Eötvös Loránd University

Faculty of Informatics

Erasmus Mundus Joint Master in

Intelligent Field Robotic Systems

Deep Reinforcement Learning for Autonomous Navigation

Author:

Preeti Verma

Bachelor's in Electrical Engineering

Internal supervisor:

Dr. Balázs Nagy

Assistant professor, ELTE

External supervisor:

Dr. Palomeras Rovira, Narcís

Research Director, UdG

Budapest, 2024

Contents

Acknowledgements

I want to express my sincere thanks to my mentors, Professor Narcís Palomeras and Professor Balázs Nagy, for their invaluable guidance and support during the development of this thesis. I am particularly grateful to Professor Narcis for his help in debugging code and brainstorming new ideas. His dedication and expertise were crucial to the success of this work, and he has been an exemplary supervisor.

I also extend my gratitude to my friend Joseph, who provided me with constant company in the lab during the experimental phase. His support was essential in navigating the complexities of this research. Additionally, I am thankful to my friend Eric for his encouragement and motivation throughout the thesis process.

I would also like to acknowledge the University of Girona for providing me with the resources and facilities. Access to labs and equipment was essential for conducting my experiments and finishing this research project.

A special mention goes out to Imma from the IFRoS Master Program Coordination Team for her assistance in securing a spot in the lab and providing the required equipment for my work. Her help was instrumental in facilitating my research activities.

Furthermore, I am grateful for the support and feedback from my peers and classmates. Their insightful discussions and collaborative approach were beneficial. It Played a significant role in advancing this thesis project.

Lastly, I express my gratitude for the moral backing I received from my family. Their support and faith in my capabilities were instrumental in keeping me inspired during this endeavor. Without the support of these individuals and resources, this thesis would not have been possible. Thank you for your motivation, direction, and assistance.

Chapter 1

Introduction

This chapter lays the groundwork for the thesis by offering crucial background context and establishing its core theme. It presents an overview of the broader field to which the thesis pertains before deliberately focusing on specific themes. Research questions and objectives are identified and subsequently addressed within these themes in the thesis. The crucial contributions of the thesis are enlisted, and the chapter concludes by outlining its scope.

1.1 Background and context

Autonomous navigation is an innovative technology that has the capacity to completely disrupt several industries, including healthcare [[1], [2]], military operations industry [[3], [4]], space exploration [[5], [6]], and agriculture [[7], [8]]. Autonomous systems rely on sensor data to perceive their surroundings, analyze this information to make judgments, and carry out movements to accomplish predetermined objectives without the need for human involvement. This procedure necessitates ongoing examination of incoming data to enhance the system's understanding of the surroundings, adapt its trajectory to avoid obstructions, optimize time travel, and ensure safety.

The notion of autonomous navigation has seen substantial development from the initial stages of robotics. The initial advancements, such as Shakey [9], the inaugural robot with the ability to think through its actions, and CART [10], a robot that could follow lines and operate from a central computer via a wireless connection, established the groundwork for the advanced automated systems presently. These

fundamental developments have driven the development of highly competent autonomous systems that can function in complex and dynamic contexts.

The potential uses of autonomous navigation are extensive and significant. Autonomous robots in the healthcare industry can efficiently deliver supplies and drugs, alleviating medical personnel's burden [11]. This feature enables healthcare providers to prioritize patient care, improving overall healthcare efficiency. Autonomous drones and ground vehicles can carry out reconnaissance and supply missions in dangerous settings during military operations, therefore reducing the potential dangers faced by human soldiers [12]. This not only improves the effectiveness of operations but also greatly promotes the safety of military personnel.

Robots with autonomous navigation skills can autonomously transfer materials and goods in industrial environments, therefore improving production and operational efficiency [[13], [14]]. These robots possess the ability to maneuver through intricate production configurations, circumventing obstructions and guaranteeing punctual transportation of products. Autonomous rovers and probes in space exploration can travel alien terrains, gather data, and perform experiments without the need for direct human control [[15], [16]]. Operating independently is vital for operations conducted in remote and inhospitable areas where immediate human assistance is not feasible.

Autonomous robots in agriculture can help with precise activities such as planting, monitoring crop health, and harvesting, resulting in higher production and sustainability [17]. Moreover, these robots can enhance farming methods by gathering and evaluating data, increasing farming efficiency, and reducing the need for manual labor.

Due to the growing need for robust and reliable performance in these diverse applications, there is a notable emphasis on developing techniques to traverse environments where Global Positioning System (GPS) signals are not accessible or dependable [[18], [19]]. This challenge underscores the need for innovative approaches in autonomous navigation, as reliable navigation is essential for successfully deploying autonomous systems in real-world scenarios. Research and development efforts are continually advancing the capabilities of autonomous navigation systems, making them more adaptable and resilient in varied and challenging environments.

Conventional navigation techniques, A-star algorithm [20], Dijkstra's algorithm [21], Rapidly-exploring Random Tree (RRT) [22], Model Predictive Control (MPC)

[23], Probabilistic Roadmaps (PRM) [24], Visibility Graphs [25],for autonomous systems frequently depend on precise maps and GPS to determine position and plan routes. GPS utilizes satellite signals to precisely detect a receiver's location on or in close proximity to the Earth's surface, resulting in absolute positioning. Nevertheless, these techniques possess notable constraints. For instance, GPS transmissions may be inaccurate in metropolitan regions due to signal interference caused by skyscrapers or situations where satellite signals are inaccessible. Additionally, atmospheric conditions can compromise the accuracy of GPS and signal multipath effects, even in open locations.

Thus, dependence on pre-existing maps and GPS data poses challenges in dynamic and unstructured landscapes. Conventional path-planning approaches usually assume a stationary environment and adhere to cautious safety standards, which might result in inferior performance when confronted with dynamic surroundings. These approaches lack the capacity to adjust to unforeseen barriers or changes in the environment, which reduces their effectiveness in real-world situations where flexibility and adaptability are essential. These limitations led to the development of more advanced and robust navigation methods, like Deep Reinforcement Learning (DRL), that often integrate sensor fusion, machine learning, and real-time data processing, ensuring precise navigation in complex and dynamic environments [26].

DRL merges the perceptual abilities of deep learning with the decision-making aptitude of reinforcement learning [27]. Within the DRL field, an intelligent agent acquires the ability to make optimum judgments by actively engaging with its surrounding environment. The deep learning component employs neural networks to analyze complex sensory inputs, such as pictures or lidar data, and identify significant characteristics. Conversely, reinforcement learning allows the agent to acquire optimum strategies by obtaining rewards or punishments based on its actions, directing it towards behaviors that maximize the total rewards over time [28].

DRL has exhibited exceptional achievement in various intricate tasks, highlighting its capacity for autonomous navigation. An exemplary instance is the utilization of DRL in game playing, namely in the game of Go, where Google's AlphaGo triumphed against human world champions by exploiting sophisticated DRL methodologies [29]. Another remarkable achievement is in robotic control, where DRL has empowered robots to do complex tasks such as precise manipulation and agile movement [30]. These achievements have substantial ramifications for autonomous navi-

gation. The capacity of DRL to proficiently conquer complex surroundings and tasks indicates its potential to fundamentally transform the way autonomous systems traverse and engage with their surroundings.

One of the key challenges in applying DRL for autonomous navigation lies in the development of a suitable reward function [31]. The reward function should precisely capture the objectives and limitations of the navigation task, including obstacle avoidance, limiting journey duration, and energy conservation. A poorly constructed reward function might result in inefficient or hazardous actions. Hence, it is imperative to thoroughly analyze and continuously adjust the reward function to ensure it aligns with the intended results.

Another challenge is effectively handling the computing expenses linked to the processing of sensory information, such as lidar, cameras, and radar [32]. Consequently, DRL techniques need substantial computer resources to efficiently handle the data with many dimensions in real-time, which might pose a constraint in practical use cases. Efficient algorithms and hardware acceleration, such as GPUs or specialized AI processors, are crucial for managing the demanding calculations needed for DRL-based navigation systems.

1.2 Problem statement

Autonomous navigation in environments where GPS is unavailable or unreliable presents significant challenges due to limited environmental knowledge. Conventional autonomous navigation methods often struggle to address these complexities effectively, leading to inefficient and unreliable navigation outcomes. This thesis explores the application of DRL algorithms in unstructured environments, focusing on their ability to navigate efficiently without precise environmental or positional information. The research involves:

- Developing a custom Gymnasium environment for testing,
- Implementing a lidar data downsampling approach to reduce computational costs,
- Designing reward functions to guide the learning process of the DRL algorithms,

- Comparing the performance of different DRL algorithms, specifically Soft Actor-Critic (SAC) [33], Proximal Policy Optimization (PPO) [34], and Dynamic Window Approach (DWA) [35].

1.3 Research questions

The main objective of this research is to investigate the following research inquiries in order to comprehend how DRL can be leveraged for efficient autonomous navigation:

1. How can Deep Reinforcement Learning be effectively applied to achieve autonomous navigation in scenarios with limited environmental knowledge?

2. What is the impact of different reward functions on the learning performance of DRL algorithms in autonomous navigation tasks?

3. How do SAC, PPO, and DWA compare in terms of their ability to navigate environments of varying complexity?

1.4 Objectives and scope

This section provides an overview of the research, outlining the objectives to accomplish and the limitations under which the research is carried out.

