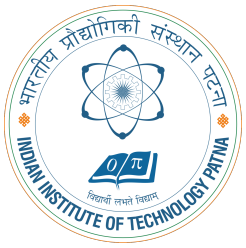
SAC-RL: Continuous Control of Wheeled Mobile Robot for Navigation in a Dynamic Environment

Submitted in partial fulfillment of the requirements
of the degree of
Master of Technology
by

Sahil Sharma
(Roll No. 1811MT14)

Supervisors:
IIT Patna: Dr. Atul Thakur
TU Darmstadt: Prof. Jan Peters
TU Darmstadt: M.Sc Oleg Arenz



Department of Mechanical Engineering
INDIAN INSTITUTE OF TECHNOLOGY PATNA

July 2020

Dedicated to my beloved parents and brother.

Acknowledgments

I want to express my sincerest gratitude to my advisors, Dr. Atul Thakur, head of Mechatronics Instrumentation and Control Lab, M.Sc. Oleg Arenz, Doctoral researcher at IAS and Prof. Dr. Jan Peters, head of Intelligent Autonomous Systems (IAS) without there support and valuable feedback, this thesis would not have been possible. I would like to thanks the DAAD for providing me the scholarship under the KOSPIE program.

During all the stressful times, my family members were the continuous source of inspiration and strength. It is only because of their presence that I have been able to survive in those difficult times. I want to dedicate this work to my parents; this is their hard work, which is reflected in my work. I want to thank my brother Aman Sharma for his selfless contribution, which always cheers me up in difficult times.

The Mechatronics batch of '18 made my one year at IIT Patna unique and comfortable. They are the symbol of the Ideal group to work with. I have learned from every person and borrowed their positive attributes. My Lab mates at IAS helped me a lot to settle down in a new country. Whenever I stuck in my work, they guide me to solve those problems. A big thanks to my best friend, Quentin Delfosse, who especially helped me to find all the essential contact needed to build my project.

During my stay in Germany, Mrs. Ursula Kleinschmidt taught me German language up to the A2 level, which helped me in my day to day life.

Thank you to all the people who came in contact with me during my Master's research project.

Sahil Sharma

Certificate

This is to certify that the thesis entitled "SAC-RL: Continuous Control of Wheeled Mobile Robot for Navigation in a Dynamic Environment", submitted by Mr. Sahil Sharma to Indian Institute of Technology Patna, is a record of bonafide research work under the supervision of Dr. Atul Thakur, Associate Professor, IIT Patna, Dr. Jan Peters, Professor, TU Darmstadt and M.Sc. Oleg Arenz, PhD Student, TU Darmstadt and we consider it worthy of consideration for the degree of Master of Technology of this Institute. This work or a part has not been submitted to any university/institution for the award of degree/diploma. The thesis is free from plagiarized material.

Date:

Dr. Atul Thakur
IIT Patna

Prof. Jan Peters
TU Darmstadt

M.Sc. Oleg Arenz
TU Darmstadt

Certificate of Approval

Date: July 9, 2020

This is to certify that the thesis entitles "SAC-RL: Continuous Control of Wheeled Mobile Robot for Navigation in a Dynamic Environment", submitted by **Sahil Sharma** (Roll No. 1811MT14) to the Indian Institute of Technology Patna for the award of the degree of Master of Technology has been accepted by the examination committee and that the student has successfully defended the thesis in the viva-voce examination held today.

(Supervisor) **(External Examiner)** **(Internal Examiner)**

Declaration

I certify that

a. The work contained in this thesis is original and has been done by myself under the general supervision of my supervisor/s.

b. The work has not been submitted to any other Institute for degree or diploma.

c. I have followed the Institute norms and guidelines and abide by the regulation as given in the Ethical Code of Conduct of the Institute.

d. Whenever I have used materials (data, theory, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and providing their details in the reference section.

e. The thesis document has been thoroughly checked to exclude plagiarism.

Sahil Sharma

Roll No. 1811MT14

Date:

Abstract

Autonomous navigation in a complex environment is essential for deployment in the real world. Traditional navigation approaches assume the environment as static objects, resulting in a non-reasonable behavior. Mobile robots should be able to cope with dynamic objects as well as dynamic crowds. In this work, we present a continuous control for a skid-steer drive wheeled mobile robot to navigate in a dynamic environment using the soft-actor critic algorithm clubbed with a global planner. Here the mobile robot takes pose and a range vector of 18-dimension from a 360-degree LIDAR sensor as input and gives continuous control commands in the form of RPS as output. We divide our controller into the upper and lower level controller. The lower-level controller uses the soft-actor critic algorithm, and the upper-level controller uses the A-star global planner algorithm. The A-star algorithm takes OGM (Occupancy Grid Mapping) data from SLAM in real-time and proposes a path consist of via-point to the final goal point. The SAC based lower-level controller trained in a dynamic environment, navigate towards the immediate via-point. Every time the robot reaches the immediate via-point, A-star computes a new path based on OGM data. The use of a global planner eliminates the need to use LSTM or RNN policies for the SAC based controller, thus making training easy. The experiments have shown that the proposed controller can navigate to the desired targets without colliding with any dynamic obstacles.

Contents

Abstract **i**

List of Figures **v**

List of Abbreviations **viii**

List of Symbols **ix**

1 Introduction **1**

1.1 Motivation 1

1.2 Background 2

1.3 Thesis Objective 6

1.4 Contribution 6

2 Literature Review **7**

2.1 SLAM Based Navigation 8

2.2 Deep-Learning Based Navigation 8

2.3 Deep Reinforcement Learning Approach 9

2.4 Research Gaps 11

3 Problem Statement and Approach Overview **13**

3.1 Problem Statement 13

3.2 Approach Overview 14

4 Environment and Setup **16**

4.1 PyRep 16

4.2 Environment 18

4.3 MDP Formulation 19

4.3.1 State/Observation Space 19

4.3.2 Action Space 20

4.3.3 Reward Functions 22

4.4 Implementation 23

4.5 Approach-1 26

4.5.1 Observation Space Definition 26

4.5.2 Action Space Definition 27

4.5.3 Reward Function Definition 27

4.5.4 Limitation 28

4.6 Final Proposed Approach 30

4.7 Experiment 31

4.8 Result 32

4.9 Evaluation 33

5 Assembly of skid-steer drive wheeled robot **36**

5.1 Torque Calculation and Motor Selection 37

5.2 Selection of Development Board 38

5.3 Selection of LiDAR Sensor 39

5.4 Miscellaneous Components 40

5.5 I2C Communication 40

5.6 Circuit Diagram 41

5.7 Action-Space in Real Hardware 44

5.8 Planned Experiments 45

6 Conclusion **46**

6.1 Future Work 47

List of Figures

3.1 Orientation of Robot with respect to the Goal 14

4.1 A skid steering wheeled robot consisting of a 360-degree LIDAR sensor. It is one of the dynamic environment designed on V-Rep, which is simulating a restaurant scenario. Here humans are the moving obstacles. 17

4.2 Figure (a) shows Env-1, consisting of static obstacles with two rooms with a narrow door connecting both rooms. Figure (b) shows Env-2, which simulates a restaurant scenario with static obstacles such as tables, chairs, poles, and plants. This environment also includes 5 pedestrians. 18

4.3 Combination of continuous actions provided to the robot 21

4.4 It is showing the overall Reward function trajectory computed for 2000 episodes. Here, the agent started learning after 500 episodes, but the policy is not stable due to discontinuity in orientation data at (0 and 2π), making the policy unstable. 23

4.5 Plot of sinusoids, which represent the euler angle in continuous form. 24

4.6 It is showing the overall Reward function trajectory computed for 2000 episodes. Here, the agent started learning after the 170th episode, but the policy is not stable due to the decoupled action space. 24

4.7 It is showing the overall Reward function trajectory computed for 2000 episodes. Here, the agent started learning after the 120th episode, and the policy got stable after 1550 episodes. 25
4.8 Showing selection of 18 points from 360 available data points 27
4.9 Limitation of the proposed approach 29
4.10 It is the flow chart of the proposed approach. It shows an Upper-level controller based on A-star and a Lower-level controller based on the trained policy on the SAC algorithm. The red dots are the via-points generated by the A* algorithm. 31
4.11 Figure4.11a shows the training curve for the static environment. The agent learned the policy in 1000 episodes. Also, learning is not stable. Figure 4.11b shows the training curve for the same static environment with lower action frequency. The effect of low action frequency improved the learning speed and made the policy more stable. The agent learned in 100 episodes. Figure 4.11c shows the training curve for the dynamic environment. The agent used the previous model trained on the static environment and learned the policy in 70 episodes. 32
4.12 Frame grabs from evaluation video showing the path followed by the agent to achieve the goal. 33
4.13 It is showing the comparison of different controllers. Fig 4.13a shows the controller based on the RL policy, which fails to achieve the goal. Fig 4.13b shows our proposed controller, which is based on a global planner that is the upper-level controller and lower-level controller based on RL policy. 34

5.1 Dimension of the robot 37
5.2 (a) Nvidia AGX Xavier development board. (b) A 360°LiDAR sensor 39
5.3 Circuit diagram of the robot 42
5.4 Snapshot of the Hardware 43

5.5 Circuit on the real hardware 43
5.6 Regression curve between RPS and the PWM for DC Motor 44

List of Abbreviations

DQN	Deep Q-Networks
A2C	Advantage Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
DPG	Deterministic Policy Gradient
DDPG	Deep Deterministic Policy Gradient
SAC	Soft Actor-Critic
SARSA	State Action Reward State Action
MDP	Markov Decision Process

List of Symbols

θ	Weight of the Policy Network
w	Weight of the Value Network
α	Learning Rate
Y	Output of the Neural Network
∇_w	Gradient w.r.t weights
λ	Hyper-parameter for Regularization
S	Set of States
A	Set of Actions
P	State Transition Probability Matrix
R	Reward Function
γ	Discount Factor
δ_t	Magnitude of TD-error
π	Policy

Chapter 1

Introduction

This chapter provides a general introduction to the topic. Section 1.1 present motivation for solving the navigation problem using reinforcement learning. Section 1.2 gives the background for this work. Then section 1.3 provides the objective of this thesis, followed by section 1.4, which gives the contribution of this work.

1.1 Motivation

Today the world is moving towards automation. Moreover, in the case of automation, robots are playing a huge role. Specifically, there are two types of robots- stationary and mobile. In the latter case, navigation is the key component to solve the automation problem. We mainly use AGVs (automated guided vehicle) on the industrial shop floor or warehouse. AGVs follows the laid physical guided path on the floor (wires, paint, tape) or a free-range path where there is no physical guidance placed on the floor [1]. If we take the example of roads, there are a set of traffic rules if followed accurately; the chances of an accident are scant. However, until today, the traffic rules are followed by a human, which introduce a little uncertainty in the environment. In all these edge cases, we use in-depth learning-based perception methods [2]. Nevertheless, we use these autonomous navigation systems in freeway where the

chances of humans or animals as an obstacle are meager. Finally, if we take a scenario of an indoor environment such as home, restaurant, or hospital, there is no set of rules for navigation for the robots and humans. Thus the factor of uncertainty increases [3]. The method like SLAM based navigation is prevalent for the indoor environment only for the static world. The SLAM based methods need a prior map of the environment for effective navigation. Also, this method consumes time to build or update the map and act accordingly [4].

1.2 Background

Autonomous cars, unmanned rovers, drones, and so on are great examples of mobile robots. According to their locomotion system, mobile robots are classified into various categories such as stationary(arm/manipulator), land-based (wheeled, walking, slip/skid, hybrid), air-based, water-based and other [5]. Amongst these, wheeled robots are easier for navigation than using treads or legs on flat and non-rugged terrain. They are easier to design, build, program, and there are no balancing issue [5]. Skid-steer drive locomotion is widely used on tracked vehicles such as bulldozers and tanks. It is also used on many wheeled vehicles such as four and six wheels [6]. The relative velocities of the left and right side wheels are used to steer in a skid-steer drive vehicle. Thus, the slipping of wheels with respect to the ground is required for turning [7]. This drive is energy inefficient due to the slipping with respect to the ground, and controlling is difficult when compared with the differential drive robot. Moreover, the tires wear out faster. However, with all the above shortcomings of the skid-steer drive vehicles, it has higher maneuverability like a differential steering [8].

Mobile robot navigation is the most extensively studied problem in the field of robotics. It is classified into three categories: global navigation, local navigation, and personal navigation. For global navigation, prior information on the environment, obstacles, and goal are required. It includes algorithms such as A*, Dijkstra, Artificial potential field method, and cell decomposition [9] [10]. Local navigation is a reactive approach as it deals with the dynamic

objects in the environment and the position of the robot with respect to the other elements. It includes algorithms such as Neural networks, Fuzzy logic, Neuro-Fuzzy, and the Genetic algorithm [9] [10].

