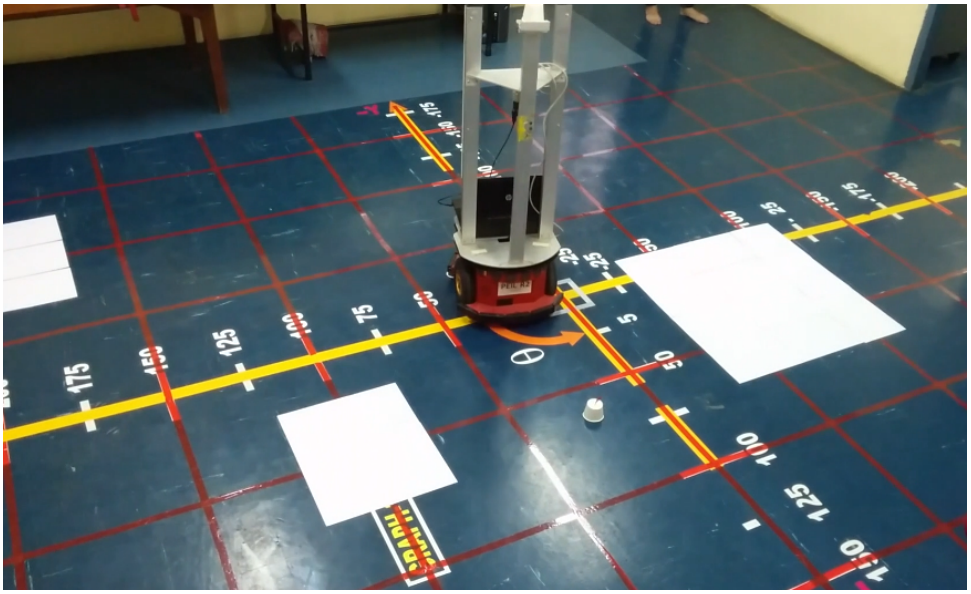


Figure 4.16: Mobile robot P3-DX navigating an environment with fixed obstacles

4.7.3 Challenges and Road Ahead

Challenges faced during the task formulation, training and testing and subsequent goals:

1. The major challenge here was re-formulating the MDP to fit in the global frame of reference. The lack of symmetry makes training more complex.
2. Per training phase, the robot learns to navigate only a specific obstacle configuration, and only to a specific goal point. It must be retrained for every new goal, or every new configuration. Making navigation more general by engaging the robot sensors during navigation is the next goal, and addressed in the next robot task.

4.8 P3-DX Robot Task 3

The third task the robot must learn to perform is to navigate the shortest path to its goal position, in an environment with **fixed or dynamic obstacles**. For this task, each time the robot is trained, the following hold true:

1. The robot is trained to go from **any** initial position to **any** goal position, and
2. It learns to do this in an environment with **any obstacle configuration**.

Added complexity

The task is made more general and adaptable to any obstacle configuration and any goal point. This means that training is only necessary once, and the robot **does not need to be retrained** for every new configuration or goal. This also means that the same algorithm can be used for **dynamic** obstacle avoidance. However, the Q value function now becomes of size:

$$\Big((2m_{max}-1)\times(2n_{max}-1)\times 2\times 2\times 2\times 2\times size(\mathcal{A})\Big),$$

as will be explained in a later section. Therefore, it is considerable more complex to train, which is the price we pay for generality.

4.8.1 Training

The MDP (refer Definition 1) is redefined for this task as follows.

State space $\mathcal{S}$

The state variables define the relative position of the robot from the goal position, as well as whether there are obstacles in its environment, as follows:

$$m_{rel} = m_g - m_c$$
$$n_{rel} = n_g - n_c$$
$$O_N,\ O_W,\ O_E,\ O_S \in \{0,1\}$$