- Creating a custom Gymnasium Setting: Creating a customized gymnasium environment tailored for testing DRL algorithms in navigation scenarios. This setting will replicate real-world simplistic environments, offering a controlled environment to assess algorithm performance.

- Implementation of Lidar Data Downsampling Technique: Applying a method to downsample lidar data with the aim of reducing expenses associated with processing high dimensional sensory information. This approach will enhance navigation system efficiency by decreasing data processing requirements without compromising environmental perception accuracy.

- Development of Effective Reward Structures: Enhancing reward structures that effectively steer DRL algorithm learning processes. The objective is to

optimize these structures to enhance learning outcomes, allowing autonomous systems to develop navigation strategies that minimize collisions, travel duration, and energy usage.

- Comparative Evaluation of DRL Algorithms: The research aims to conduct a comprehensive comparative analysis of SAC, PPO, and DWA algorithms. These methods will be tested across simulated environments from basic to complex surroundings to uncover their advantages and drawbacks and pinpoint the circumstances where they excel the most.

- Gain Practical Insights: To gain insights into the practical challenges of implementing and tuning DRL algorithms. This includes addressing issues related to policy modifications and hyperparameter tuning and documenting the challenges and limitations encountered during the research process.

1.5 Contribution of the thesis

This thesis aims to make the following significant contributions to the field of autonomous navigation:

- Development of a custom Gymnasium environment specifically designed for testing DRL algorithms in autonomous navigation scenarios.

- Implemented a novel lidar data downsampling approach to reduce computational expenses while preserving navigation efficiency, inspired by existing methodologies mentioned in research article [36].

- Comparative analysis of SAC, PPO, and DWA algorithms provides valuable insights into their effectiveness in navigating varying complex environments.

- Documentation of the challenges and limitations faced during policy modification and hyperparameter tuning of DRL algorithms, offering guidance for future research.

1.6 Overview of the thesis structure

This thesis is organized into five main chapters, each focusing on different aspects of the research on Deep Reinforcement Learning (DRL) for autonomous navigation.

- **Chapter 2: Literature Review**
  - **Literature Survey**: Reviews existing research on autonomous navigation, focusing on map-based, map-less, and learning-based approaches.
  - **Gaps in the Literature**: Identifies the gaps in current research.
- **Chapter 3: Methodology**
  - **Customized Gymnasium Environment**: Describes the development of the custom environment, including action and observation space, reward structure, workflow, and TurtleBot integration.
  - **Lidar Data Preprocessing**: Details the Lidar sensor simulation and data downsampling techniques.
  - **Deep Reinforcement Learning Algorithms**: Explains the DRL algorithms used, specifically PPO and SAC, their mathematical foundations, and implementation details.
- **Chapter 4: Results and Discussion**
  - **Experimental Setup**: Describes the experimental setup and scenarios tested.
  - **Results and Analysis**: Presents the results for different scenarios, including no obstacle, range sensor with obstacles, and Lidar data downsampling.
  - **Summary of Key Findings**: Summarizes the main findings from the experiments.
- **Chapter 5: Conclusion**
  - **Summary of Research Questions**: Recaps the research questions and how they were addressed.
  - **Conclusions Drawn from the Research**: Synthesizes the study's conclusions and discusses the research's contributions to the field of DRL and autonomous navigation.
  - **Recommendations for Further Research**: Suggests areas for future research to build on the findings of this thesis.

Chapter 2

Literature Review

In this chapter, a thorough review of relevant literature from fields closely aligned with the research conducted in this thesis is provided. In particular, deep reinforcement learning for autonomous navigation of vehicles is studied. This chapter systematically summarizes existing works, identifies limitations, and highlights the research gaps.

2.1 Literature Survey

The primary focus of this work is the analysis of Deep Reinforcement Learning (DRL) techniques for vehicle self-navigation in locations without GPS. The objective is to investigate how well these algorithms perform in comparison to more conventional methods so that autonomous vehicles can navigate uncharted territory while using the least amount of resources. A comprehensive overview of the state of the art research is given in the following sections, highlighting recent advancements and shortcomings in the methods that have been employed up to this point.

2.1.1 Map-based Navigation

Traditional algorithms

In 1959, E.W. Dijkstra proposed Dijkstra's algorithm [37], which uses a greedy approach to find the shortest path between two vertices on a graph in static situations. However, the algorithm can become slow for massive graphs. As a part of the Shakey project, N. Nilsson improved on Dijkstra's by incorporating a heuristic and named it the A* method [38]. It prioritizes nodes closest to the goal, cuts

down on search times, and adds a heuristic to help direct the search, improving system efficiency. Although more sophisticated, the D* algorithm [39], introduced by A. Stentz in 1994, extends A* for dynamic environments with reverse incremental search, changing pathways when barriers appear. The LPA* algorithm [40], by S. Koenig and M. Likhachev in 2001, effectively responds to dynamic changes by combining A* with incremental updates; however, it uses more memory. Further, the same authors introduced the D* Lite approach [41], which balances simplicity and efficiency for real-time applications by streamlining D* using backward search and incremental updates. Table 2.1 enlist the comparison between the conventional algorithm based on several parameters.

Table 2.1: Comparison of Traditional Path-planning Algorithms

Algorithm	**Description**	**Mechanism**	**Search Direction**	**Applications**	**Strengths**	**Limitations**
Dijkstra's	Shortest path in static environments	Explores all nodes, selects the nearest unvisited node	Forward	Static maps, network routing	Guarantees shortest path, simple implementation	Slow for large graphs
A*	Heuristic path planning	Uses cost $g(n)$ and heuristic $h(n)$ to prioritize nodes	Forward	Games, robotics, geographic navigation	Faster than Dijkstra, flexible heuristic	Efficiency depends on heuristic
D*	Dynamic path planning	Reverse search, updates paths dynamically as obstacles change	Reverse	Dynamic maps, robotics	Efficient for dynamic environments, reduces replanning	Complex, higher computational cost
LPA*	Incremental heuristic search	Combines A* with incremental updates	Forward	Dynamic environments, frequent replanning	Reuses previous search results, efficient updates	More complex, higher memory usage
D* Lite	Simplified incremental search	Combines A* with backward search, incremental updates	Reverse	Real-time dynamic navigation	Efficient, suitable for real-time applications	Complex, requires careful tuning

* Search direction stands for the trajectory that the search method follows to get from its the current position (real-time search location) to its destination (target location).

Probabilistic Algorithms

Probabilistic methods are crucial in navigating autonomous robots through real-world environments by tackling uncertainties effectively. These methods rely on tools to make intelligent choices based on incomplete or noisy sensor information. These algorithms leverage statistical methods to make informed decisions based on incomplete or noisy sensor data. One popular strategy is Particle Filters, which represent the robot's understanding of its location using samples called particles. These particles are continuously adjusted as new sensor data comes in, helping the robot accurately gauge its position and path. Markov Decision Processes (MDPs) are another probabilistic tool that offers a structured way to model decision-making in environments with unpredictable changes [42]. MDPs help robots plan their actions wisely by considering gains and future uncertainties. Such probabilistic techniques

are crucial for tasks like localization, mapping, and route planning, boosting the resilience and dependability of navigation systems in dynamic and uncertain conditions.

The study [43] introduces the Adaptive Unscented Particle Filter (AUPF) method, which enhances navigation precision in complex urban settings. This design incorporates an Inertial Navigation Systems (INSs) based processing stage and a Redundant Measurement Noise Covariance Estimation (RMNCE) stage to address instability resulting from multipath effects and improve environmental prediction [44]. The use of the two-stage adaptive method results in a reduction of 10% in horizontal position inaccuracy beyond the percentile when compared to conventional procedures. This work represents a breakthrough in the integration of INS, offering dependable and accurate location even under challenging circumstances.

In 2021, G. Raja et al. introduced the Particle Filter-based Indoor Navigation (PFIN) framework as a complete solution to the issues of indoor UAV navigation [45]. This framework greatly improves the capabilities of drones in terms of mapping, localization, path planning, and collision avoidance. The main contributions consist of the Quadcopter Mapping Algorithm (QMA), which enhances mapping accuracy by minimizing errors in state prediction through particle re-sampling, and the Optimized Localization Algorithm (OLA), which improves localization precision by utilizing particle weights and counts to reduce noise. The Adaptive Velocity Procedure (AVP) is a system that continuously modifies the speed of a drone to prevent collisions. It optimizes the braking distance to minimize the delay in attaining the desired objectives. The PFIN framework, which has been confirmed using Software-In-The-Loop (SITL) and Hardware-In-The-Loop (HITL) simulations, [46], exhibits a 14% decrease in position error when compared to the traditional Extended Kalman Filter (EKF) model [47].

K. Berntorp et al., in article [48], proposed a statistical framework that integrates real-time decision-making with route planning for autonomous cars using particle filtering (PF). This approach considers the task as an estimated problem, where probabilities are assigned to various paths depending on their adherence to specified driving rules, such as staying on the road and avoiding obstacles. PF technique guarantees the generation of computationally efficient trajectories. The efficacy of this technique has been demonstrated through simulations and experiments using a miniature robot. It has proven to be effective in challenging conditions, such as

avoiding crashes and navigating through traffic congestion, thereby highlighting its practicality and resilience in real-time settings. This work represents a breakthrough in autonomous vehicle navigation by offering a dependable method for generating secure driving courses that mimic human behavior.