Neural network methods for local navigation includes Deep Learning (DL) and Deep Reinforcement Learning (DRL). Deep learning takes the labeled data and extracts the pattern for navigation in that scenario. However, the robustness of the model depends on the quality and quantity of the labeled data used for training. Moreover, it is painstaking to collect the labeled data for all the scenarios in any environment, which makes DL unsuitable for solving the navigation problem in many cases [11]. Nevertheless, DRL does not require labeled data for learning. DRL learns from the experiences gathered by the interaction with the environment. Thus, the agent collects all the edge cases and learns from those experiences [11].

Reinforcement learning is old and can be dated back to the early 1980s when it is famously known as animal learning as the animal learns by doing trial and error. During that time, there was another approach called optimal control, which was solved using value functions and dynamic programming. The concept of dynamic programming given by Richard Bellman is famously known as the Bellman equation, which leads to the modern reinforcement learning called temporal-difference learning [12]. Finally, in 1989 Chris Watkins developed a Q-learning algorithm [13], which was able to solve the control problem consisting of discrete action-space and tiny observation-space because of the limitation of the Q-table. Nonetheless, it was a great achievement on its own as the agent was able to solve the task merely based on the reward function without any information about the world. Reinforcement learning again gained popularity in 2016, due to the breakthrough achievement in the neural network and GPUs. DeepMind group was able to integrate Q-learning with the neural network for estimating the Q-values called Deep Q-Network (*DQN*); thus, there is no need of Q-table. The above achievements open a wide array of opportunities in terms of observation-space. In 2016, DeepMind showed an RL agent achieving human-level performance in playing Atari 2600 video games [14]. *AlphaGo* [15] was the second success of reinforcement learning in

2016. They developed an algorithm which combines the Deep reinforcement learning (*DRL*) and tree search method to solve the game of Go, which was challenging to solve artificially due to vast observation-space. The agent beats the world champion by 4-1, which is impressive because the agent was initially trained on recorded amateur videos and later on learned by playing against itself.

In reinforcement learning, we have either discrete or continuous action. In the case of discrete action space, an agent can choose from a distinct set of actions, such as control modes, gear switching, and so on. There is no need of a function approximator to find the optimal discrete action. The major disadvantage of discrete action space is its inability to represent the agent's action-space accurately. In the case of continuous action space, the chosen action is a real-valued vector, such as velocity setpoints, control gains, or analog outputs. It gives more control to the agent to operate the system [16].

DQN is a value-based model-free algorithm. After the DQN achievements, the researcher introduced a policy-based method which opens the domains for continuous action-space. The policy is a function that takes states as input and output the desired action. The RL algorithms can be classified into actor-only, critic-only, and actor-critic methods. The critic is the value function and uses temporal difference learning to find the action leading to an optimal value. The actor is the policy function that works with the parameterize policies on which optimization procedure is applied directly [17]. The policy-based methods use a policy gradient approach that uses continuous action but with poor optimization. This leads to actor-critic methods, which is the combination of actor and critic based approach, thus bringing the advantage of both the methods. Here the actor brings the advantage of computing continuous action without the need for optimization because the critic evaluates the action by computing the value function [17]. Several policy-based algorithms, such as Trust Region Optimisation (TRPO) [18], Proximal Policy Optimization (PPO) [19], Asynchronous Advantage Actor-Critic (A3C) [20] which are on-policy methods, require a new sample for every episode, that makes these algorithms suffer from sample complexities.

The issue of sample complexities is solved by using off-policy-based algorithms that uses experience stored in the replay buffer, thus providing a wide range of uncorrelated data. A commonly used off-policy gradient-based approach is Deep Deterministic Policy Gradient (DDPG) algorithms. It removes the sample complexity issue, but it is challenging to implement due to its brittleness and hyperparameter setting [21]. Soft actor-critic (SAC) solves the brittleness problem of these algorithms and also makes the entropy coefficient automatic. It combines the standard maximum reward function with the entropy maximization term. It uses a function approximator, a neural network trained to output the mean and variance of the action. Thus, it allows the agents to be stochastic during training. It can take any action based on the network's output variance and be deterministic during policy evaluation by taking the mean of the output action [22] [23].

Now coming back to the topic of navigation, an agent has to deal with two kinds of environment. These are static and dynamic environments. An environment is static only if the action of the agent modifies the environment; if the environment is changing without the constant engagement of the agent, then it is a dynamic environment [24]. The static environment is easy as an agent's constant interaction is not required; the agent can take its action without looking to the environment and worrying about the passage of time. However, in a dynamic environment agent's constant interaction with the environment is required, the agent has to worry about the passage of time, and no decision is interpreted as doing nothing. Thus navigation in a dynamic environment is challenging because of the added uncertainty. Furthermore, navigating for long-distance using local RL-policy is another challenging problem. A task is long-range when a robot navigates to a considerable distance. The long-range navigation task consists of two-part: finding a long-range collision-free path with via-points and a local robot control capable of avoiding dynamic obstacles [25]. A local robot controller uses sparse reward during training, which leads agents to the local minima, making the agent either challenging to train or successful for short-range distances.

The deep reinforcement learning (DRL) algorithm requires constant interaction with the en-

vironment to gather experiences for learning. Therefore, implementing this on real hardware is very expensive and time-consuming. The solution to this problem is doing simulation first and then transferring the model to the real world. V-REP by CoppeliaSim is a versatile and scalable platform for doing robotics simulation. It is an integrated development environment having support for ROS and can be controlled by C/C++, Python, Java, Lua, MATLAB, or Octave. PyRep [26] 2019, which is a python library explicitly developed for reinforcement learning. PyRep makes the environment 10,000x times faster than conventional Python APIs.

1.3 Thesis Objective

In light of the motivation and the background as discussed above, we summarize the thesis's objective: -

1. To train a policy for continuous control of a mobile robot for navigation in a dynamic environment using a soft actor-critic RL algorithm.
2. To build a skid-steer drive four-wheel robot capable of doing online learning.

1.4 Contribution

In the background of the earlier work, the contribution of the thesis is summarized as follows:-

- We have proposed an approach for navigating in a dynamic environment for long-range goals.
- We have found an approach based on lowering the action frequency makes the SAC based agents learn faster.
- We have build a skid-steer drive four-wheel robot for doing online learning in the real world.

Chapter 2

Literature Review

Path planning aims to find a sequence of actions that transform from a given initial state to the desired goal state. The states are the agent's location or position, and the actions are the transition allowed between the states. The path found by the algorithm is optimal when the sum of the transition cost is minimal across all the possible paths. A planning algorithm is complete when it finds the optimal path in finite time and will let us know if it does not exists. Thus planning a path can be categorized as search problem on graphs. The most famous algorithms for calculating the least-cost paths are Dijkstra's algorithm (Dijkstra 1959) and A* (Hart, Nilsson, and Rafael 1968; Nilson 1980). Both algorithm return an optimal path and are a particular form of dynamic programming (Bellman 1957) [27]. A*, on the other hand, considers the most promising states, thus saving considerable computation time. In the case of a dynamic world, the path generated by agent based on the initial information can be invalid or suboptimal. Thus the agent needs to compute a new path based on the updated information. However, it is computationally expensive to replan every time a new path from scratch. [27].

2.1 SLAM Based Navigation

Simultaneous Localization and Mapping (SLAM) problem arises when the robot has no information of its poses or the map of the environment. Thus in SLAM, a robot builds a map of the environment while simultaneously localizing itself relative to this map. SLAM with visual localization experiences a problem of illumination variance because places have different appearances at different times of the day, months, and mostly along the seasons [28]. A technique developed by Maddern *et al.* [29] and McManus *et al.* [30], transforms the images to an illuminant invariant color space. Thus, intending to improve the performance for place recognization in challenging illumination conditions. Jared Le Cras *et al.* investigates the applicability of the color model. The idea is to separate the brightness from chromaticity to eliminate the features and matches that may have occurred due to dynamic illumination [31]. The above problem occurs when static obstacles cast dynamic illumination, making localization complicated. It becomes even worse when a dynamic object is casting dynamic illumination [31]. Another problem with SLAM is the Kidnapped robot problem. It is similar to the global localization problem but more difficult to solve. Because in this robot believes that it knows where it is, but in reality, it does not [32]. Another problem with SLAM is navigation in dynamic environments because computational complexity increases in a dynamic environment [33]. Moreover, moving obstacles causes data association error, failure in landmark detection, and failure in loop closure [34].

2.2 Deep-Learning Based Navigation

Due to the availability of computational power, researchers are shifting towards deep learning to solve the navigation problem. Yann LeCun *et al.* used supervised learning to map the outdoor data collected by the human driver on a vehicle using a two forward-pointing camera

to a set of possible steering angle [35]. Another work from Lei Tai *et al.* and Chenyi Chen *et al.* trained a CNN model to map the indoor data gathered from the stereo camera mounted on the turtle bot to 5 discrete actions [36] [37]. But the control action in all these approaches are discrete or high level like *left* and *right*

Chao Yu *et al.* purposed the DS-SLAM approach, which combines semantic segmentation network with moving consistency check to separate the dynamic section of the scene such as moving objects to improve the localization accuracy in SLAM. The algorithm runs five threads in parallel to do semantic segmentation, tracking, loop closing, local mapping, and dense semantic map [38]. Another approach called ML-RANSAC introduced RANdom-SAmple Consensus (RANSAC), which used MTT (multi-target tracking) to detect features and to differentiate dynamic from static objects. It was clubbed with the Extended Kalman filter (EKF) to fuse the LIDAR and vision sensor to detect object and depth information [39]. But still, deep learning approaches are limited to the quality and quantity of data and suffers from the generalization [40].

2.3 Deep Reinforcement Learning Approach

Deep reinforcement learning has a significant application in control tasks in robotics [41]. Christopher J.*et al.* [13] showed Q-learning converges to the optimum action-value when all the action-values are discrete. Lv Qiang *et al.* built an end to end solution for navigation in a static environment. He used a turtle bot equipped with a lidar sensor as input to the model, and the output actions were divided into 5 different actions. They have shown that their approach outperforms the traditional navigation method Dijkstra's algorithm clubbed with the local planner dynamic window approach [42]. Yang Liu *et al.* used q-learning with OS-ELM (online sequential extreme learning machine), which improves the efficiency and speed of Q function approximation [43]. Lei Tai *et al.* used the DQN framework with a raw image from the RGB-D camera as an input to the CNN policy, which gives discrete

action to the robot. They tested their approach in an indoor environment. The test result demonstrated that their method outperforms the controller from supervised learning. Also, their model trained on simulation image performs directly in the real world [44]. Jingwei Zhang *et al.* proposed a successor-feature-based DRL algorithm to transfer learning from the learned model to a fresh one. They used an improved version of DQN called SF-RL (Success Feature Reinforcement Learner) with discrete action-space [45]. Another work from Shumin Feng *et al.* used different DRL methods such as DQN, DDQN, and DDQN-PER with discrete action-space in a static environment shows that DDQN-PER performs better in a new environment than DQN and DDQN [46]. Xiaogang Ruan *et al.* showed effective navigation using Dueling DQN, Double DQN, and D3QN (Deep Double DQN). The input is a single RGB-D image, and the action-space is discrete, D3QN perform better than Dueling DQN and Double DQN [47]. Gregory Kahn *et al.* proposed an R.L. algorithm based on generalized computation graphs. The generalized computation graph encompassed model-free methods and model-based methods and was trained as a supervised learning problem. The algorithm is capable of learning from the raw depth image as input and is sample efficient. They have shown that their algorithm outperforms the N-step double Q-learning approach [48].

A3C is an asynchronous variant of A2C reinforcement learning algorithms. It has shown the stabilizing effect by making the actor parallel during training. It runs the parallel instances of the same environment, thus making learning fast [20]. Research from DeepMind showed navigation in a static environment with a single depth image as input. They used the A3C algorithm with the LSTM policy. The action space is discrete. They have shown that their approach reaches human-level performance even under conditions where the goal location changes frequently [49]. Yuke Zhu *et al.* used a similar approach as done by DeepMind, but their policy takes goal and current state as input, which leads to generalization. The action space here is also discrete. They trained their network on their AI2-THOR, which gives high-resolution 3D scenes making learning easily transferable to real hardware. They have shown that their approach converges faster than the state of the art DRL methods [50]. Similar work

from Matej Dobrevski *et al.* [51] for mobile surface robot and Yushan Sun *et al.* [52] for the underwater vehicle used an actor-critic framework for navigation in a static environment with discrete action-space. Michael Everett *et al.* used A3C with collision avoidance based on social norms [53] called the GA3C-CADRL algorithm. They used the LSTM policy with a discrete action-space. The LSTM policy made the algorithm to take observation of other agents, instead of fixed observation size. The proposed algorithm outperforms the vanilla A3C algorithm in dynamic environment [54].