These are the differences between goal and current references of the robot. The other four variables are binary, indicating whether or not an obstacle is present to the robot's north, west, east or south respectively. 0 indicates lack of obstacles, 1 indicates the presence of one. Therefore, state s is defined as:

$$s\ =\ (\ m_{rel},\ n_{rel},\ O_N,\ O_W, O_E,\ O_S\ )$$

The Q (state-action) value function updates are carried out for various random start and goal

references, as well as various random obstacle configurations so as to completely train the robot for decision-making given **any** start and goal point and faced with **any** obstacles. As the maximum values of m and n (m_{max} and n_{max}) increase, the time taken for complete training increases, due to Q being a state-action value matrix of size:

$$\begin{aligned}
&\Big((m^{rel}_{max} - m^{rel}_{min} + 1) \times (n^{rel}_{max} - n^{rel}_{min} + 1) \times 2 \times 2 \times 2 \times 2 \times \text{size}(\mathcal{A})\Big) \\
=&\Big((2(m_{max} - 1) + 1) \times (2(n_{max} - 1) + 1) \times 2 \times 2 \times 2 \times 2 \times \text{size}(\mathcal{A})\Big) \\
=&\Big((2m_{max} - 1) \times (2n_{max} - 1) \times 2 \times 2 \times 2 \times 2 \times \text{size}(\mathcal{A})\Big)
\end{aligned}$$

These hold true because:

1. m_{rel} are integers $\in \big[(-m_{max} + 1), (m_{max} - 1)\big]$,
2. n_{rel} are integers $\in \big[(-n_{max} + 1), (n_{max} - 1)\big]$,
3. $O_N,\ O_W,\ O_E,\ O_S \in \{0, 1\}$.

Action space $\mathcal{A}$

The action space (Refer 4.2) is now reduced, excluding diagonal actions to facilitate rapid training:

$$\begin{aligned}
a &= (m_{change}, n_{change}) \\
\mathcal{A}_s &= \big\{(1,0)\ \ (0,1)\ \ (0,-1)\ \ (-1,0)\big\}
\end{aligned}$$

Therefore, $\forall s \in \mathcal{S}, \quad |\mathcal{A}_s| \ = \ 4$

Probabilities, Rewards and Discount: $P(s'|s,a)$, $R_a(s,s')$, γ

These are the same as shown in section 4.6.1.

Training the virtual robot

To train the robot, a virtual environment was setup in MATLAB. Figure 4.17 shows code for the same. This algorithm can be used to train the robot for decision-making for **any** goal point

and **any** obstacle configuration.

```
while( (m_rel ~= 0 || n_rel ~= 0) && (step <= step_max) )
    obst = obst_next;
    [max_cur_act, max_cur_index] = max(state_q_values(:,(m_rel + m_max), ...
                                        (n_rel + n_max), obst(1), obst(2), obst(3), obst(4)));
    m_change = next_rel(max_cur_index, 1);
    n_change = next_rel(max_cur_index, 2);
    m_rel_next = m_rel + m_change;
    n_rel_next = n_rel + n_change;

    if((abs(m_rel_next) > (m_max-1)) || (abs(n_rel_next) > (n_max-1)) || obst(max_cur_index) == 2)
        state_q_values(max_cur_index,(m_rel + m_max),(n_rel + n_max),obst(1),obst(2),obst(3),obst(4)) = -Inf;
    else
        while(1)
            obst_next = randi(2,1,4);
            if(length(obst_next(obst_next(:,:) == 2)) < 2)
                break;
            end
        end
        max_next_act = max(state_q_values(:,(m_rel_next + m_max), ...
                            (n_rel_next + n_max), obst_next(1), obst_next(2), obst_next(3), obst_next(4)));
        dist_line = (abs(m_rel*n_change - n_rel*m_change)/(m_rel^2 + n_rel^2)^0.5);
        dist_goal = ((m_rel^2 + n_rel^2)^0.5) - abs(m_rel*m_change + n_rel*n_change)/((m_rel^2 + n_rel^2)^0.5);
        reward = -( ( 1 + dist_line)*( 1 + dist_goal) );
        state_q_values(max_cur_index,(m_rel + m_max), ...
                        (n_rel + n_max), obst(1), obst(2), obst(3), obst(4)) = reward + gamma * max_next_act;

        m_rel = m_rel_next;
        n_rel = n_rel_next;
        step = step + 1;
     end
  end
```

Figure 4.17: Training code - fixed/dynamic obstacles task

New random **start and end** points and new **obstacle configurations** are chosen when any of the **termination conditions** is satisfied:

1. If the robot reaches the goal point.
2. If the maximum number of steps in each trajectory is exceeded.

The number of steps in each trajectory is limited, as is number of trajectories for training. The robot immediately penalized with a **reward of** $-\infty$:

1. If it tries to explore outside the arena.
2. If it tries to move into a state where an obstacle is present.

These ensure it learns to remain within the designated exploration area, and does not hit any obstacle. As can be seen in the code, the 'state_q_values' state-action Q value function is of size $\Big((2m_{max} - 1) \times (2n_{max} - 1) \times 2 \times 2 \times 2 \times 2 \times \text{size}(\mathcal{A})\Big)$. Once training is complete, the policy chosen is as follows (refer Algorithm 1): $\pi(s) = \arg\max_a Q(s, a)$

4.8.2 Testing

Testing for this task is significantly different from the others. Since the state (refer section 4.8.1) includes whether or not obstacles are present in the robot's environment, the robot's SONAR sensors must be engaged so that it is able to make an accurate estimate of its state. This estimate is based on two major variables in SONAR detection: **Angle and Range**.