The work presented in paper [49] proposed three innovative particle filter algorithms: mixture particle filter (MPF), prior-correction particle filter (PPF), and generic likelihood particle filter (GLPF), developed to improve the precision and resilience of terrain-aided navigation (TAN) for underwater robotic vehicles. These algorithms specifically tackle difficulties like uncertain topography and the limited availability of landmarks frequently encountered in marine areas. The novel algorithms exhibit improved performance in terms of estimate accuracy and resilience compared to standard filters such as sequential importance sampling filter, auxiliary particle filter, and likelihood particle filter, especially in confusing terrain settings. Through the use of real bathymetric data, the PPF method has been proven to significantly enhance localization accuracy in real-world trials, demonstrating its efficacy for TAN applications.

In research work [50], the authors introduce a new approach to tackle the difficulties of navigating in intricate, off-road terrains without the need for complete knowledge of state transition models. This method uses a second-order Taylor expansion to estimate the Bellman optimality equation. It utilizes a partial differential equation and considering only the transition model's first and second transitions. The technique improves computing efficiency by including kernel functions for value approximation, allowing for efficient policy iteration using a finite number of discrete supporting states. The method's exceptional performance in 2D obstacle avoidance and 2.5D terrain navigation is supported by extensive simulations and real-world trials, surpassing baseline techniques.

In 2021, A. Rutherford et al. presented a novel approach called rapidly exploring random Markov decision processes (RRMDPs) that significantly enhances motion planning in uncertain contexts [51]. RRMDP combines the advantages of particles rapidly exploring random trees (RRT) and rapidly exploring random graphs (RRG) to gradually build a MDP and create reliable motion strategies. This approach formalizes the problem as a cost-optimal reachability problem in a MDP and utilizes particle filtering to handle uncertainty. The experimental assessments show that RRMDP works better than pRRT in terms of robustness, allowing for more depend-

able achievement of goals under unpredictable circumstances.

A new method called SafeEst-MDP has been proposed in article [52] that incorporates Gaussian Processes (GP) to forecast environmental characteristics and integrates these forecasts into an Estimated Markov Decision Process (Est-MDP), [53], for autonomous navigation. This approach enables robots to safely navigate unfamiliar surroundings by making predictions about feature values in unexplored areas and considering probabilistic transitions. SafeEst-MDP differs from earlier approaches by explicitly considering uncertainty and optimizing exploration by selecting distant targets, resulting in improved speed and safety of exploration. The efficacy of this method has been proven by utilizing actual gamma radiation data from real-world scenarios, exhibiting improved performance compared to previous algorithms such as SafeMDP in terms of exploration efficiency and ability to handle intricate safety restrictions.

2.1.2 Map-less Navigation

Reactive Algorithms These algorithms enable robots to navigate autonomously by making real-time decisions based on sensor data. Unlike pre-planned path techniques, reactive methods adjust dynamically to the surroundings for navigating unpredictable and ever-changing environments. This flexibility is made possible through sensor data processing, allowing robots to respond promptly to alterations and obstacles in their vicinity.

In 2022, F. Melo and P. Moreno presented navigation systems for robots that can interact socially with humans [54]. This represents a noteworthy advancement in the field of autonomous navigation in human-oriented environments. This study presents an adaptive paradigm that allows robots to interact with groups of people in a socially appropriate way by flexibly modifying personal and group space characteristics according to the group's layout and spatial limitations. This adaptive technique improves the realism and context sensitivity compared to typical fixed-parameter models. When integrated with the Robot Operating System (ROS), the adaptive approach enhances navigation in situations with human groups and barriers.

The research work presented in article [55] proposed a reactive control framework designed for autonomous navigation in densely populated areas. The system

explicitly emphasizes continuous obstacle avoidance and post-contact control. The methodology entails assessing the effectiveness of two low-level obstacle avoidance techniques (Modulated Dynamical System and Reactive Driving Support) in comparison to a shared control baseline. The evaluation is based on criteria such as efficiency, controller reaction, and crowd interactions. The framework can adjust to various crowd densities and kinds, resulting in a 10% enhancement in time to reach the target in situations with high crowd density while maintaining other efficiency metrics at the same level. The system exhibited reduced jerk and increased command fluency, suggesting a high degree of compatibility with pedestrian behavior.

A. K. Akmandor and T. Padır, in 2020, introduced a 3D reactive navigation framework for mobile robots that operate without relying on global map information, achieving rapid navigation through tentacle-based sampling and heuristic evaluations [56]. The method samples the navigation space using pre-arranged navigation points on tentacles (parametric contours). It evaluates these points at each timestep based on closeness to the goal, previous tentacle preferences, and nearby obstacles within a robot-centered 3D grid. The algorithm introduces offline and online parameters to enhance adaptability and performance in unknown environments. The proposed method's effectiveness is demonstrated through physics-based simulations, showing superior performance over state-of-the-art methods regarding success rate and navigation duration.

The article [57] presents a nature-inspired technique for movement in an unknown environment, emphasizing quick responses and detailed spatial planning. This strategy imitates how insects process cues to navigate, achieving balance and avoiding obstacles with just one depth sensor (RPLIDAR A3). The method incorporates wide-field integration and Fourier residual analysis for reactive control. At the same time, a topological graph is estimated through image processing on a continuous occupancy grid to facilitate rapid path planning to unexplored edges. The integration of this reactive control with topological planning enables robust and computationally efficient navigation without explicit path planning or heavy reliance on state estimation. Rigorous field testing demonstrated the system's reliability and efficiency, notably in large and featureless environments where standard methods struggle.

2.1.3 Learning-based Algorithms

Learning-based algorithms have become a potent method for improving the capabilities of autonomous navigation systems, which can be used in both map-less and map-based navigation scenarios. These algorithms utilize developments in artificial intelligence to enable robots to independently perceive, comprehend, and navigate through intricate environments. Unlike model-driven approaches, learning-based solutions can adapt to dynamic and disorganized situations by continually acquiring knowledge and improving data.

Reinforcement learning (RL) is a crucial technique in this domain, wherein agents acquire navigation techniques through active engagement with their environment and getting feedback in the form of rewards or penalties [58]. Methods such as Deep Q Networks (DQN), [59], [60], and Proximal Policy Optimization (PPO), [34], have effectively navigated complex settings by acquiring knowledge from vast simulated encounters. In addition, supervised learning techniques have been employed to train models using datasets, including labeled navigation tasks. This allows robots to predict and execute appropriate actions depending on sensor inputs [61].

Another vital component is the integration of sensor data through networks to improve perception and decision-making. Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have demonstrated efficacy in processing visual and sequence data, leading to precise obstacle recognition, route planning, and mapping of the environment [62]. These machine learning algorithms may be seamlessly integrated into mapless navigation systems, where robots rely on real-time sensor feedback, while navigation systems with maps provide assistance in navigation tasks.

The paper [63] presents a deep reinforcement learning (DRL) method to provide safe local planning for unmanned ground vehicles (UGVs) in unfamiliar and challenging terrain. This approach utilizes reward shaping to provide frequent incentive signals. It integrates self-attention modules to improve the interpretability of the policy, enabling the vehicle to adjust to different types of terrain. After being tested in dynamic simulations, the strategy has shown to have better planning time and success rates than older approaches like potential fields and ego-graphs, while using less processing resources.

2.2 Gaps in the literature

Despite significant advancements in the field of deep reinforcement learning (DRL) for autonomous navigation, several critical areas necessitate further investigation and enhancement. The primary gaps identified in existing research include the following:

1. **Real-World Deployment and Scalability**
    - Simulation vs. Reality: Most DRL research in autonomous navigation is conducted in simulated environments. The gap between simulation and real-world deployment remains significant, with challenges related to robustness, sensor noise, and unmodeled dynamics in real-world scenarios.
    - Scalability: Scaling DRL algorithms for large-scale, real-world applications is still an open challenge. This includes navigating complex, dynamic environments such as urban settings with varying traffic conditions and unpredictable human behavior.
2. **Safety and Reliability**
    - Safety Assurance: Ensuring the safety and reliability of DRL-based navigation systems is critical, particularly in safety-critical applications like autonomous driving. The current literature needs comprehensive strategies to verify and validate DRL policies formally.
    - Risk-Aware Navigation: There is a need for DRL algorithms that can explicitly account for and mitigate risks, ensuring safe navigation even in uncertain and dynamic environments.
3. **Multi-Agent Coordination**
    - Cooperative Navigation: The literature on DRL for multi-agent systems, where multiple autonomous agents must coordinate and collaborate to achieve shared goals, is still developing. Practical communication, coordination, and conflict resolution strategies among multiple agents are needed.
    - Competitive Scenarios: Research is also needed on DRL algorithms that can handle competitive scenarios where agents might have conflicting objectives, such as in adversarial environments.

4. **Transfer Learning and Generalization**:

   - Transfer Learning: DRL models often require extensive training. However, transfer learning techniques that enable models trained in one environment to generalize and adapt to new environments with minimal retraining still need to be explored.
   - Generalization: Ensuring that DRL policies generalize well to a wide range of environments and conditions beyond those encountered during training remains a significant challenge.