Research work from Aleksandra Faust *et al.* from Google Brain showed an approach of combining the Probabilistic Roadmaps algorithm with reinforcement learning (PRM-RL) for long-range navigation tasks. They used the DDPG (Deep Deterministic Policy Gradient) algorithm with continuous control action for the static environment. They used a 220° LIDAR sensor combined with a relative robot position with the goal as input states. They have shown that their PRM-RL approach completes up to 215m long trajectory in the indoor environment. Taiping Zeng used Proximal Policy Optimization (PPO) with continuous state and action space. A static environment without any obstacle is used for testing the algorithm [55]. Another research work from Lei Tai *et al.* used the DDPG RL algorithm with a 10 dimension LIDAR sensor as input state and continuous action-space. The robot successfully navigates in a static environment without colliding with the obstacles [56].

2.4 Research Gaps

We found that most of the work in mobile robot navigation using reinforcement learning solves the navigation problem in discrete action-space in a static environment. There are some research work implementing continuous action-space but in a static environment. Following are the research gaps we found in this domain: -

- We didn't find any work using continuous action for mobile robot navigation in the

dynamic environment.

- We found improvement in the RL algorithm by implementing LSTM policies. Considering the advantage of LSTM policies, it will be interesting to see the improvements it will bring on the recent off-policy based soft actor-critic algorithm.

Chapter 3

Problem Statement and Approach Overview

3.1 Problem Statement

Given

1. The relative position (x, y) of the mobile robot with respect to the goal see figure 3.1.
2. The orientation α of the mobile robot with respect to the goal along the LOS see figure 3.1.
3. A range vector from a 360° 2D lidar sensor.
4. The previous actions of the mobile robot are used as encoder values.
5. The continuous action of the mobile robot as angular velocity ω of each wheel.

The thesis's objective is to train a policy $\pi(a_t|s_t)$ using a soft actor-critic algorithm for continuous control of skid steer wheeled mobile robot for navigation in a dynamic environment.

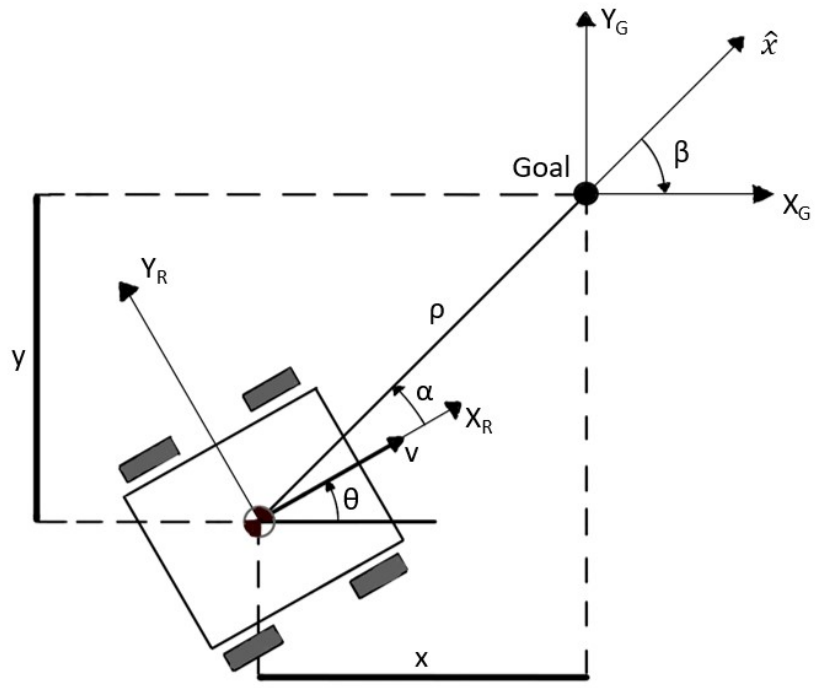


Figure 3.1: Orientation of Robot with respect to the Goal

3.2 Approach Overview

To solve the objective enlisted section 3.1. We employed the following steps: -

1. We first train the agent in the simulation environment with no obstacles to find the best performing observation-space and action-space.
2. After this, we formulate the collision penalty function and reward function for the agent, to train it in the static environment.
3. Then we increase the environment's complexity by introducing pedestrians and providing the stack of four observations to deal with the moving objects as they imply time-dependent decisions.

4. Since for indoor navigation task, we deal with the sparse rewards, especially over the long-range navigation over the challenging environment. Sparse reward makes the agent either challenging to train or prone to local minima [25]. To solve this, we integrate a global planner A* with the local SAC based RL policy.

The steps mentioned above are explained in detail in chapter 4.

Chapter 4

Environment and Setup

This chapter discusses the environment and setup used for the experimentation. Section 4.1 discusses the PyRep library for V-Rep specifically developed for Reinforcement Learning. Section 4.2 shows the static and dynamic environment. Section 4.3 discusses the MDP formulations. Section 4.4 shows the implementation results of the various MDPs. Section 4.5 discusses the first Approach and its limitations. Section 4.6 discusses the Final Proposed Approach for this work.

4.1 PyRep

V-REP (Virtual Robotics Experimentation Platform) is a versatile platform for the rapid prototyping of robots. It has the built-in physics engine Bullet, ODE, Newton, and Vortex and a set of customizable robot building interface [57]. However, although it is controllable by using several API, including python API, the data collection is very slow in vrep. Thus, it restricts the vrep to be used for large scale data collection applications [26].

Modification: -

The 6 remote API usually suffer from 2 communication delays. One of these delays is the

socket communication delay between the remote API and the simulated environment. The second delay and the most notable one is the inter-thread communication delay, which is significant when the python script needs to modify the state in a vrep simulated environment during learning. To alter this latency, the author changes the base code of the vrep and offers direct control of the vrep simulation loop to python API, thus making the simulation 10,000x faster [26]. We built our learning environment in V-REP with OpenAI GYM as the backend. A dynamic environment is shown in Figure 5.1

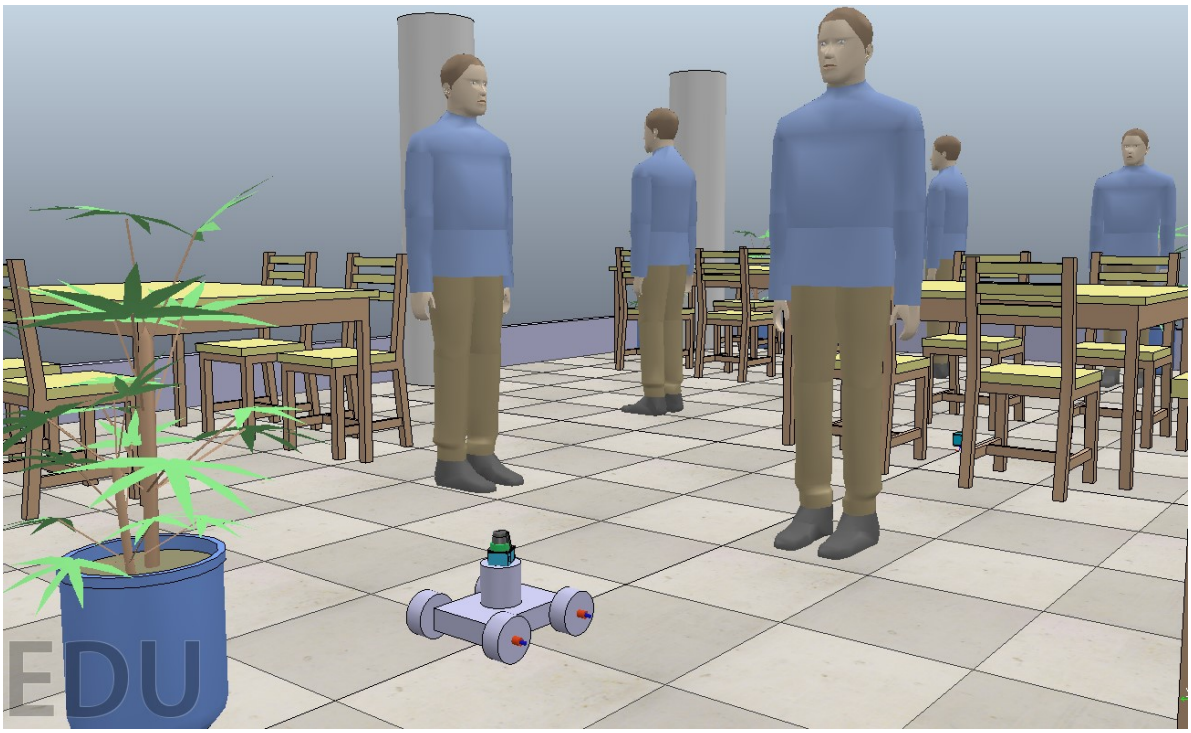
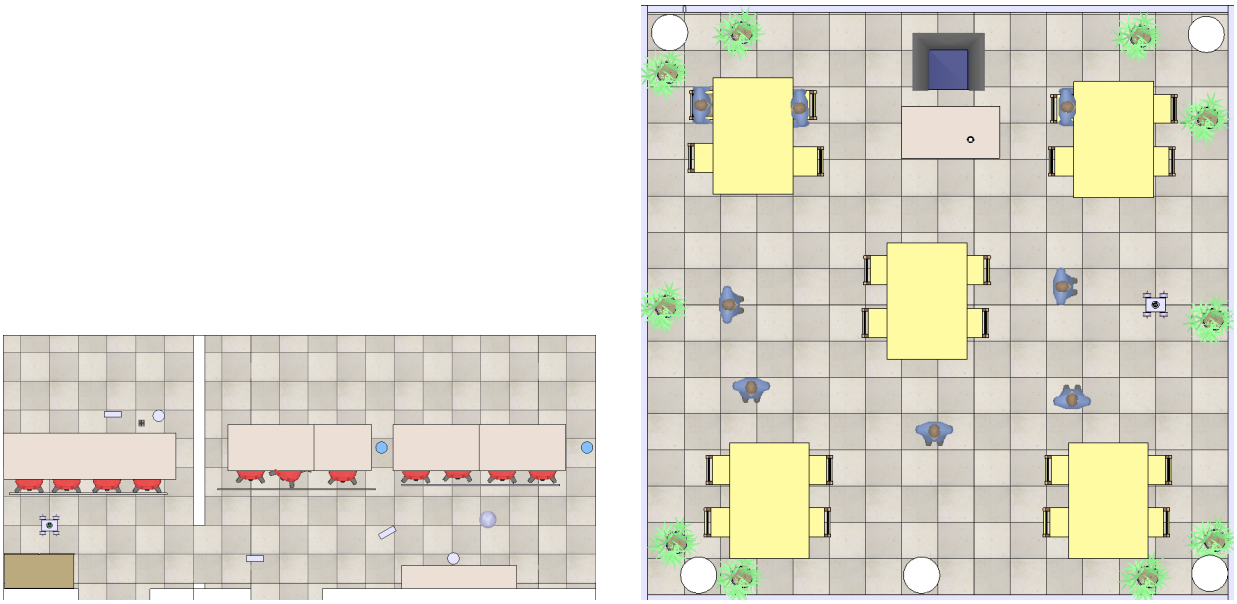


Figure 4.1: A skid steering wheeled robot consisting of a 360-degree LIDAR sensor. It is one of the dynamic environment designed on V-Rep, which is simulating a restaurant scenario. Here humans are the moving obstacles.

4.2 Environment



(a) (b)

Figure 4.2: Figure (a) shows Env-1, consisting of static obstacles with two rooms with a narrow door connecting both rooms. Figure (b) shows Env-2, which simulates a restaurant scenario with static obstacles such as tables, chairs, poles, and plants. This environment also includes 5 pedestrians.

We built two environments on vrep controlled by pyrep for the agent to train its policy. Our first environment is one with static obstacles. For the static obstacle, we used all the pure shapes available to us in vrep. Apart from that, we also used standard furniture as static obstacles. The scene consists of 2 rooms connected by a door. The idea of introducing a door in the scene is to enable the agent to navigate in the narrow passages. This idea also leads us to formulate the collision penalty function explained in the later section.

Our second environment is one with moving obstacles. For that, we introduced pedestrians to

our environment. These pedestrians are not following any social norms for interacting with the world; they are just walking randomly. Our scene mimics the restaurant scenario as it consists of chairs, tables, pole, and plants as static obstacles.

4.3 MDP Formulation

An MDP is typically defind by a 4-tuple (S,A,R,T)

where,

S is the state/observation space of an environment.

A is the action space for the agent.

R(s,a) is the reward function which compute the reward for the action a in state s.

T(s'|s,a) is the transition probability from state s to s' by taking an action a.

Here, our goal is to find a policy π that maximize the expected future (discounted) reward.