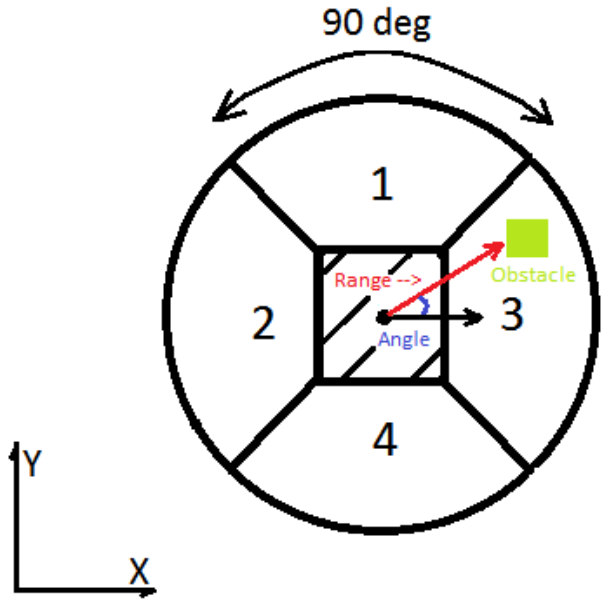


Figure 4.18: Obstacle range and angle detection - mobile robot P3-DX SONAR

1. **Angle:** $Angle$ is the angle from the x-axis of the obstacle nearest to the robot, as shown in Figure 4.18. Let $\angle sonar$ be the detected angle from the robot's SONAR sensors relative to its heading angle. Let $\angle head$ be the robot's heading angle relative to the X-axis. Therefore,

$$\text{Generally, } Angle = \angle sonar + \angle head.$$
$$\text{If } Angle \geq 180, Angle = \angle sonar + \angle head - 360.$$
$$\text{If } Angle \leq -180, Angle = \angle sonar + \angle head + 360.$$

2. **Range:** $Range$, as seen in Figure 4.18, is the distance of the obstacle from the center of the robot.

From Figure 4.18, according to value of $Angle$, the obstacle is assigned to zone $1,\ 2,\ 3,$ or 4, i.e., one of $O_N,\ O_W,\ O_E,\ O_S$ respectively is assigned the value 1, while all others are 0 :

1. $O_N = 1$ for $Angle \in [45, 135)$
2. $O_W = 1$ for $Angle \in [135, 180) \cup (-180, -135)$
3. $O_E = 1$ for $Angle \in [-45, 45)$
4. $O_S = 1$ for $Angle \in [-135, -45)$

Testing in simulation

```
#define PI 3.14159265
#define m_max 10
#define n_max 10
#define edge_size 500

int get_policy_matrix(int policy_mat[][2*(n_max-1)+1][5], int n);
void get_position(ArRobot *robot, double *X_current, double *Y_current);
void get_n_m (double X, double Y, int n_m[]);
void get_n_m_rel(int n_m_current[], int n_m_goal[] , int n_m_rel[]);
void get_row_col_pol(int n_m_rel[], int row_col_pol[], int n_range[], int m_range[]);
int get_mat_pol(ArRobot *robot, ArSonarDevice *sonar, double *range, double *angle);
void get_pol(int policy_mat[][2*(n_max-1)+1][5], int n, int row_col_pol[], int *pol);
void get_n_m_next_policy(int pol, int next_rel[][2], int n_m_current[], int n_m_next[]);
void get_X_Y_next(int n_m_next[], double *X_next, double *Y_next);
void move_to_next(ArRobot *robot, double *X_current, double *Y_current, double X_next, double Y_next, float error);
void turn_to(ArRobot *robot, double AngleToEnd);
```

Figure 4.19: Function headers for testing in simulation and on the physical robot

Once training is complete and a policy matrix learnt, the robot is tested in a simulated environment using MobileSim. As before, the arena is set up using ARIA. Figure 4.19 shows various functions defined to accomplish this. An additional function 'int get_mat_pol(...)' has been defined to return the zone which the obstacle has been found in (refer Figure 4.18), and make decisions based on the policy matrix learnt for that zone.