5. **Sensor Fusion and Robust Perception**:

   - Multi-Sensor Integration: Integrating diverse sensor inputs (e.g., Lidar, camera, radar) for robust perception and decision-making is still a challenging problem. Effective DRL approaches for sensor fusion are needed to improve navigation accuracy and reliability.
   - Robust Perception: Developing DRL algorithms that can handle noisy, incomplete, or degraded sensor data while still making reliable navigation decisions is critical for further research.

6. **Computational Efficiency**:

   - Real-Time Processing: Many DRL algorithms are computationally intensive, challenging real-time processing. There is a need for more efficient algorithms that can operate within the computational constraints of onboard hardware in autonomous vehicles.
   - Energy Efficiency: Optimizing DRL algorithms to reduce energy consumption is vital for applications in mobile robotics, where battery life is a limiting factor.

7. **Ethical and Social Implications**:

   - Ethical Decision-Making: Incorporating ethical considerations into DRL-based navigation systems is a largely unexplored area. Developing algorithms to make ethically sound decisions in complex scenarios is crucial for societal acceptance.

- Social Acceptance: Understanding and addressing the social implications of deploying autonomous systems in public spaces, including trust, privacy, and user acceptance, is necessary for widespread adoption.

Chapter 3

Methodology

This chapter presents the methodological framework employed for training reinforcement learning (RL) agents using custom gymnasium environments. Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC) algorithms are utilized to train the agents. The custom gymnasium environment simulates simplified scenarios for agent interaction, explicitly focusing on local navigation using lidar information. By integrating LiDAR downsampling techniques, the data efficiency is significantly enhanced. This combination ensures robust training and performance evaluation of reinforcement learning agents in complex environments, allowing for effective local navigation and obstacle avoidance.

3.1 Customized Gymnasium Environment

This section comprehensively explains the custom gymnasium (standard API for reinforcement learning, [64]) environment designed to mimic a two-dimensional map with rectangular obstacles at random positions. The custom environment incorporates many modules, such as vehicle kinematics, laser-based obstacle detection, and grid-based mapping, to mimic a scenario of robotic navigation. This environment functions as a controlled experimental setting for developing and assessing Deep Reinforcement Learning (DRL) algorithms to obtain efficient and collision-free navigation. The custom gymnasium environment view is shown in Figure 3.1. Figure 3.2 represent the RL that Gymnasium implements. The agent is shown by an arrow, whose head represents the agent's current heading orientation. Lidar laser beams are shown in red lines. Note that the lidar beams are not straight, as the environment

is perceived as grid lines. The obstacles are represented by grid cells, blue in color, having grid value 1.

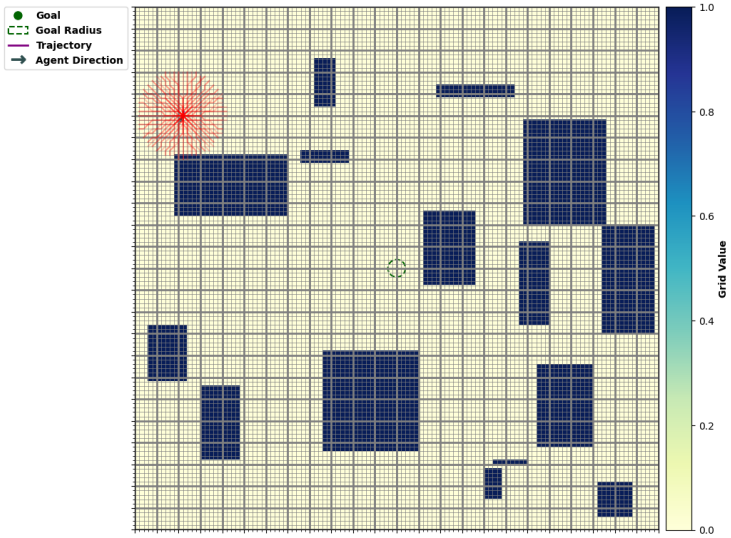


Figure 3.1: Custom Gymnasium environment (Here, red lines indicate the laser beams.)

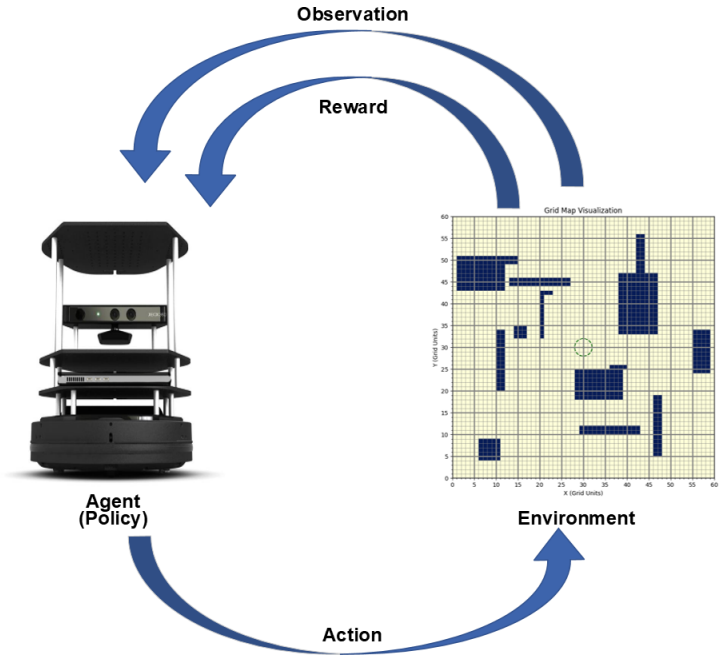


Figure 3.2: Agent-environment loop

3.1.1 Action and Observation Space in the Gymnasium Environment Framework

Action Space The action space allows the agent to maneuver across the environment by modifying its velocity and orientation. This control mechanism is crucial for the agent to reach its goal while circumventing obstacles. The action space in the custom environment is defined by two continuous values that reflect the linear and rotational velocities of the TurtleBot. The action vector A comprises of the linear velocity (η) and angular velocity (ω) and expressed as follows:

$$A = [\eta, \omega] \tag{3.1}$$

Observation Space The observation space inside the Gymnasium environment framework defines how the agent perceives the state of the environment at any given moment. The observation space in the custom TurtleBot environment is represented as a continuous space. The dimensions of the observation space are equal to the sum of five fundamental observations and the number of Lidar beams.

The fundamental observations include position coordinates (x, y), orientation θ, linear velocity v, angular velocity ω, and lidar measurements corresponding to each beam.

The observation vector O can be expressed as follows:

$$O = [x, y, \theta, v, \omega, \text{Lidar}_1, \text{Lidar}_2, \ldots, \text{Lidar}_n] \tag{3.2}$$

3.1.2 Reward Structure

The reward system is intended to incentivize quick movement while preventing collisions and staying inside the map boundaries. It encourages the DRL agent to devise tactics that maximize navigation efficiency and safety. The crucial elements of reward system encompass:

- Positive rewards for achieving the objective; reward: 100
- Negative reward for collisions with obstacles.; reward: -100
- Negative reward for exceeding boundaries.; reward: -50

- Minor penalties for each time step to incentivize quicker navigation; reward: -1
- Exceeding the threshold for a maximum number of time steps; reward: -50

3.1.3 Environmnet Workflow

Initialization

The environment is initialized with a set of settings, including the number of laser beams, goal location, number of obstacles, maximum lidar range, and grid map resolution. This setup sets up the grid map, TurtleBot, obstacles, and lidar simulation to establish a realistic simulation environment.

Reset

With the reset technique, the environment is returned to its initial state. This entails rearranging obstacles at random positions, cleaning up the grid map, and resetting the TurtleBot's orientation and location (random position inside the map at an obstacles-free place). After that, the initial observation is returned, along with the status of the TurtleBot (position and orientation) and the first measurements from its laser sensor.

Step

The step method performs a single-time step iteration inside the environment. The TurtleBot's state is modified in accordance with the action given by the DRL agent. The new state encompasses the TurtleBot's revised location, orientation, and laser sensor measurements. The algorithm computes the reward by considering the updated state. Further, it verifies if any termination criteria are fulfilled, such as achieving the goal, encountering an obstacle, or exceeding the map boundaries. The flowchart in Figure 3.3 illustrates step function process in detail.

Render

This method offers a graphical depiction of the environments. TurtleBot's navigation behavior may be better understood and debugged with the use of frames or

RGB arrays that humans can easily read. It is essential for visualizing the training process and the performance of the DRL algorithms.

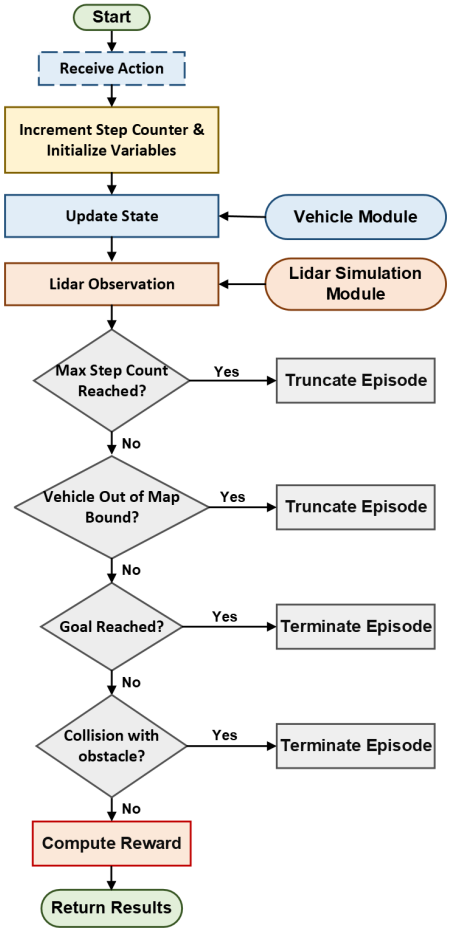


Figure 3.3: Flowchart of the step function in the custom Gymnasium environment, detailing the process from receiving the agent's action to updating the state, simulating laser beams, and calculating rewards and observations.