4.3.1 State/Observation Space

Here in V-Rep currently we have 3 state-space combinations for experimentation.

- **Relative distance of the robot with respect to the goal with the encoder values**
  The state-space is :-

$$state\ space = (x, y, \alpha, e_1, e_2, e_3, e_4) \tag{4.1}$$

  where,
  x is the relative distance of robot in x-axis with respect to the goal

y is the relative distance of robot in y-axis with respect to the goal

α is the orientation of robot with respect to the goal shown in Fig 3.1.

e_1, e_2, e_3, e_4 are the encoder reading(angular velocity) of the wheels.

- **Relative distance of the robot with respect to the goal and orientation as sine and cosine with the encoder values**

  The state-space is :-

$$state\ space = (x, y, sin\alpha, cos\alpha, e_1, e_2, e_3, e_4) \tag{4.2}$$

- **Relative distance of the robot with respect to the goal with the previous action as encoder value**

  Here the previous action taken by the robot is the present state of each wheel. Thus using this data as the state space for each wheel will help eliminate the need for the encoder in our robot. The state-space is:-

$$state\ space = (x, y, sin\alpha, cos\alpha, e_1, e_2, e_3, e_4) \tag{4.3}$$

4.3.2 Action Space

In reinforcement learning, we have either discrete or continuous action. In the case of discrete action space, an agent can choose from a distinct set of actions. There is no need for a function approximator to find the optimal discrete action. The major disadvantage of discrete action space is its inability to represent the agent's action-space accurately. In the case of continuous action space, the chosen action is a real-valued vector and gives more control to the agent to operate the system [16]. Therefore in this work, we have investigated the application of

continuous action space for navigation in a dynamic environment.

We selected two types of action space; for characterization, we named it coupled and decoupled action space. In decoupled action-space, the action of each wheel is independent of each other. This action-space is used in all the previous work. The second action-space we considered is coupled action-space, where the actions are shared with each wheel. Following are the action-space: -

- **Action Space-1 (Decoupled action space): -** This is the simplest one where each action is the angular velocity of each wheel. Wheels at each side have given the same velocities. Thus we have only 2 continuous actions with a range from -5, 5. The action space is:-

$$action\ space = (action_1, action_2) \tag{4.4}$$

- **Action Space-2 (Coupled action space): -** In this, the combination of A_1 and A_2 have given to each wheel.

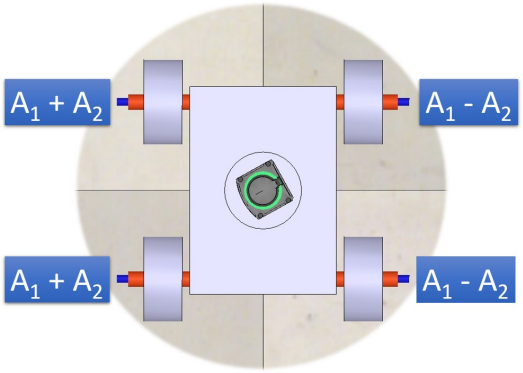


Figure 4.3: Combination of continuous actions provided to the robot

Thus for forward or backward motion, only A_1 action is needed, and for rotation about

CG, only A_2 action is required.The action space is:-

$$action\ space = (A_1, A_2) \tag{4.5}$$

where,

$A_1 + A_2$ and $A_1 - A_2$ are the continuous action for the left and right side wheels respectively ranging from -5, 5

4.3.3 Reward Functions

There are various reward functions used for the training the policy.

1. The negative L2 norm between robot and goal

$$Reward\ function = -\sqrt{x^2 + y^2} + r_{terminal} \tag{4.6}$$

where,

x and y are the relative distance between goal and robot

$r_{terminal}$ = 10000, when goal is reached

$r_{terminal}$ = 0, otherwise

2. The negative L2 norm between robot and goal with action penalty and smoothness function.

$$Reward\ function = -\sqrt{x^2 + y^2} - \sum(A) - (A - prev_A)^2 + r_{terminal} \tag{4.7}$$

where,

x *and* y are the relative distance between goal and robot

$\sum(A)$ is the action penalty to stop agents to take high action every time.

$(A - prev_A)^2$ is the smoothness reward which gives penalty for every sudden action.

4.4 Implementation

1.

Setup	
State Space	$(x, y, \alpha, e_1, e_2, e_3, e_4)$
Action Space	$(action_1, action_2)$ || Decoupled Action-Space
Reward Function	$-\sqrt{x^2+y^2} - \sum(A) - (A - prev_A)^2 + r_{terminal}$

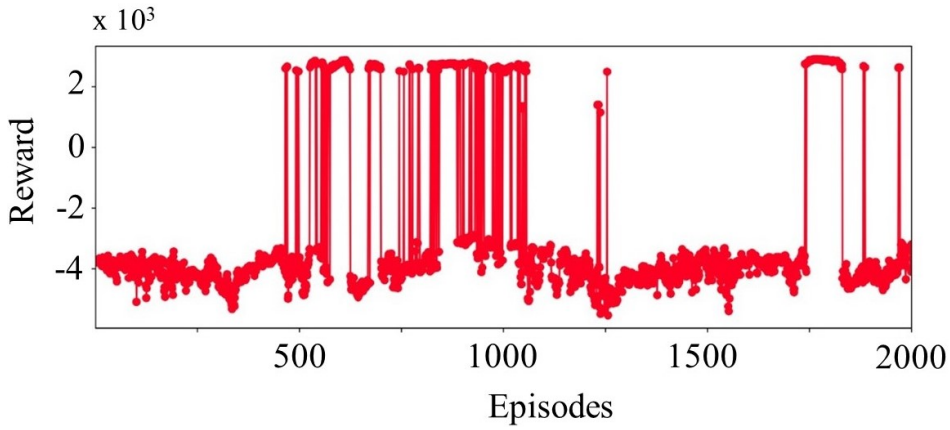


Figure 4.4: It is showing the overall Reward function trajectory computed for 2000 episodes. Here, the agent started learning after 500 episodes, but the policy is not stable due to discontinuity in orientation data at (0 and 2π), making the policy unstable.

The above problem of discontinuity at (0 and 2π) is solved by using sine and cosine function as the sinusoids are continuous and periodic in the domain (0 to 2π). Thus

they represent a unique Euler angle and removes the discontinuity.

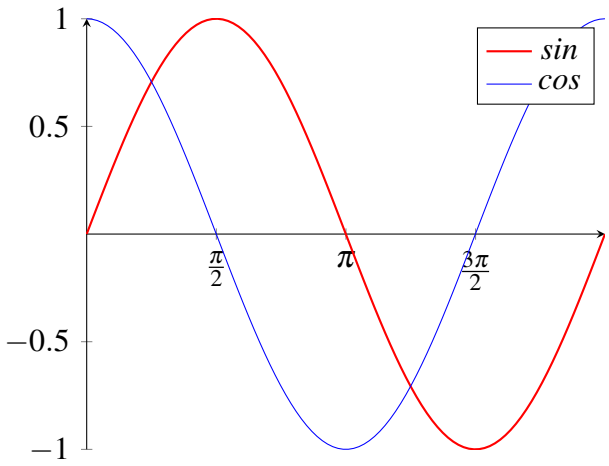


Figure 4.5: Plot of sinusoids, which represent the euler angle in continuous form.

2.

Setup	
State Space	$(x, y, sin\alpha, cos\alpha, e_1, e_2, e_3, e_4)$
Action Space	$(\text{action}_1, action_2)\|\|DecoupledAction - Space$
Reward Function	$-\sqrt{x^2+y^2} - \sum(A) - (A - prev_A)^2 + r_{terminal}$

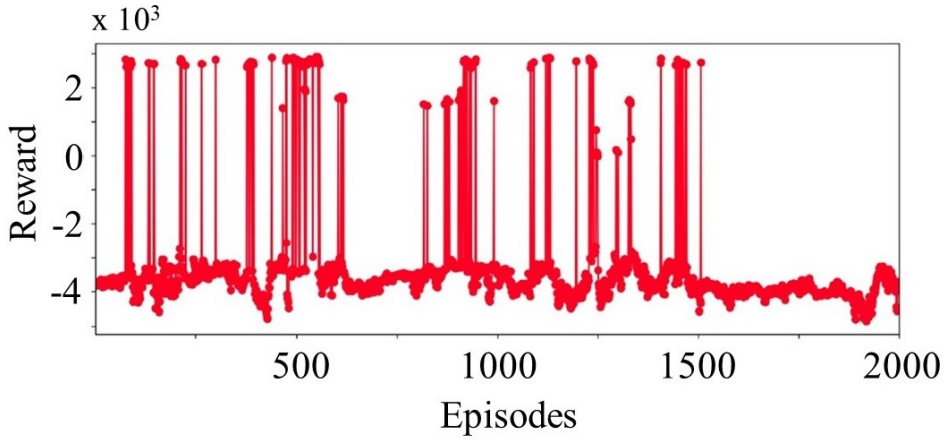


Figure 4.6: It is showing the overall Reward function trajectory computed for 2000 episodes. Here, the agent started learning after the 170th episode, but the policy is not stable due to the decoupled action space.

3.

Setup	
State Space	$(x, y, sin\theta, cos\theta, e_1, e_2, e_3, e_4)$
Action Space	(A_1, A_2)
Reward Function	$-\sqrt{x^2+y^2} + r_{terminal}$

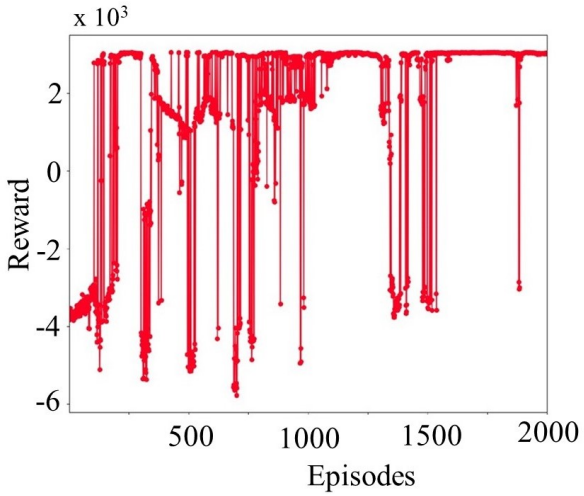


Figure 4.7: It is showing the overall Reward function trajectory computed for 2000 episodes. Here, the agent started learning after the 120th episode, and the policy got stable after 1550 episodes.

Following are the observation made from the above experiments: -

1. We found a discontinuity at 0 *and* 2π of the robot's orientation angle with respect to the goal. This discontinuity was solved by using sine and cosine of that angle in the observation-space.

2. We found that the policy using decoupled action is inferior to one using coupled action.

4.5 Approach-1

Considering the above observation, we came up with our first approach. We used the policy learned in the previous experiments and trained them in a static and dynamic environment.

4.5.1 Observation Space Definition

Now the complexity of the environment has been increased by introducing pedestrians. Moving objects imply time-dependent decisions; thus, we gave a stack of four observations at different time steps to the agent. The stacking of observation simulates an optical flow. We used the Deque stack and kept the last three observations in a stack. It removes the oldest observation when a new one is added.

Observation Space:- $[(x, y, sin(\alpha), cos(\alpha), E, X)_t,$

$(x, y, sin(\alpha), cos(\alpha), E, X)_{t-1},$

$(x, y, sin(\alpha), cos(\alpha), E, X)_{t-2},$

$(x, y, sin(\alpha), cos(\alpha), E, X)_{t-3}]$

where,

x, y is the relative distance of robot w.r.t goal.

E is the vector of previous action of the robot.

X is the 18-dimension vector of LIDAR ranges.

α is the orientation of robot w.r.t goal shown in Fig 3.1.

The robot does not have any encoders; thus, we use the robot's previous actions as the input to the observation space. The LIDAR is a 360°sensor that provides 360 data points. We used

only 18 data points from LIDAR to reduce the dimensionality of the observation space. These points are minimum range values between 20°interval throughout the 360°, shown in Fig 4.8.

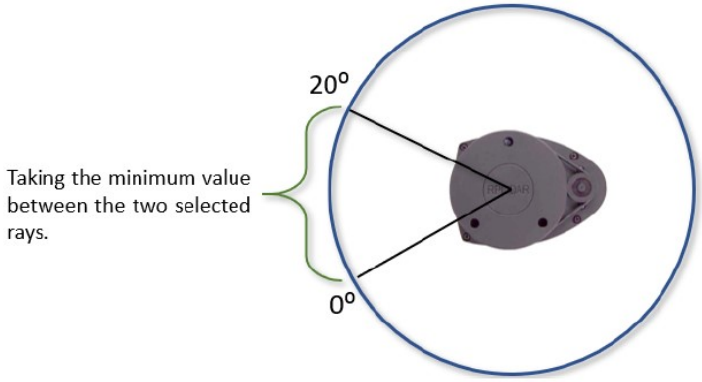


Figure 4.8: Showing selection of 18 points from 360 available data points

4.5.2 Action Space Definition

It is continuous action space which is the RPS values i.e. $[A_1, A_2]$ ranging from [-5, 5] which then provided to the robot as a combination of RPS of each wheel i.e. $a_1 = [A_1 + A_2], a_2 = [A_1 - A_2]$ shown in Fig 5.3. We found that the combination of action makes the learning faster, which is intuitive as only action A_1 is required to move forward or backward and to move left or right combination of A_1 and A_2 is required.