The robot was tested for multiple goal points in an environment cluttered with obstacles, as shown in Figure 4.20. At each step, it detects its closest obstacles and makes decisions based on this and its learnt policies. Therefore, this algorithm can be used for **any goal points** and **any obstacle configurations.** This means that faced with new goals at any point, the robot will be able to navigate its environment successfully. Therefore, **dynamic obstacle navigation** can be performed using this algorithm.

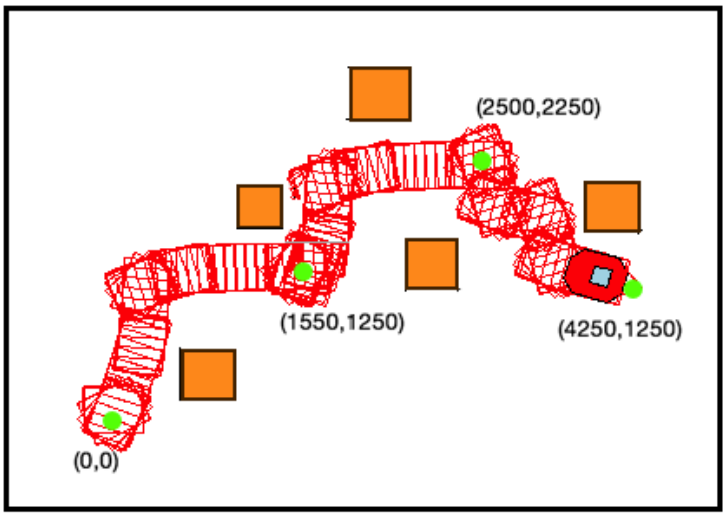


Figure 4.20: Simulation results - obstacle navigation task using SONAR sensors

The obstacles are shown in orange, and the robot navigates the following paths (in x-y coordinates (mm)):

$$(0,0) \rightarrow (1550,1250)$$
$$(1550,1250) \rightarrow (2500,2250)$$
$$(2500,2250) \rightarrow (4250,1250)$$

4.8.3 Challenges and Road Ahead

Challenges faced during the task formulation, training and testing and subsequent goals:

1. Training a state-action Q value function in order to generalize the problem so that the robot is able to navigate given **any goal point and any obstacle configuration** was difficult due to the size of the Q function being:

$$\Big((2m_{max}-1)\times(2n_{max}-1)\times 2\times 2\times 2\times 2\times size(\mathcal{A})\Big)$$

2. Future work possible is outlined in the next chapter.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

In this work, we have addressed the case where the robot is able, by exploration, to independently gather the knowledge necessary to make decisions in unknown situations. Human input is then unnecessary to make decisions. During training, much like the human brain, the robot explores its environment and learns to make the correct decision by receiving rewards (praises or admonishments for its various actions) from the environment it interacts with. These rewards are a property of the environment, and not the agent. Then, based on this knowledge which it has independently gathered during exploration, it is able to make decisions when put in any situation.

This kind of learning-based approach, as opposed to one wherein the agent is directly commanded, makes the problem formulation much more general, as well as harder.

1. **Generality** stems from the fact that the robot may be trained for any task in any situation (environment), given the necessary training period for exploration and reward feedback. This is unlike custom-made commands for specific tasks.
2. **Added complexity** stems from the fact that the agent now assumes no knowledge of the environment.

We have applied reinforcement learning to solve real-world navigation problems in unknown environments using the Pioneer 3-DX robot, with discrete state- and action-spaces. By solving three independent tasks of mounting complexity, we successfully trained the robot to go from **any** initial position to **any** goal position, in an unknown environment with **unknown obstacles**, by engaging its SONAR ranging sensor. This algorithm may also be applied to navigate an environment with **dynamic obstacles.**[1].

[1]Video for testing can be found here - https://www.dropbox.com/sh/fd2b2p1xi6wsjnu/AACxvR9695aX-UGdan80Ta3ba?dl=0

Furthermore, each task was performed in three stages:

1. Training on a virtual robot using MATLAB.
2. Testing in simulation, using MobileSim, a simulator for MobileRobots. The arena is set up using ARIA, the higher-level software interface for MobileRobots platform.
3. Testing on the physical mobile robot Pioneer 3-DX.