3.1.4 Vehicle (Turtlebot)

The vehicle component imitates the TurtleBot by replicating its location, orientation, and mobility capabilities. The state of the TurtleBots can be updated utilizing approaches from the system that depend on actions performed by the DRL agent. Additionally, the system has implemented procedures to restore the TurtleBot to its initial state. This component is crucial for accurately reproducing the locomotion and behavior of the TurtleBot inside its environment.

Mechanical Attributes

- **Position and Orientation**

  - **Position**: The TurtleBot's position is indicated by Cartesian coordinates (x, y) on the 2D grid map. These coordinates represent the precise position of the TurtleBot on the map concerning the world coordinate system.
  - **Orientation**: The TurtleBot's orientation is expressed in radians, indicating the angle between its current direction and the positive x-axis. This angle is normalized within the range of -π to π.

- **Movement:**
  - **Linear Velocity**: The TurtleBot can move forward or backward at a predetermined linear velocity. This velocity defines the distance covered in the direction the TurtleBot faces.
  - **Angular Velocity**: It refers to the rate at which the TurtleBot may rotate around its vertical axis. This spin alters the orientation of the TurtleBot, enabling it to maneuver and navigate around obstacles.

The movement of the TurtleBot can be characterized by the subsequent kinematic equations:

1. **Orientation Update**:

$$\theta_{t+1} = \theta_t + \omega \Delta t \tag{3.3}$$

   where, θ_{t+1} stands for the updated orientation, θ_t represents the current orientation, ω indicates the angular velocity, and Δt is the time step.

   The updated orientation is normalized to ensure it remains within the range of -π to π.

2. **Position Update**:

$$x_{t+1} = x_t + v \cos(\theta_{t+1}) \Delta t \tag{3.4}$$
$$y_{t+1} = y_t + v \sin(\theta_{t+1}) \Delta t \tag{3.5}$$

   where, (x_{t+1}, y_{t+1}) are the updated position coordinates, (x_t, y_t) are the current position coordinates, and v signifies the linear velocity.

3. **Boundary Constraints**: It is essential to ensure that TurtleBot stays inside the designated map borders when operating in the environment. Boundary

limitations and padding are used to accomplish this. The clipping equation is as follows:

$$x_{t+1} = \text{clip}(x_{t+1}, x_{\min} + \text{padding}, x_{\max} - \text{padding}) \tag{3.6}$$
$$y_{t+1} = \text{clip}(y_{t+1}, y_{\min} + \text{padding}, y_{\max} - \text{padding}) \tag{3.7}$$

Here, (x_{min}, y_{min}) and (x_{max}, y_{max}) represent the minimum and maximum coordinates of the map boundaries. Padding refers to a safety buffer that is included inside the bounds of the map in order to prevent the TurtleBot from approaching the edges too closely. This prevents instances when the TurtleBot may partially exit the map or meet issues caused by boundary circumstances.

3.1.5 LIDAR Sensor Simulation

The LIDAR (Light Detection and Ranging) simulation component is an essential element of the vehicle's sensor system used for autonomous navigation. The software emulates the functionality of a lidar rangefinder sensor by accurately measuring the distances to obstacles present in the surrounding environment. These measures are crucial for the TurtleBot to understand its environment and make well-informed judgments during navigation.

To generate multiple laser beams that emanate from the TurtleBot's position and spread evenly around it based on its orientation, following key points are followed sequentially:

1. **Angular Increment and Beam Angle Calculation**: The beams are spaced evenly around a 360-degree field of view. The angular increment between each beam is calculated by dividing the entire circle (2π radians) by the number of beams. For each beam, the angle is adjusted by the agent's current orientation, ensuring that the beams cover the area around the TurtleBot uniformly.

2. **Beam End Point Calculation**: The endpoint of each beam is determined based on the beam's angle and maximum range of the Lidar sensor.

3. **Points calculation along each beam** Bresenham's algorithm is employed to determine the particular grid points that properly depict a straight-line laser beam connecting its starting and ending points. Beforehand, each beam's initial and final positions are transformed into grid coordinates to simplify the

implementation of the method. This transformation is performed using the equations outlined below:

For a single point, the grid coordinates are calculated as:

$$\text{grid\_x} = \left\lfloor \frac{x_w - x_{\min}}{r} \right\rfloor \tag{3.8}$$

$$\text{grid\_y} = \left\lfloor \frac{y_w - y_{\min}}{r} \right\rfloor \tag{3.9}$$

If a size is provided, the function calculates the grid coordinates for the entire region as:

$$\text{grid\_x\_end} = \left\lfloor \frac{x_w + s_x - x_{\min}}{r} \right\rfloor \tag{3.10}$$

$$\text{grid\_y\_end} = \left\lfloor \frac{y_w + s_y - y_{\min}}{r} \right\rfloor \tag{3.11}$$

Here, (x_w, y_w) signifies world coordinates of a point and $(x_{\min}, y_{\min})$ indicates minimum bounds of the map in the x and y directions, respectively. r denotes grid resolution, i.e., the size of each grid cell, and (s_x, s_y) represents the size of the region in the x and y directions, respectively.

4. **Compute observation along each beam**: The distance from the TurtleBot to the first obstacle encountered along the beam's path is stored as observation. If no obstacle is found along a beam, a default value (e.g., -1) is appended to indicate the absence of an obstacle.

5. **Return Observations**: The list of distances is returned as a numpy array, providing a structured format for further processing.

3.1.6 Lidar Data Preprocessing

Lidar data processing is essential for precise perception and decision-making in the field of autonomous navigation. Several strategies can be employed to preprocess the Lidar information. A hand-crafted feature extraction mechanism inspired by [36] is presented here. In this work, preprocessing includes reducing the Lidar data to a specific angular resolution and limiting the distance measurements to a predetermined maximum range. Algorithm is described as follows:

1. Calculate the downsampling parameters: This process decreases the number of Lidar beams in order to get the desired angular resolution.

   - Calculate the initial resolution:

   $$\text{initial\_resolution} = \frac{360.0}{\text{num\_beams}} \tag{3.12}$$

   - Compute the downsampling ratio:

   $$\text{downsample\_factor} = \left\lceil \frac{\text{target\_resolution}}{\text{initial\_resolution}} \right\rceil \tag{3.13}$$

   - Verify that the downsampling factor is acceptable: Throw a ValueError exception if the value is less than 1.
   - Calculate number of sectors:

   $$\text{sectors} = \left\lceil \frac{\text{num\_beams}}{\text{downsample\_factor}} \right\rceil \tag{3.14}$$

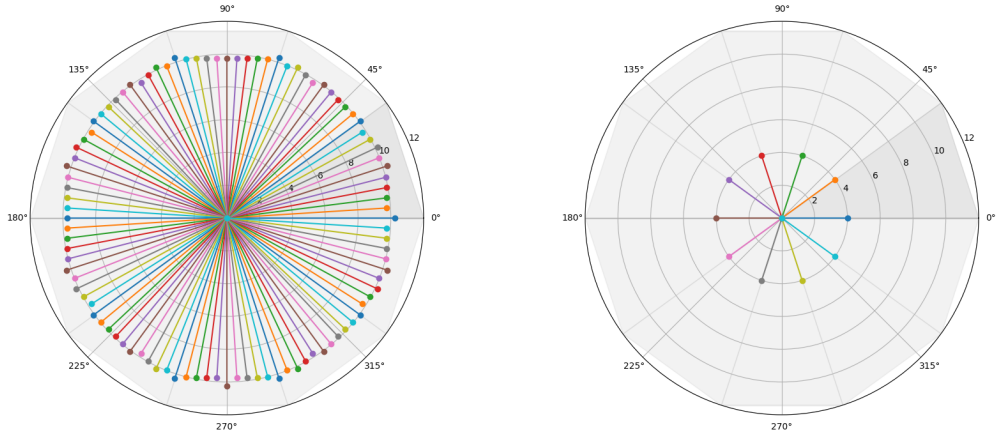
2. Iterate over the Lidar data in sectors according to the specified downsample factor. For each sector:

   - Substitute the value of -1 corresponding to actual observations with infinity in order to facilitate min-pooling.
   - Limit the distances to the maximum range.
   - Compute the minimum distance for the segment as observation. If the minimum distance is equal to maximum range, then use -1 as observation for that segment.
   - Append the observation for that segment to the downsampled observations array.

3. Verify that the reduced-resolution observations size correspond to the anticipated number of sectors.

4. Return the downsampled and clipped Lidar observations as a numpy array.

For a comprehensive understanding of downsampling approach implementations, refer to Appendix A for the Code A.1.