4.5.3 Reward Function Definition

There are two conditions for the reward function i.e., the reward for reaching and not reaching the goal.

$$r(s_t, a_t) = \begin{cases} -\alpha_1(d_t) + \alpha_2 r_c + r_{terminal} & if \quad d_t \leq th_d \\ -\alpha_1(d_t) + \alpha_2 r_c & if \quad d_t > th_d \end{cases} \tag{4.8}$$

where,

$$d_t = \sqrt{x^2 + y^2} \tag{4.9}$$

$$r_c \longleftarrow r_c - \frac{a}{1 + e^{b.X_{[i]}}} \tag{4.10}$$

The eq 4.9 is the L2 norm or distance penalty between the robot and the goal. The eq 4.10 is the collision penalty, which is similar to the sigmoid function. We found that it is tough to train the robot to navigate through the narrow passage when considering the equal collision penalty of all the sides. In reality, only the front and the back collisions are dangerous when compared to side collisions. Thus in order to regulate the collisions penalty constant a and b are changed. If a $\uparrow$ and b $\downarrow$, then collision value increases and vice-versa. We used $a > b$ for front and back values of lidar ,and $a < b$ for left and right side lidar values. If the d_t, i.e., the distance between the goal and robot is less then the threshold, then a terminal reward $r_{terminal}$ is added.

4.5.4 Limitation

- **Problems: -**

    1. The agent learns, but soon the policy shifts from the expert to the worst, shown in

Figure 4.9. The obtained policy is very stochastic.

2. The obtain policy is unstable in a dynamic environment and takes a long time to navigate.

3. Especially in a Dynamic Environment, the agent struggles to navigate for long-distance. The maximum it traveled is 3-4 meters.

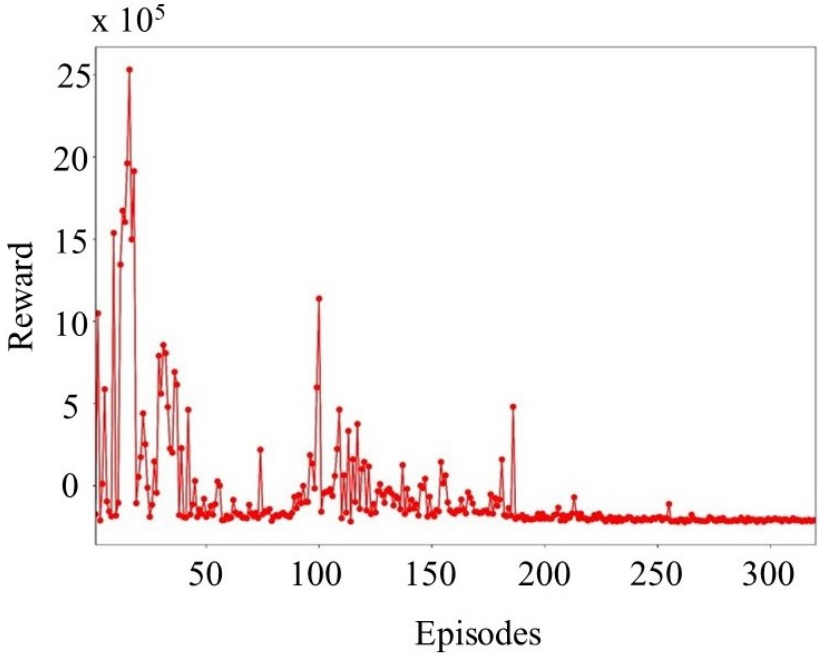


Figure 4.9: Limitation of the proposed approach

- **Solution: -**

1. There is a trade-off between the total reward and entropy. SAC tries to maximize both entropy and reward, but it leans towards the quantity with a higher value. This was solved by making a reward quantity higher in amount than entropy.

2. Still, there is a brittleness in SAC in a very uncertain environment. This was solved by making the action frequency lower.

3. We integrate a global planner which acts as an upper-level controller; it finds an

optimal path based on the available information. The optimal path is divided into via-points at an interval of 4 meters. Then the local planner takes the immediate via-point as its goal and navigates. This helps the agent to travel for long-distance.

4.6 Final Proposed Approach

By applying all the solutions to the approach-1 and integrating global planner, we got our final approach. For the policy, we used a dense neural network of three layers of size 512, 256, 128. We used Stable-Baselines implementation, which uses RELU as an activation function. The policy takes observation space consisting of relative distance with sine and cosine of the robot's orientation with respect to the goal, and a vector of range data from lidar as input and gives deterministic continuous action as output. The entropy coefficient is equivalent to the inverse of the reward scale in the original SAC paper and is set at auto to learn it automatically. Doing this avoids having too high errors when updating the Q functions.

We divided our controller into the upper and lower level controller. The upper-level controller is based on a global planner; for this, we used the A* algorithm. The A* algorithm works online, which means it does not need a complete saved SLAM map. Based on the initial OGM (Occupancy Grid Mapping) data from the Hector-SLAM algorithm, the A* finds a global path to the goal point. The unknown grids in OGM data were considered a free grid or obstacle-free. It gives a path consisting of via-points placed 4-meters apart. The lower level controller takes the first via-point as a virtual goal point. The lower level controller, which is the trained policy using the SAC algorithm, navigate the robot towards the given virtual goal point. When the distance between the virtual goal point and the robot is less than the set threshold, the A* takes the new OGM data and finds the new global path and then gives the first via-point as a virtual goal to the lower level controller until the final goal point is reached. The flow chart of the proposed approach is shown in Fig 4.10.

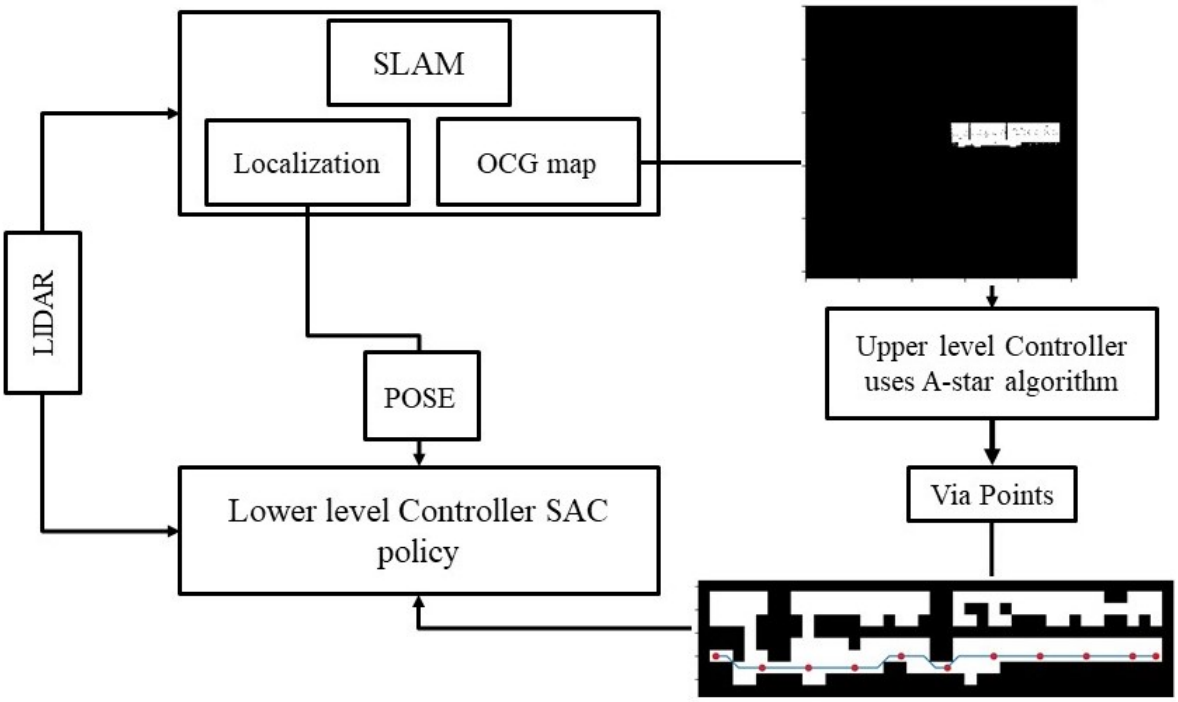


Figure 4.10: It is the flow chart of the proposed approach. It shows an Upper-level controller based on A-star and a Lower-level controller based on the trained policy on the SAC algorithm. The red dots are the via-points generated by the A* algorithm.

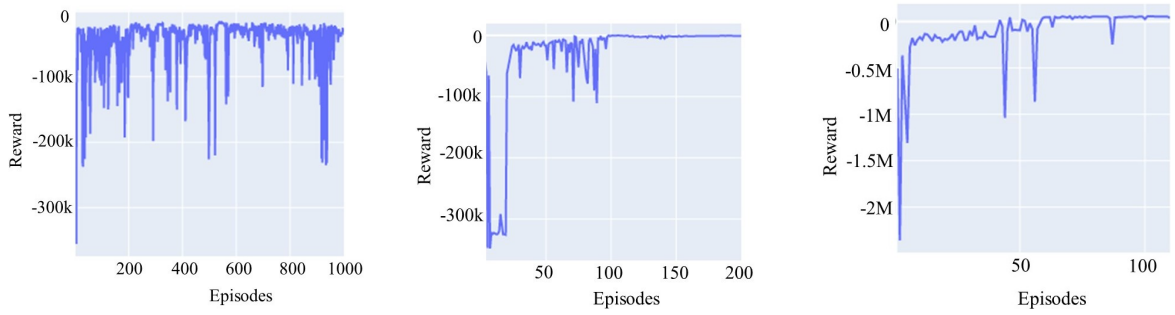
4.7 Experiment

The training of the policy is done in the simulation environment created in the V-REP. We trained our policy on two environments, as shown in Fig 4.2a. Env-1 is a static environment that consists of a wide variety of shapes as static obstacles. The robot's goal is the cylinder, which cannot be rendered by the laser sensor mounted on the robot. The static obstacles change their position in every episode randomly to get a generalized policy. It is taken care that the goal point and the location of the obstacles did not overlap with each other. Env-1 consist of two rooms connected by a narrow door. The environment also includes tables with thin legs, which makes it challenging for a low-cost LIDAR sensor. Env-2 is the dynamic

environment replicating a restaurant scenario with chairs, tables, poles, and plants with randomly walking five pedestrians, as shown in Fig 4.2b. Pedestrian did not follow any rule, thus making the environment very challenging.

The learning rate for all networks is set at 0.0001, and the hyper-parameters of the reward function were set trivially. The buffer size is 50000, batch-size is 128, and the gamma is 0.995. We trained the model from scratch on Env-1, i.e., for static obstacles and transferred the learning from the Env-1 to Env-2, i.e., environment with dynamic obstacles with an SGD optimizer on a single Nvidia GeForce GTX 1060 3GB with an Intel Core i5-8400 CPU @ 2.80GHz 6-core processor.

4.8 Result

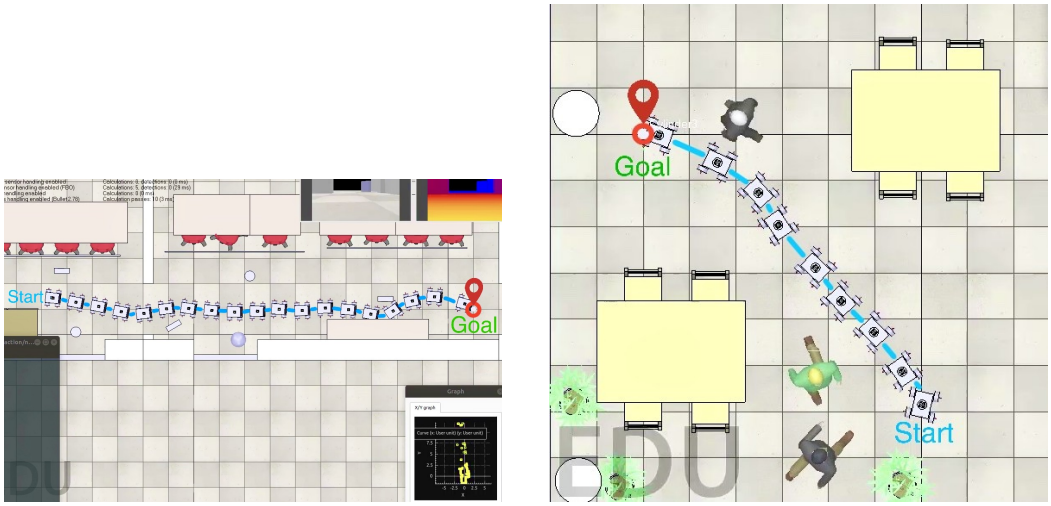


(a) (b) (c)

Figure 4.11: Figure4.11a shows the training curve for the static environment. The agent learned the policy in 1000 episodes. Also, learning is not stable. Figure 4.11b shows the training curve for the same static environment with lower action frequency. The effect of low action frequency improved the learning speed and made the policy more stable. The agent learned in 100 episodes. Figure 4.11c shows the training curve for the dynamic environment. The agent used the previous model trained on the static environment and learned the policy in 70 episodes.