5.2 Future Work

5.2.1 Continuous Actions

Currently, our framework deals with discrete action spaces. Continuous action spaces are advantageous because they can yield smooth policies. Not only do such policies reduce power consumption, but they also decrease wear-and-tear and result in better path navigation.

Dealing with continuous action spaces involves generalization through function approximation and discretization in order to reinstate computational tractability. As discussed earlier, the state-action value function complexity depends directly on the size of the action space.

5.2.2 Options to Navigate Dynamic Obstacle Environments

Robot navigation in an unknown environment is a hard task and there are several related approaches to problem-solving of this nature. Su *et al.*, 2004 use fuzzy logic combined with a classifier-system-based neuro-fuzzy inference system called ACSNFIS in order to learn rules through reinforcement learning.

The emerging field of behavioral robotics involves building behavior-based controllers which consist of a collection of behaviors. These basic behaviors are then used to solve a more complicated task. This approach is intimately related to the **options framework** of hierarchical reinforcement learning [Stolle and Precup, 2002]. Here, a set of base policies (options) analogous to a collection of behaviors is used to learn a higher task. These **options can be considered as macro-actions**.

For navigating moving obstacles or for the equivalent task of robot navigation in unknown environments, an example set of options that may be learnt are:

1. **Avoid obstacle** - This behavior causes the agent to avoid collisions with any direct obstacle. The robot turns around until it founds a direction without direct obstacles.
2. **Go to goal** - This behavior allows the agent to move towards the goal point in a space without direct obstacles. The robot first orients itself towards the goal point and then heads towards it.
3. **Follow the wall** - The robot moves alongside a wall (right or left) without colliding with or going to far away from it, until it senses that the obstacle is ending.

The first two options, by themselves are insufficient to navigate non-convex obstacles. The additional option of following the wall is a must in order to navigate obstacles successfully.

REFERENCES

Baird, L. *et al.*, Residual algorithms: Reinforcement learning with function approximation. *In Proceedings of the twelfth international conference on machine learning*. 1995.

Barto, A. G., *Reinforcement Learning: An Introduction*. MIT press, 1998.

Barto, A. G. and **S. Mahadevan** (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, **13**(4), 341–379.

Carona, R., **A. P. Aguiar**, and **J. Gaspar** (2008). Control of unicycle type robots tracking, path following and point stabilization.

Er, M. J. and **C. Deng** (2005). Obstacle avoidance of a mobile robot using hybrid learning approach. *Industrial Electronics, IEEE Transactions on*, **52**(3), 898–905.

Gavrilov, A. V. and **A. Lenskiy**, Mobile robot navigation using reinforcement learning based on neural network with short term memory. *In Advanced Intelligent Computing*. Springer, 2012, 210–217.

Hafner, R. and **M. Riedmiller**, Reinforcement learning on an omnidirectional mobile robot. *In IROS*. Citeseer, 2003.

Hagras, H., **V. Callaghan**, and **M. Colley** (2004). Learning and adaptation of an intelligent mobile robot navigator operating in unstructured environment based on a novel online fuzzy–genetic system. *Fuzzy Sets and Systems*, **141**(1), 107–160.

Juang, C.-F. and **C.-H. Hsu** (2009). Reinforcement ant optimized fuzzy controller for mobile-robot wall-following control. *Industrial Electronics, IEEE Transactions on*, **56**(10), 3931–3940.

Kaelbling, L. P., **M. L. Littman**, and **A. W. Moore** (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 237–285.

Martínez-Marín, T. and **T. Duckett**, Fast reinforcement learning for vision-guided mobile robots. *In Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE, 2005.

Smart, W. D. and **L. P. Kaelbling**, Effective reinforcement learning for mobile robots. *In Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 4. IEEE, 2002.

Stolle, M. and **D. Precup**, Learning options in reinforcement learning. *In Abstraction, Reformulation, and Approximation*. Springer, 2002, 212–223.

Su, M.-C., **D.-Y. Huang**, **C.-H. Chou**, and **C.-C. Hsieh**, A reinforcement-learning approach to robot navigation. *In Networking, Sensing and Control, 2004 IEEE International Conference on*, volume 1. IEEE, 2004.

Sutton, R. S., **D. Precup**, and **S. Singh** (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, **112**(1), 181–211.

Thomaz, A. L. and **C. Breazeal** (2008). Teachable robots: Understanding human teaching behavior to build more effective robot learners. *Artificial Intelligence*, **172**(6), 716–737.