(a) Original 100 beams Lidar (maximum range 10 m) data with sectors

(b) Downsampled 10 beams Lidar data with maximum clip range of 4 m

Figure 3.4: Lidar data downsampling

3.2 Deep Reinforcement Learning Algorithms

3.2.1 Proximal Policy Optimization (PPO)

PPO is a reinforcement learning (RL) technique that is specifically developed to update policies in a dependable and efficient manner. Schulman, in 2017, proposed the PPO algorithm that achieves a trade-off between simplicity and performance [34]. It addresses the limitations of previous techniques like Trust Region Policy Optimization (TRPO) [65], which involve computing complexity, scalability problems, and optimization difficulties, by utilizing first-order optimization methods.

Background PPO is based on the policy gradient framework, which aims to directly optimize the policy by maximizing an objective function that reflects the expected return. Conventional policy gradient approaches frequently experience instability and inefficiency as a result of making substantial revisions to the policy. PPO addresses these problems by setting limitations on policy modifications to provide stability.

Mathematical Context The objective in the policy gradient framework is to optimize the policy parameters θ by maximizing the expected return $J(\theta)$ [66]. This is achieved by modifying the parameters in the direction of the gradient of the cost

function $J(\theta)$.

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s,a) \right] \tag{3.15}$$

where, $\pi_\theta(a|s)$ is the policy, and $Q^{\pi_\theta}(s,a)$ indicates the action-value function. Conventional policy gradient approaches can be unstable due to the potential for substantial changes in π_θ (the policy) resulting from extensive updates to θ (the parameters), which can negatively impact performance. PPO addresses this problem by providing a surrogate objective function that includes a technique for limiting the range of values it can take.

$$L^{CLIP}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{3.16}$$

where

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{3.17}$$

Here, $r_t(\theta)$ indicates the ratio between new and old policies, and $\hat{A}_t$ signifies the advantage estimate at timestep t. The clipping mechanism, $L^{CLIP}(\theta)$, constrains the function $r_t(\theta)$ within the interval $[1-\epsilon, 1+\epsilon]$, hence preventing significant updates that may disrupt the training process. ϵ is a hyperparameter.

Adaptive Kullback-Leibler (KL) Coefficient PPO can include an adaptive Kullback-Leibler (KL) penalty to guarantee that the new policy remains close to the previous policy and does not deviate significantly. The KL divergence quantifies the disparity between the updated policy π_θ and the previous policy $\pi_{\theta_{\text{old}}}$. The adaptive KL penalty coefficient modifies the intensity of the punishment by considering the KL divergence between consecutive policies. If the KL divergence exceeds a specified threshold, the penalty is augmented in order to discourage substantial updates; conversely, if the divergence is below the threshold, the penalty is reduced to facilitate greater exploration. The revised objective, with an adaptable KL penalty, is as follows:

$$L^{\text{KL}}(\theta) = \mathbb{E}_t \left[r_t(\theta)\hat{A}_t - \beta \text{KL}[\pi_{\theta_{\text{old}}}(\cdot|s_t) || \pi_\theta(\cdot|s_t)] \right] \tag{3.18}$$

where β denotes the adaptive KL penalty coefficient, and $\text{KL}[\cdot\|\cdot]$ represents the KL divergence.

The PPO algorithm flowchart is exhibited in Figure 3.5. The policy and value function update follows the subsequent computation.

- Policy Update: Modify the policy by optimizing the clipped surrogate objective by utilizing stochastic gradient descent (SGD). The update rule may be defined as follows:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} \min \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} \hat{A}_t, \right.$$
$$\left. \text{clip} \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}, 1-\epsilon, 1+\epsilon \right) \hat{A}_t \right) \tag{3.19}$$

  where $|D_k|$ is the size of the dataset collected in iteration k, and T is the length of each trajectory.

- Value Function Update: Adjust the value function by reducing the mean squared error (MSE) between the expected value and the actual returns.

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2 \tag{3.20}$$

  where $\hat{R}_t$ signifies the rewards-to-go (or return) at time step t.

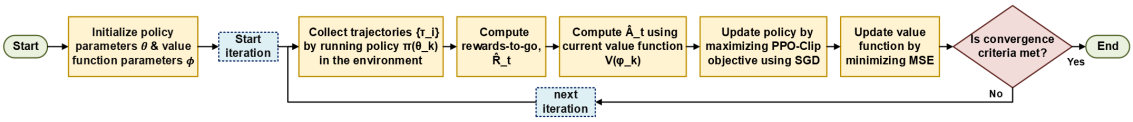


Figure 3.5: Flowchart of the PPO Algorithm

Exploration vs. Exploitation The PPO algorithm achieves a trade-off between exploration and exploitation by training a policy that incorporates randomness. At first, the policy is mainly focused on exploration, where it does random acts to uncover potential rewards. As the policy evolves, it becomes increasingly deterministic, prioritizing activities that result in greater returns. This gradual transition facilitates the identification of an optimal strategy while circumventing the occurrence of local optima.

3.2.2 Soft Actor-Critic (SAC) Algorithm

Soft Actor-Critic (SAC) algorithm was proposed by T. Haarnoja, et al. in 2018 [33]. It is an advanced model-free, off-policy strategy that combines elements of both value-based and policy-based reinforcement learning techniques, known as the actor-critic approach. It is designed to address the stability and sample efficiency challenges often encountered in DRL. The algorithm optimizes a stochastic policy in an off-policy way, utilizing the maximum entropy framework to encourage exploration by maintaining high entropy policies.

Actor-Critic algorithm

The Actor-Critic algorithm merges policy gradient and value function strategies in the realm of RL. It is a technique based on policy gradient that integrates Temporal Difference (TD) methods. The system consists of two components: the Actor and the Critic. The architecture of actor-critic technique is depicted in Figure 3.6.

Mechanism of Actor-Critic

1. Actor Network:
    - The actor determines its course of action in a certain situation by adhering to a policy $\pi_\theta(a \mid s)$, where θ represents the parameters of the policy network, s denotes the state of the environment, and a signifies the action taken by the agent.
    - The objective of the actor is to optimize the anticipated outcome by adjusting the policy parameters in a manner that enhances performance.
2. Critic Network:
    - Critic assesses the actor's actions by estimating the value function, usually represented as $\mathrm{Q}_\theta(a \mid s)$.
    - Critic offers an evaluation of the actor, assessing the effectiveness of the action done and suggesting adjustments to the policy to enhance future acts.

Advantage Function in Actor-Critic Within policy gradient approaches, the advantage function quantifies the disparity between the observed return and a

reference point, usually the value function. In Actor-Critic algorithms, the advantage function is commonly denoted as the Temporal Difference (TD) error. This error represents the disparity between the anticipated value and the sum of the observed reward plus the projected value of the subsequent state.

The modified advantage function in the Actor-Critic framework may be mathematically represented as:

$$A_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{3.21}$$

where A_t is the advantage function and r_t is the reward at t timestep. γ is the discount factor, and $V(s_t)$ is the value function of the current state.

Policy Gradient in Actor-Critic The learning of the actor is driven by the policy gradient method. The policy gradient expression for the actor is:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho_\pi, a\sim\pi_\theta} [\nabla_\theta \log \pi_\theta(a \mid s) A_t] \tag{3.22}$$

where $\nabla_\theta J(\theta)$ is the gradient of the expected return with respect to the policy parameters.

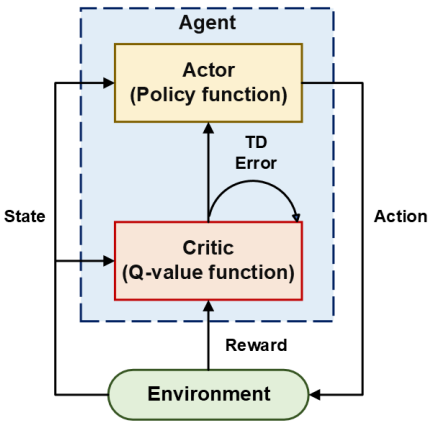


Figure 3.6: Architecture of Actor-Critic Algorithm

3.2.3 Mathematical Foundation of SAC

1. Maximum Entropy Objective: The SAC objective integrates entropy to ensure the policy remains stochastic, encouraging exploration. The objective function

is given in the succeeding equation.

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi} \left[r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t)) \right] \tag{3.23}$$

$$\mathcal{H}(\pi(\cdot|s_t)) = -\mathbb{E}_{a_t\sim\pi(\cdot|s_t)} \left[\log \pi(a_t|s_t) \right] \tag{3.24}$$

where $\mathcal{H}(\pi(\cdot|s_t))$ represents the entropy of the policy (π), whereas α is a temperature parameter that determines the balance between the reward and entropy. $\mathbb{E}$ denotes the expected value or expectation.

2. Soft Q-Function: It incorporates the entropy term and defined as follows:

$$Q^\pi(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1}\sim p} \left[V(s_{t+1}) \right] \tag{3.25}$$

where $V(s_{t+1})$ is the soft state value function:

$$V(s_{t+1}) = \mathbb{E}_{a_{t+1}\sim\pi} \left[Q(s_{t+1}, a_{t+1}) - \alpha \log \pi(a_{t+1}|s_{t+1}) \right] \tag{3.26}$$

3. Policy Improvement: The policy is updated by minimizing the KL divergence between the policy and the exponentiated soft Q-function.

$$\pi_{\text{new}} = \arg\min_{\pi'} D_{\text{KL}} \left(\pi'(\cdot|s_t) \| \frac{\exp(Q(s_t, \cdot))}{Z(s_t)} \right) \tag{3.27}$$

where ($Z(s_t)$ denotes a normalizing constant.