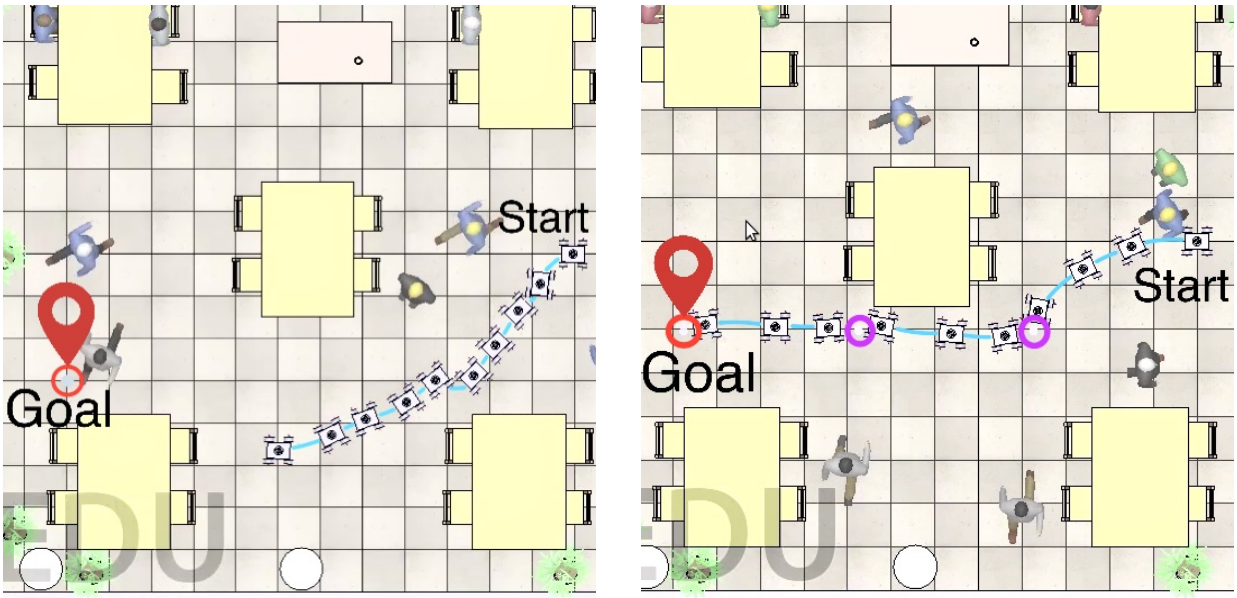
Figure 4.11 shows the training curve for a static and dynamic environment. Each episode has 800 time-step, with each time-step taking 0.2 seconds after lowering the action frequency, which helps stabilize the learning. It is found that with the default action frequency of around 20Hz makes the learning unstable and slower. We decrease the frequency to 5Hz, which makes a big difference in learning. This makes the policy stable and 10x times faster than that based on default frequency. Fig 4.11a shows the unstable learning curve under default action frequency, which is improved by lowering the frequency, which is shown in Fig 4.11b for the static environment. We tried to train the policy from scratch for a dynamic environment. However, the agent did not learn anything, so we transferred the previously learned model to the dynamic environment. We got a stable policy in just 70 episodes, which is shown in Fig 4.11c.

4.9 Evaluation



(a) (b)

Figure 4.12: Frame grabs from evaluation video showing the path followed by the agent to achieve the goal.



(a) (b)

Figure 4.13: It is showing the comparison of different controllers. Fig 4.13a shows the controller based on the RL policy, which fails to achieve the goal. Fig 4.13b shows our proposed controller, which is based on a global planner that is the upper-level controller and lower-level controller based on RL policy.

We evaluated the learned policy; the frame grab of the path traveled by the agent in a static environment is shown in Fig 4.12a. The agent traveled a distance of 9.5 meters in 72 seconds without colliding with the obstacles. Obstacles were in a different position in evaluation than in the training environment. Click on this *Static World Video*.

Figure 4.12b shows the frame grab of the path traveled by the agent in a dynamic world. The agent achieved the goal place at 4 meters in 20 seconds without colliding with pedestrians who are moving randomly. Click on this *Dynamic world video*.

Figure 4.13 shows the comparison of different controllers for the long-range. The controller, based on RL policy failed to achieve long-range goals in a dynamic world. It may be because

we are not using any LSTM or RNN policy, or policy got stuck in local minima.Click on this *Controller failed video*. This problem is solved by our proposed approach, which consists of a global planner as the upper-level controller, which gave via-point to the policy, which is the lower-level controller. The purple dots in fig 4.13b were the via-points. We found that our proposed controller for a similar goal position covers a distance of 8 meters in 36 seconds without any collision. Click on this *Long-range controller video.*

Chapter 5

Assembly of skid-steer drive wheeled robot

Skid-steer drive locomotion is widely used on tracked vehicles such as bulldozers and tanks. It is also used on many wheeled vehicles such as four and six wheels [6]. The relative velocities of the left and right side wheels are used to steer in a skid-steer drive vehicle. Thus, the slipping of wheels with respect to the ground is required for turning [7]. This drive is energy inefficient due to the slipping with the ground, and controlling is difficult when compared with the differential drive robot. Moreover, the tires wear out faster [8]. However, with all the above shortcomings of the skid-steer drive vehicles, it has higher maneuverability like a differential steering. Here more wheels make a robust structure than a differential drive vehicle, which makes enough room for circuit and sensors [8].

Seeing the advantage of maneuverability and robustness, we assembled a four-wheel skid-steer vehicle. Its Chassis is made up of ABS with a dimension of 266 mm x 230 mm. We used wheels of 110 mm in diameter. Figure 5.1 shows the dimension of the robot.

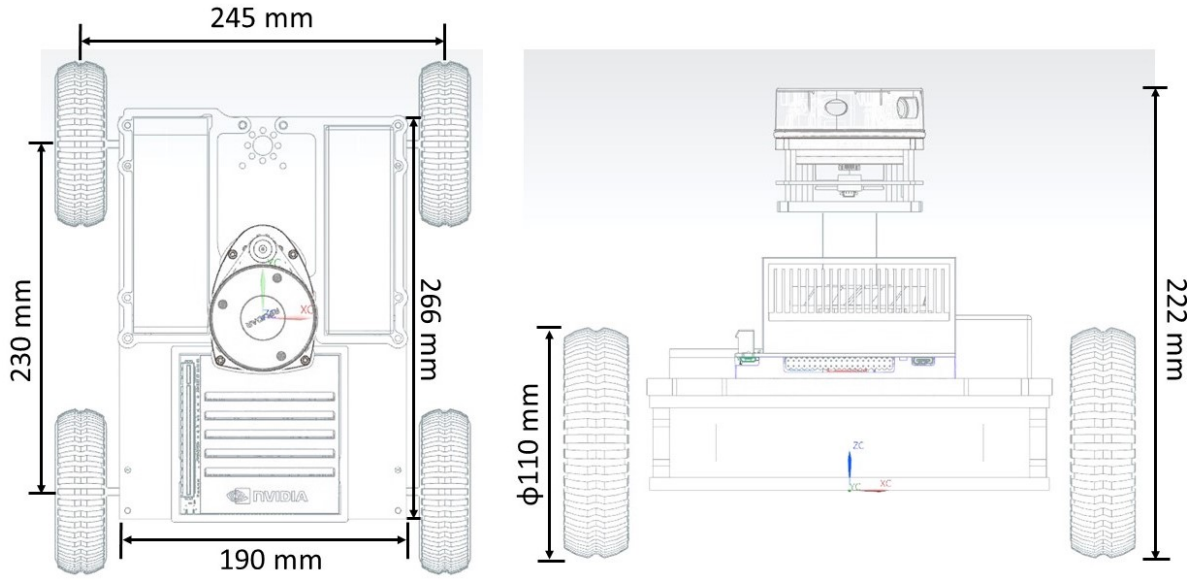


Figure 5.1: Dimension of the robot

5.1 Torque Calculation and Motor Selection

Assumptions:-

1. The maximum acceleration of the robot at maximum load is $0.5 m/s^2$.
2. We took a motor efficiency of 60%.
3. The calculated torque is 1/4th of the actual torque.

Radius of the wheel = 0.055 m

Number of drive wheels = 4

Maximum Load = 4 kg

Thus,

$$Calculated\ Torque = \frac{weight \times acceleration \times radius}{Number\ of\ drive\ wheel}$$

$$Calculated\ Torque = \frac{4 \times 0.5 \times 0.055}{4} = 0.0275 Nm$$

Taking Motor efficiency of 60%.

$$Calculated\ Torque\ =\ \frac{0.0275}{0.6}\ =\ 0.0458Nm$$

Since calculated torque is 1/4th of the actual torque.

$$Actual\ Torque\ =\ 0.0458 \times 4\ =\ 0.188Nm$$

Motor Selection:- Since the actual torque required is 0.188 Nm for a load of 4kg with a maximum acceleration of $0.5m/s^2$. We selected a dc motor with a stall torque of 3 kgf-cm, i.e., 0.294 Nm with an RPM of 624. We selected a motor with a higher rating to consider all the factors not taken in assumptions. This motor has an operating voltage of 3$\sim$12 V with its no-load current value of 0.19 A. The weight of the motor is 82 grams, which makes this motor lightweight considering the load capacity of the robot.

5.2 Selection of Development Board

As we aim to build a platform capable of doing online learning in the real world, we need a board that has a powerful CPU and GPU. The board also requires to be power-efficient as the objective is to build a mobile platform. Therefore we selected Nvidia AGX Xavier development board as it features an NVIDIA VOLTA 512 core GPU with 64 tensor cores and a 64-bit ARM-based eight-core CPU. In addition to this, it consists of a dedicated chip for accelerating deep learning programs coupled with 16GB RAM and 32GB eMMC flash memory. The boards have their custom Ubuntu OS which is capable of running ROS and almost all arm based python libraries. The board has three power modes to choose from, i.e., 10 watts, 15watts, and 30 watts. Thus based on the processing requirement, different modes can be selected, which makes it power efficient.

Power supply:- We selected a LiPo battery of capacity 5000mAh 14.8 V with a constant discharge of 20C. Thus the maximum discharge rate of the battery is: -

$$Discharge\ rate\ =\ \frac{5000 \times 20}{1000}\ =\ 100\ Ampere\ constant\ discharge\ rate$$

Thus, we can only draw a maximum of 100 A current from the battery. We have used a voltage regulator to eliminate the surge in power for the safety of the board. Also, we have stepped down the voltage to 12V. Considering Nvidia board to run at 30 watts with 12 V supply we need a current of 2.5 A.

Thus,

$$Theoretical\ running\ time\ =\ \frac{5000}{1000\times 2.5}\ =\ 2\ hours$$

5.3 Selection of LiDAR Sensor

(a) Nvidia AGX Xavier[1] (b) RPLiDAR A1 M8[2]

Source: [1]Nvidia autonomous machines [2]Slamtec Co., Ltd

Figure 5.2: (a) Nvidia AGX Xavier development board. (b) A 360°LiDAR sensor

LiDAR stands for light detection and ranging, it uses a pulse from a laser to collect the measurement from the environment to build a 3D or 2D map of the world. We used a low-cost RPliDAR A1 M8 model for the robot. It provides a 360°scan field with a frequency ranging

from 2∼10 Hz. The sensor's frequency is changed by changing the PWM of the motor, which rotates the laser sensor. It has a sample rate of up to 8000 times at 10 Hz. The detection range at 10 Hz is 8 meters, with an angular resolution of 1-degree and a minimum range of 0.2 cm. It requires a power supply of 5 V. Another reason to select this sensor is its ROS support, which makes it easy to use with SLAM algorithms. It is directly connected to the Nvidia board with a micro USB cable.