4. Temperature Update: The temperature parameter α is modified to regulate the entropy of the policy. The mathematical model for α computation is:

$$J(\alpha) = \mathbb{E}_{a_t\sim\pi_t} \left[-\alpha \log \pi_t(a_t|s_t) - \alpha H_{\text{target}} \right] \tag{3.28}$$

where H_{target} indicates a target entropy.

Algorithm Steps

1. Initialize Networks:

   - Two Q-networks; $Q_{\theta_1}(s, a)$ and $Q_{\theta_2}(s, a)$
   - A policy network; $\pi_\phi(a \mid s)$

- A target value network; $V_\psi(s)$

2. Experience Replay: Maintain a replay buffer $\mathcal{D}$ to store experiences (transitions) (s, a, r, s'). Each transition consists of a state (s), an action (a), a reward (r), and the resulting state (s').

3. Policy Evaluation (Critic Update):

   - Retrieve a subset of transitions, called a mini-batch, from the replay buffer.
   - Compute the target value using the target network:

$$y = r + \gamma \mathbb{E}_{s' \sim p} \left[V_{\psi'}(s') \right] \tag{3.29}$$

     where y stands for the target value used to update the Q-networks, r indicates reward received, γ is the discount factor, representing the importance of future rewards, and p denotes the transition probability distribution. $V_{\psi'}(s')$ is the estimated value of the next state according to the target value network.

   - Optimize the Q-function parameters θ_i by minimizing the Bellman residual.

$$\mathcal{L}(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[\left(Q_{\theta_i}(s, a) - y \right)^2 \right] \tag{3.30}$$

     where $\mathcal{L}(\theta_i)$ indicating the loss function for the policy network, which promotes higher entropy (more exploration) and higher anticipated rewards.

4. Policy Improvement (Actor Update): Update the policy parameters ϕ by maximizing the expected reward plus entropy:

$$\mathcal{L}(\phi) = \mathbb{E}_{s \sim \mathcal{D}} \left[\mathbb{E}_{a \sim \pi_\phi} \left[\alpha \log(\pi_\phi(a \mid s)) - \min_{i=1,2} Q_{\theta_i}(s, a) \right] \right] \tag{3.31}$$

5. Temperature Update: By utilizing Equation 3.28, the temperature parameter, α, is adjusted to control the stochasticity of the policy.

6. Target Network Update: Periodically update the target network parameters ψ' to slowly track the Q-network parameters:

$$\psi' \leftarrow \tau\theta + (1-\tau)\psi' \tag{3.32}$$

where τ signifies the update rate, determining how quickly the target network parameters track the Q-network parameters.

Advantages of SAC

- Sample Efficiency: SAC operates on off-policy, enabling it to utilize experience samples repeatedly, hence enhancing sample efficiency.
- Stability: The employment of twin Q-networks and entropy regularization ensures the stability of training and mitigates the problem of overestimating Q-values.
- Exploration: The maximum entropy framework promotes increased exploration, which is advantageous in intricate contexts.

3.2.4 DRL algorithms implementation

The following implementations of DRL were tested to evaluate the effectiveness of the DRL algorithms in the proposed Gymnasium environment:

1. **Training with SAC**:
   - It is implemented using the Stable-Baselines3 library.
   - The agent was trained to navigate from random starting points to the goal, avoiding obstacles.
   - Various reward functions (sparse and dense) were tested to determine the most effective structure for enhancing learning performance.
2. **Training with PPO**:
   - It is also implemented using the Stable-Baselines3 library.
   - A similar training setup as SAC focuses on navigation efficiency and obstacle avoidance.

- Hyperparameter tuning was conducted to optimize the performance of PPO.

3. **Dynamic-Window Approach (DWA)**:

   - The DWA controller, developed specifically for this study, was used as a baseline comparison.
   - The test focused on obstacle avoidance and navigation, albeit highly sensitive to hyperparameter settings.

Chapter 4

Results and Discussion

This chapter demonstrates the results obtained from applying Deep Reinforcement Learning (DRL) algorithms, using Stable-Baselines3 implementation, to the custom simulation environment developed for this study. Although the initial objective was to evaluate these algorithms in more advanced simulators such as Gazebo or Stonefish and on a physical TurtleBot platform, the scope was limited to the custom environment due to various constraints. Despite these limitations, the findings offer valuable insights into the efficacy of DRL for autonomous navigation in GPS-denied environments.

4.1 Experimental Setup

This section provides a comprehensive explanation of the experimental setting employed to assess the efficacy of Deep Reinforcement Learning (DRL) algorithms, Soft Actor-Critic (SAC), and Proximal policy optimization (PPO) in achieving autonomous navigation. The trials were conducted in a custom constructed Gymnasium environment with specified settings to replicate simpler navigation conditions. The DRL algorithms were built using the Stable-Baselines3 [67] library, renowned for its cutting-edge implementations of diverse reinforcement learning methods.

Custom Gymnasium Environment The Gymnasium environment, as explained in Chapter 3, was designed to mimic an unstructured simplified navigation setting, to assess the resilience and effectiveness of the DRL algorithms. The general

characteristics of the environment are outlined in Table 4.1, followed by explanations of each parameter.

Table 4.1: Custom Gymnasium Environment parameters varied for different experiments

Parameter	**Value**
num_beams	0-100
map_bounds	[(-30, -30), (30, 30)] (m)
goal_position	[0, 0]
num_obstacles	0-15
max_range	5 (m)
grid_resolution	0.5
edge_padding	3 (m)
centre_padding	10 (m)

- num_beams: The lidar sensor is configured to provide 10 beams.
- map_bounds: The environment's dimensions are set to a 60m x 60m area, defined by the coordinates [(-30, -30), (30, 30)]. This large area allows for significant maneuvering space and obstacle placement.
- goal_position: The goal for the autonomous vehicle is placed at the origin [0, 0]. The vehicle's objective is to navigate from random starting points to this goal position.
- num_obstacles: There are 15 static obstacles randomly placed within the environment.
- max_range: The maximum range of the lidar sensor is set to 5 meters. This defines the radius within which the sensor can detect obstacles and map the environment.
- grid_resolution: The grid map resolution is 0.5 meters. This resolution determines the granularity of the grid cells used for mapping the environment and obstacle positions.
- edge_padding: Set to 3 meters from the boundary inside the map where the agent cannot be initialized, preventing it from starting too close to the edge and potentially moving outside the map.

- centre_padding: Set to 10 meters, creating a central exclusion zone around the goal position where the agent cannot be initialized, ensuring more diverse starting positions.

Algorithms and Training The DRL algorithms were selected and implemented using the Stable-Baselines3 library. Two prominent algorithms were chosen for evaluation: Soft Actor-Critic (SAC) and Proximal Policy Optimization (PPO).

4.1.1 Scenarios

A series of scenarios with increasing complexity of the environment has been designed. All of them use the parameters in Table 4.1 with a few changes:

- **No obstacles scenario**: In this scenario, the number of beams is set to 10 and the number of obstacles to 0. The goal is to start from a random position and reach the position (0,0).
- **Range sensors with obstacles**: In this case, several 15 obstacles have been added randomly to the environment. The robot is equipped with 10 range sensors equally spaced in 360°around it to measure distances to its surroundings.
- **Lidar sensor with obstacles**: This scenario is similar to the previous one, but instead of simulating only 10 range sensors, we simulate 100 beams around the robot, as a 2D Lidar would provide. The number of obstacles remains the same as in the previous scenario.

4.2 Results and Analysis

The results of training and testing the agent using Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC) algorithms in a custom environment with different scenarios are presented in following sections. The performance metrics include Total Loss and Mean Episodic Reward over Total Timesteps explained as follows:

- Mean Episodic Reward: The average total reward earned by the agent in each episode is known as the episodic reward, which reflects how effectively the agent accomplishes its objectives. Monitoring changes in the episodic reward offers valuable insights into how efficiently and effectively the algorithm is learning

over time. A higher mean episodic reward signifies better performance as the agent successfully navigates the environment to reach its objectives.

- Total Loss: The total loss in DRL algorithms includes many components that assess the effectiveness and consistency of the learning process. PPO encompasses the policy loss, value function loss, and entropy bonus, which collectively guarantee successful policy updates while preserving enough exploration. The total loss for the SAC algorithm consists of three components: the critic loss, which represents Q-value estimation error; the actor loss, which measures the performance of the policy; and the entropy term, which encourages exploration. Analyzing the overall loss aids in comprehending the algorithm's capacity to balance learning an ideal policy and sustaining stability during the training process.

4.2.1 No obstacle Senario

The mean episodic reward and total loss graphs over timesteps for no obstacle in environment case, for SAC and PPO are exhibited in Figure 4.1 as follows: (a) and (c) show the mean episodic rewards for PPO and SAC respectively, while (b) and (d) display the total loss for PPO and SAC respectively.

The SAC and PPO demonstrate unique properties in their training dynamics. PPO Mean Episodic Reward curve exhibits a slower increase in the mean episodic reward, with significant improvements occurring later in the training process. This gradual improvement suggests that PPO may require more timesteps to converge to an optimal policy than SAC. SAC shows a rapid increase in the mean episodic reward, reaching approximately 80% of its maximum reward within the first 200,000 timesteps. This indicates that SAC is highly effective at quickly learning an optimal policy for the given environment.