5.4 Miscellaneous Components

We used an L298n motor driver with Arduino to control the dc motor since it can control both the speed and the spinning direction of the motor. The speed is controlled by changing the PWM (Pulse Width Modulation), and H-bridge controls the direction. The PWM values are transferred from the Nvidia board to Arduino via i2c communication between Arduino and Nvidia board. The L298n motor driver requires either 5 V or 12 V for the operation. We used a 12 V power supply as our motor rating is 12 V. We used Arduino nano as it offers the same connectivity features as of Arduino UNO in a smaller factor. It has 8-bit PWM output at pins 3, 5, 6, 9, 11. For i2c communication, we use pin A4(SDA) and A5(SCA). A step-down buck converter is used to provide an input voltage of 5 V from a battery capacity of 11.1 V.

5.5 I2C Communication

Xavier's GPIO library supports PWM only on external hardware PWM controllers such as Arduino since the library does not implement Software emulated PWM, which makes the Xavier control the motor in two modes either on or off. Therefore, we used Arduino nano via i2c communication. The i2c protocol needs two lines to send and receive data. With the i2c interface, one can connect a single master to multiple slaves or multiple masters to a

single slave. It uses SDA (Serial Data) for data transmission and SCL (Serial Clock), which carries the clock signal. As the clock signal changes from low to high, a single bit of data is transferred. Thus, it is a serial communication as the data is transferred bit-by-bit. To remove the latency between the right and the left side wheels, we used a dedicated Arduino nano board. Thus, the Nvidia board transmits PWM values to the Arduino board.

5.6 Circuit Diagram

Figure 5.3 shows the schematics of the circuit diagram of the robot. There are two independent power supplies. The one with 11.1 V is connected to the dc motors, Arduino nano, and the other with 14.8 V is connected to the Nvidia Xavier development board. There is a voltage regulator between the Nvidia board and its power supply, which step-downs the voltage from 14.8 V to 12 V. The purpose of the regulator is to ensure constant power supply free from any power surge.

The 11.1 V battery is connected to dc motor via L298n motor driver. The L298n can control two motors. The left side motor driver is connected with 10, 11, 12 pins of the left side Arduino nano, and the right side moor driver is connected with 9, 6, 7 pins of the right side Arduino nano. The two buck converters convert 11.1 V supply to 5 V supply. These 5 V supplies are connected with the Ardunio nano boards. For i2c communication left and right side Ardunios are connected with the (1 x 8 address bus), i.e., pins [3, 5, 6] and (1 x 1 address bus), i.e., pins [25, 27, 28] of the Nvidia GPIO pins respectively.

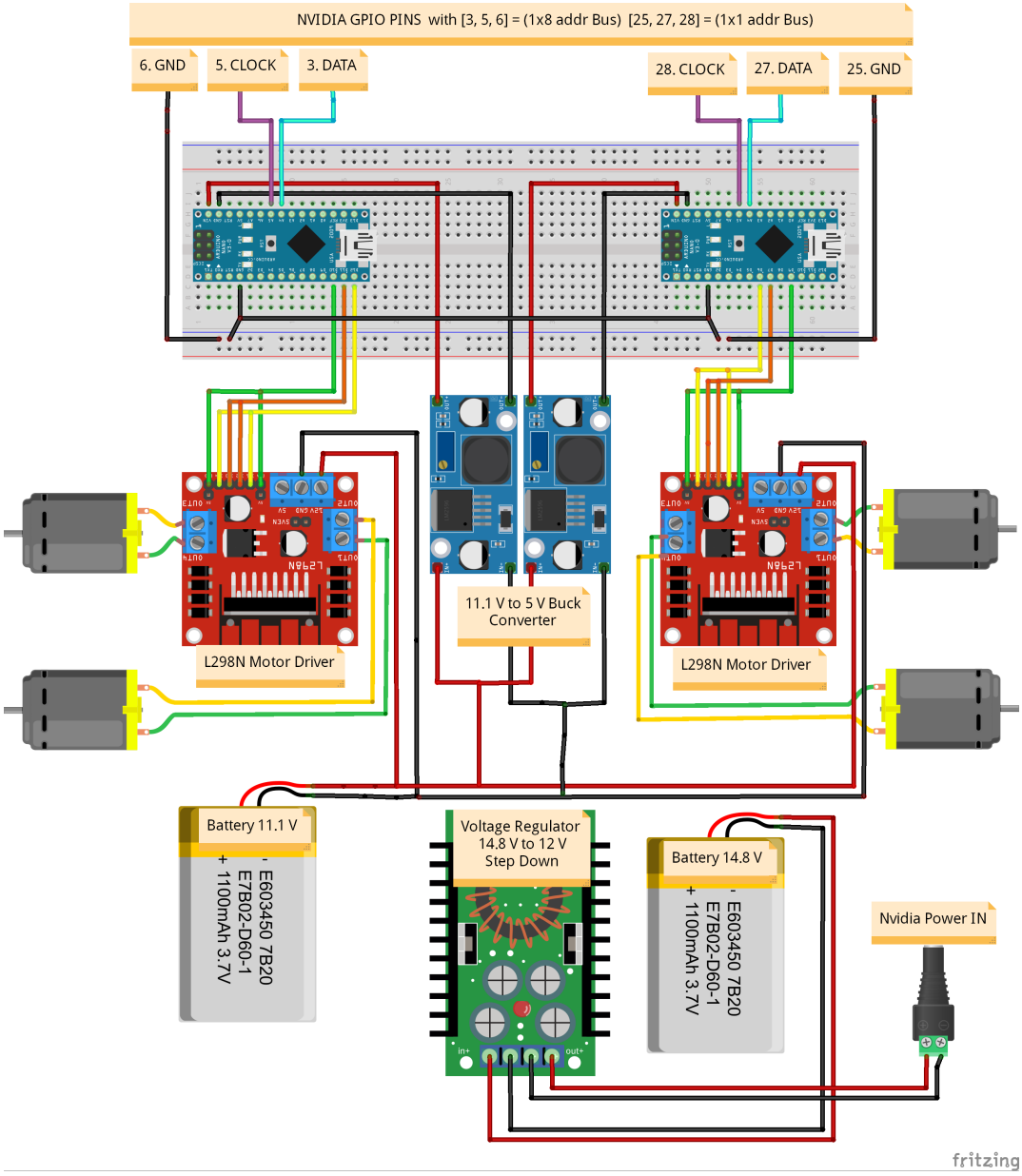


Figure 5.3: Circuit diagram of the robot

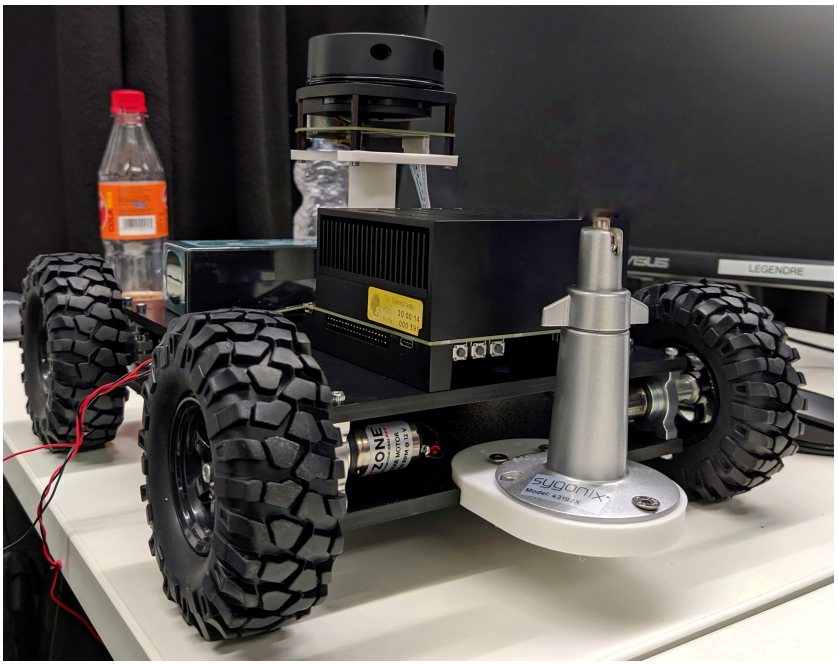


Figure 5.4: Snapshot of the Hardware

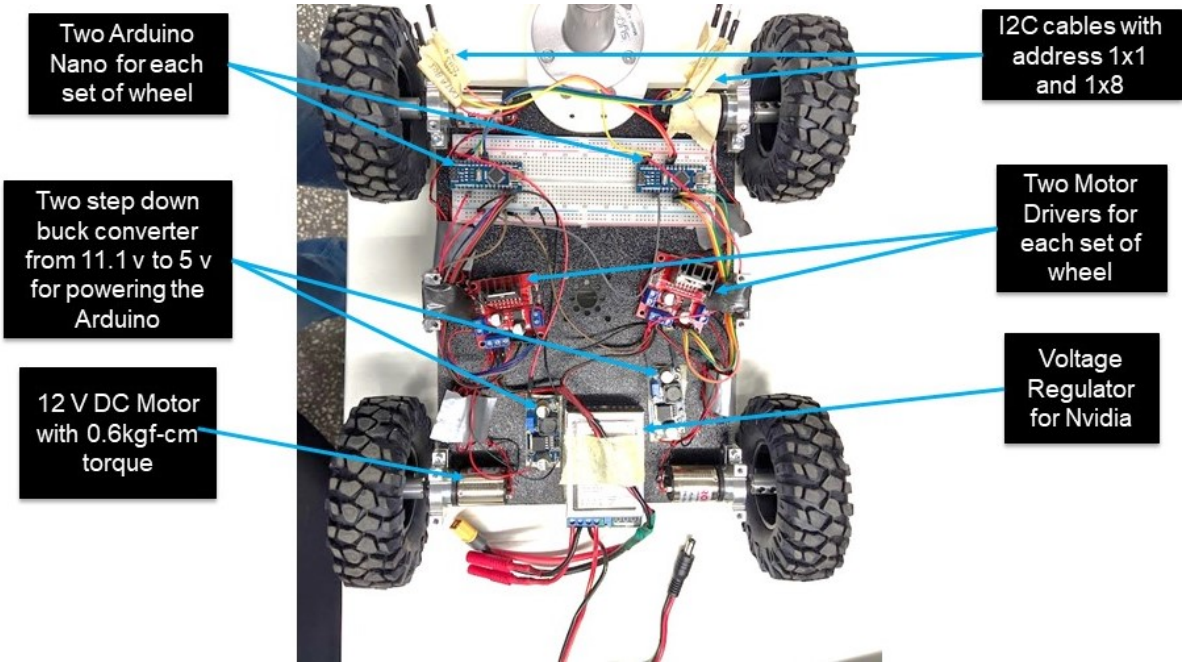


Figure 5.5: Circuit on the real hardware

5.7 Action-Space in Real Hardware

- The motor needs PWM signals to change the RPS of wheels. However, in the simulation, we can only control the RPS of the wheels. The model obtained in simulation gives action in the form of RPS as output that is useless in real hardware. Thus, a transfer function is needed to convert the PWM to RPS values in simulation. Therefore our agent can directly learn the PWM values in simulation, which in theory can be used directly in real hardware.
- Since we did not get any direct transfer function from the manufacturer, we did polynomial regression on the PWM values vs. the velocities obtained from the SLAM. Each experiment is run for 1 second with different PWM values. We received a transfer function for the robot. Figure 5.6 shows the polynomial regression curve obtain after running the experiments up to 85th PWM value.

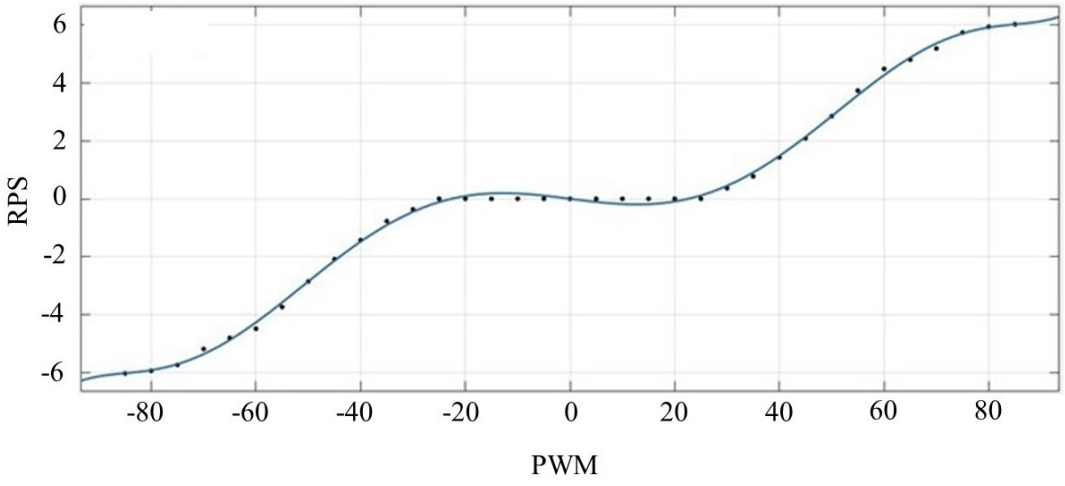


Figure 5.6: Regression curve between RPS and the PWM for DC Motor

We got the following Transfer function: -

$$\begin{aligned} y \;=\; & 0.3362x^7 - 1.781 \times 10^{-15}x^6 - 2.665x^5 + 5.669 \times 10^{-}15x^4 \\ & + 6.533x^3 - 3.77 \times 10^{-}15x^2 - 1.174x + 4.245 \times 10^{-}16 \end{aligned} \tag{5.1}$$

Since it a four-wheel-drive system, which means each wheel has its own power. Thus, we have not taken into account the effect during turning of the vehicle on the slippery floor as there will be a drop at the coefficient of road adhesion, which may cause drive wheels to slip [58]. There is no provision of traction control for reducing the torque on the wheel, which is slipping. We also found this issue during our limited testing on the robot. The cause was the uneven load distribution, which leads to the slipping of one wheel.

5.8 Planned Experiments

1. To implement the policy learned on the simulation in the environment with no obstacles. From this experiment, we are expecting a policy which is capable of navigating in the real world. As there are no obstacles, the agent's primary aim is to learn the dynamics of the vehicle while navigating towards the goal.

2. To implement the policy learned in the previous experiment in the environment with static obstacles. We are expecting the agent to learn the dynamics of the system in the last experiment. Then the primary aim of this experiment is to navigate towards the goal while avoiding the static obstacles.

3. In the final experiment, if the agent learns to navigate in the environment with static obstacles. Then the primary aim of this experiment is to implement the previous policy in the environment with the dynamic obstacles. The dynamic obstacles here will be the humans walking randomly in the test space.

Chapter 6

Conclusion

In this thesis, deep reinforcement learning is implemented for navigation of the skid steer wheeled robot. We have kept the traditional global planner but have replaced the local planner with the DRL policy. The objective of the global planner is to find an optimal path based on the available information. The optimal path is divided into via-points at an interval of 4 meters. Then the local planner takes the immediate via-point as its goal and navigates. We have developed a learning infrastructure for a local policy that uses the hector slam algorithm for localization, mapping, and A* algorithm as a global planner. The state of the art DRL algorithm soft-actor critic (SAC) is applied to train local policy. The trained policy outputs continuous control action in the form of PWM to the robot. The continuous control actions gives better control over the trajectory executed by the agent.

In the first stage, we have trained the agent in the simulation environment containing no obstacles. The aim was to find the best performing observation-space and action-space. The observation-space containing sine and cosine of the robot's orientation to the goal performs better than others. We found a unique way of providing action to the robot, which makes the agent learn faster.

In the second stage, we have trained the agent in the environment with static obstacles. We have formulated a function for the collision penalty. The agent has successfully learned to

navigate and achieved all the goals. To check the robustness of the policy, we have also placed obstacles in a different position. The agent successfully meets all the targets.

In the third stage, the complexity of the environment has been increased by introducing pedestrians. Moving objects imply time-dependent decisions; thus, we gave a stack of four observations at different time steps to the agent. We decreased the agent's action frequency to 5 Hz from 20 Hz, which makes the policy extremely stable compared to other policy we found earlier. The integration of the global planner eliminates the need to use LSTM or RNN policies, making the learning easier.

6.1 Future Work

Although the obtained results are promising, future improvements are required in this approach. The following future work are needed: -

- The above work is tested only in the simulated environment. Thus there is a need to transfer model from simulation to real-world.
- There is a slight delay when the global planner finds the via-points. Thus it is needed to make A* to compute new paths parallelly with the SAC policy. Therefore, the upper-level controller finds the new path when the robot is near the via-point, which is the short term goal. The upper-level controller, i.e., A*, will take that short term goal as the start point for the new path.

Bibliography

[1] I. F. Vis, "Survey of research in the design and control of automated guided vehicle systems," *European Journal of Operational Research*, vol. 170, no. 3, pp. 677–709, 2006.

[2] H. Fujiyoshi, T. Hirakawa, and T. Yamashita, "Deep learning-based image recognition for autonomous driving," *IATSS research*, vol. 43, no. 4, pp. 244–252, 2019.

[3] J. Cecılio, K. Duarte, P. Martins, and P. Furtado, "Robustpathfinder: Handling uncertainty in indoor positioning techniques," *Procedia computer science*, vol. 130, pp. 408–415, 2018.

[4] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: Part i," *IEEE robotics & automation magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[5] F. Rubio, F. Valero, and C. Llopis-Albert, "A review of mobile robots: Concepts, methods, theoretical framework, and applications," *International Journal of Advanced Robotic Systems*, vol. 16, no. 2, p. 1 729 881 419 839 596, 2019.

[6] J. Yi, H. Wang, J. Zhang, D. Song, S. Jayasuriya, and J. Liu, "Kinematic modeling and analysis of skid-steered mobile robots with applications to low-cost inertial-measurement-unit-based motion estimation," *IEEE transactions on robotics*, vol. 25, no. 5, pp. 1087–1097, 2009.

[7] A. Mandow, J. L. Martinez, J. Morales, J. L. Blanco, A. Garcia-Cerezo, and J. Gonzalez, "Experimental kinematics for wheeled skid-steer mobile robots," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2007, pp. 1222–1227.

[8] K. Kozłowski and D. Pazderski, "Modeling and control of a 4-wheel skid-steering mobile robot," *International journal of applied mathematics and computer science*, vol. 14, pp. 477–496, 2004.

[9] B. Patle, A. Pandey, D. Parhi, A. Jagadeesh, *et al.*, "A review: On path planning strategies for navigation of mobile robot," *Defence Technology*, vol. 15, no. 4, pp. 582–606, 2019.

[10] F. Gul, W. Rahiman, and S. S. Nazli Alhady, "A comprehensive study for robot navigation techniques," *Cogent Engineering*, vol. 6, no. 1, p. 1 632 046, 2019.

[11] J. Zeng, R. Ju, L. Qin, Y. Hu, Q. Yin, and C. Hu, "Navigation in unknown dynamic environments based on deep reinforcement learning," *Sensors*, vol. 19, no. 18, p. 3837, 2019.

[12] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," 2011.

[13] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

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

[16] W. Masson, P. Ranchod, and G. Konidaris, "Reinforcement learning with parameterized actions," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[17] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012.

[18] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, 2015, pp. 1889–1897.

[19] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[20] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[21] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[22] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *arXiv preprint arXiv:1801.01290*, 2018.

[23] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[24] S. Russell and P. Norvig, "Artificial intelligence: A modern approach," 2002.

[25] A. Faust, K. Oslund, O. Ramirez, A. Francis, L. Tapia, M. Fiser, and J. Davidson, "Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 5113–5120.

[26] S. James, M. Freese, and A. J. Davison, *Pyrep: Bringing v-rep to deep robot learning*, 2019. arXiv: 1906.11176 [cs.RO].

[27] D. Ferguson, M. Likhachev, and A. Stentz, "A guide to heuristic-based path planning," in *Proceedings of the international workshop on planning under uncertainty for autonomous systems, international conference on automated planning and scheduling (ICAPS)*, 2005, pp. 9–18.

[28] R. Arroyo, P. F. Alcantarilla, L. M. Bergasa, and E. Romera, "Towards life-long visual localization using an efficient matching of binary sequences from images," in *2015 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2015, pp. 6328–6335.

[29] W. Maddern, A. D. Stewart, and P. Newman, "Laps-ii: 6-dof day and night visual localisation with prior 3d structure for autonomous road vehicles," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, IEEE, 2014, pp. 330–337.

[30] C. McManus, W. Churchill, W. Maddern, A. D. Stewart, and P. Newman, "Shady dealings: Robust, long-term visual localisation using illumination invariance," in *2014 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2014, pp. 901–906.

[31] J. Le Cras, J. Paxman, and B. Saracik, "Improving robustness of vision based localization under dynamic illumination," in *Recent Advances in Robotics and Automation*, Springer, 2013, pp. 155–170.

[32] S. Thrun, "Probabilistic robotics," *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.

[33] N. D. Carlevaris-Bianco, "Long-term simultaneous localization and mapping in dynamic environments," MICHIGAN UNIV ANN ARBOR DEPT OF ELECTRICAL ENGINEERING and COMPUTER SCIENCE, Tech. Rep., 2015.

[34] H. Zhao, M. Chiba, R. Shibasaki, X. Shao, J. Cui, and H. Zha, "Slam in a dynamic large outdoor environment using a laser scanner," in *2008 IEEE International Conference on Robotics and Automation*, IEEE, 2008, pp. 1455–1462.

[35] U. Muller, J. Ben, E. Cosatto, B. Flepp, and Y. L. Cun, "Off-road obstacle avoidance through end-to-end learning," in *Advances in neural information processing systems*, 2006, pp. 739–746.

[36] L. Tai, S. Li, and M. Liu, "A deep-network solution towards model-less obstacle avoidance," in *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, IEEE, 2016, pp. 2759–2764.

[37] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2722–2730.

[38] C. Yu, Z. Liu, X.-J. Liu, F. Xie, Y. Yang, Q. Wei, and Q. Fei, "Ds-slam: A semantic visual slam towards dynamic environments," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 1168–1174.

[39] M. S. Bahraini, A. B. Rad, and M. Bozorg, "Slam in dynamic environments: A deep learning approach for moving object tracking using ml-ransac algorithm," *Sensors*, vol. 19, no. 17, p. 3699, 2019.

[40] K. Kawaguchi, L. P. Kaelbling, and Y. Bengio, "Generalization in deep learning," *arXiv preprint arXiv:1710.05468*, 2017.

[41] 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.

[42] L. Qiang, D. Nanxun, L. Huican, and W. Heng, "A model-free mapless navigation method for mobile robot using reinforcement learning," in *2018 Chinese Control And Decision Conference (CCDC)*, IEEE, 2018, pp. 3410–3415.

[43] Y. Liu, H. Liu, and B. Wang, "Autonomous exploration for mobile robot using q-learning," in *2017 2nd International Conference on Advanced Robotics and Mechatronics (ICARM)*, IEEE, 2017, pp. 614–619.

[44] L. Tai and M. Liu, "Towards cognitive exploration through deep reinforcement learning for mobile robots," *arXiv preprint arXiv:1610.01733*, 2016.

[45] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard, "Deep reinforcement learning with successor features for navigation across similar environments," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017, pp. 2371–2378.

[46] S. Feng, H. Ren, X. Wang, and P. Ben-Tzvi, "Mobile robot obstacle avoidance based on deep reinforcement learning," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, American Society of Mechanical Engineers, vol. 59230, 2019, V05AT07A048.

[47] X. Ruan, D. Ren, X. Zhu, and J. Huang, "Mobile robot navigation based on deep reinforcement learning," in *2019 Chinese control and decision conference (CCDC)*, IEEE, 2019, pp. 6174–6178.

[48] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine, "Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 1–8.

[49] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, *et al.*, "Learning to navigate in complex environments," *arXiv preprint arXiv:1611.03673*, 2016.

[50] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *2017*

IEEE international conference on robotics and automation (ICRA), IEEE, 2017, pp. 3357–3364.

[51] M. Dobrevski and D. Skocaj, "Map-less goal-driven navigation based on reinforcement learning," in *23rd Computer Vision Winter Workshop*, 2018.

[52] Y. Sun, J. Cheng, G. Zhang, and H. Xu, "Mapless motion planning system for an autonomous underwater vehicle using policy gradient-based deep reinforcement learning," *Journal of Intelligent & Robotic Systems*, vol. 96, no. 3-4, pp. 591–601, 2019.

[53] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017, pp. 1343–1350.

[54] M. Everett, Y. F. Chen, and J. P. How, "Motion planning among dynamic, decision-making agents with deep reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 3052–3059.

[55] Z. TP *et al.*, "Learning continuous control through proximal policy optimization for mobile robot navigation," 2018.

[56] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017, pp. 31–36.

[57] E. Rohmer, S. P. Singh, and M. Freese, "V-rep: A versatile and scalable robot simulation framework," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2013, pp. 1321–1326.

[58] G. Shuang, N. C. Cheung, K. W. E. Cheng, D. Lei, and L. Xiaozhong, "Skid steering in 4-wheel-drive electric vehicle," in *2007 7th International Conference on Power Electronics and Drive Systems*, 2007, pp. 1548–1553.