The SAC loss curve displays rapid convergence and early stability during training, making it well-suited for applications that need swift adaptability and consistent performance. Conversely, the PPO total loss curve converges slowly. It has regular policy tweaks, which indicates it may be more appropriate for situations requiring cautious updates and stability in the learning process.

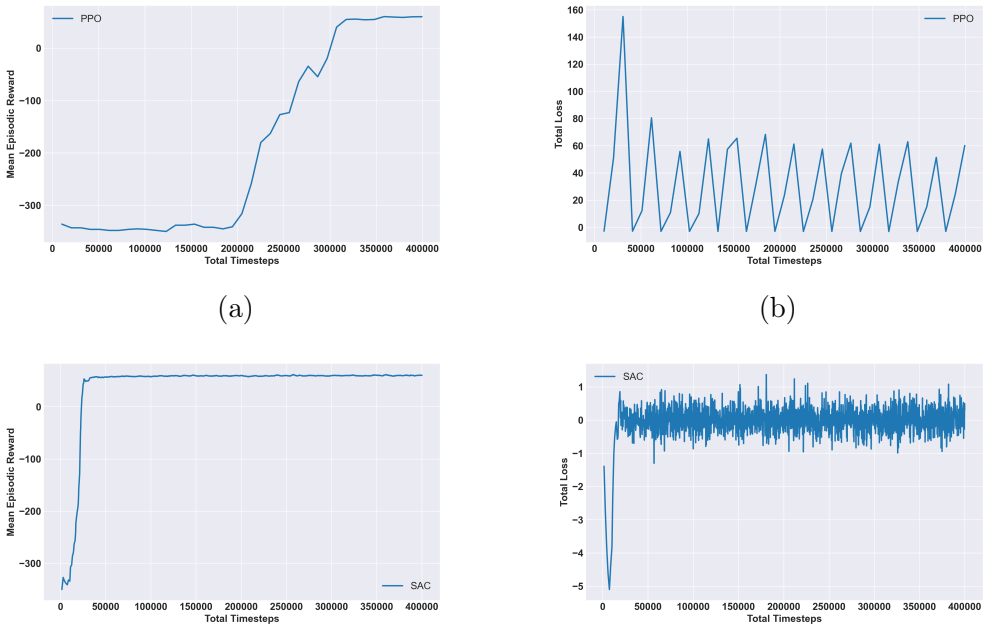
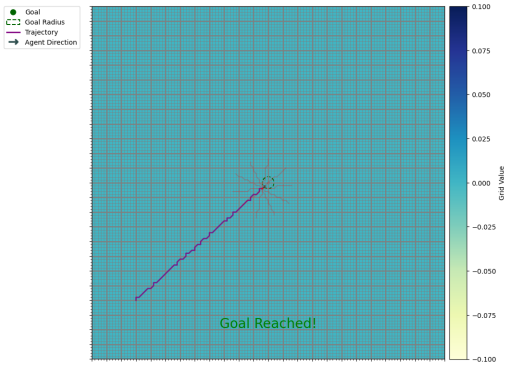


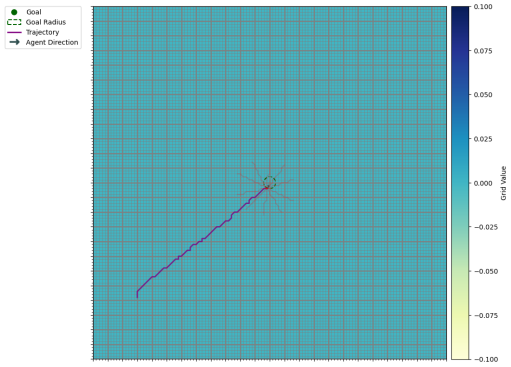
Figure 4.1: Performance comparison between PPO and SAC trained in an environment without obstacles. (a) Mean episodic reward for PPO, (b) Total loss for PPO, (c) Mean episodic reward for SAC, (d) Total loss for SAC.

Figure 4.2 displays the trajectories of agents trained using the Dynamic Window Approach (DWA), PPO, and SAC algorithms tested in a no-obstacle environment. Subfigures (a), (b), and (c) correspond to the trajectories of DWA, PPO, and SAC, respectively.

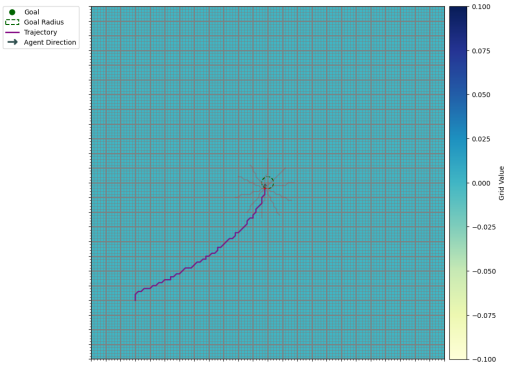
SAC demonstrates a curved route with adjustments, showing its effectiveness in reaching the destination, although its navigation could be more straightforward and steady than PPO and DWA. PPO displays a direct path to the goal with minimal adjustments, indicating efficient and consistent learning but somewhat less effective than DWA. DWA demonstrates the direct and seamless path to the destination, implying that it navigates efficiently and steadily. However, these paths do not determine exact inference, as the performance and trajectories heavily depend on the parameter tuning for each algorithm. Further experiments with varied parameter settings are necessary to draw more definitive conclusions about the relative performance of these algorithms.



(a) Trajectory covered by DWA to reach the goal.



(b) Trajectory covered by PPO to reach the goal.



(c) Path followed by SAC to reach the goal.

Figure 4.2: Comparison of trajectories followed by DWA, PPO, and SAC to reach the goal in an environment without obstacles.

4.2.2 Range sensor with obstacles

In this scenario, the agent equipped with a 10-beam Lidar sensor is trained in an environment containing 15 obstacles. Figure 1 displays graphs of the learning and training curves for SAC and PPO algorithms for the obstacle environment. **Qualitative and quantitative analysis for PPO trained policy**

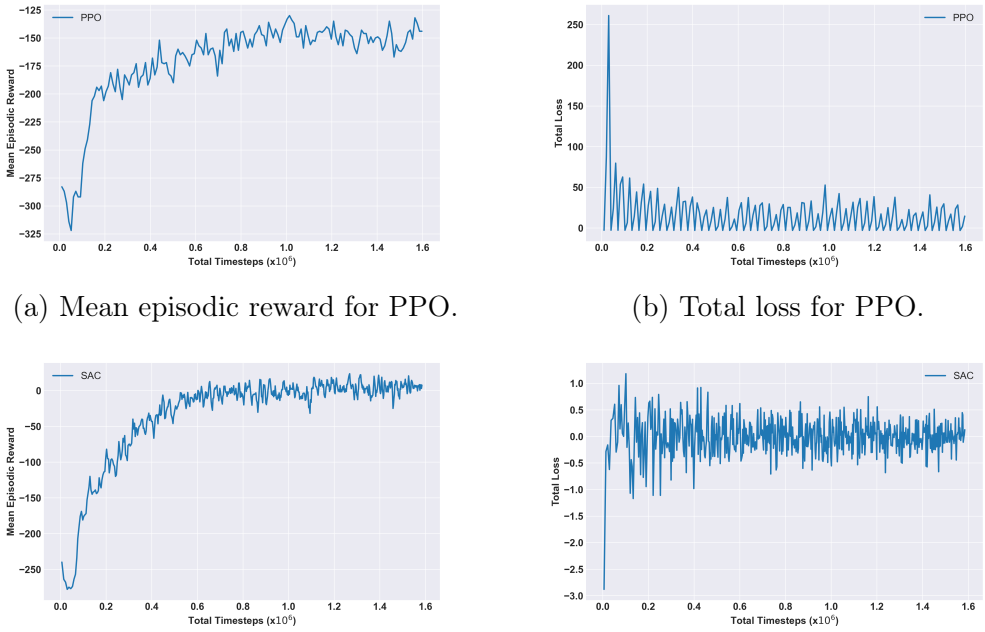
- Figure 4.3 (a) represents the improvement in mean episodic reward for PPO over total timesteps; reward shows a gradual increase, stabilizing around -125 after 1.6 million timesteps. While the reward improves over time, the slower rate of improvement and lower stabilized reward compared to SAC indicate that PPO is less efficient in adapting to the environment with obstacles.

- Figure 4.3 (b) exhibits the total loss curve for PPO; it can be easily inferred from the curve that periodic spikes throughout the training process, with initial spikes reaching up to 250 and gradually decreasing in magnitude. These

spikes indicate periodic significant adjustments in the policy, which can cause instability. However, the decreasing magnitude of spikes reflects gradual learning and adaptation to the environment.

Qualitative and quantitative analysis for SAC trained policy

- Mean episodic curve over total timesteps curve represented in Figure 4.3 (c), indicates that mean episodic reward starts around -250. The reward steadily increases, stabilizing around 50 after 1.6 million timesteps. This steady rise indicates effective learning and adaptation to the environment with obstacles.
- The total loss graph for SAC shows significant fluctuations during the early stages of training, dropping to around -3 initially and then varying between -1.5 and 1.0 for the first 200,000 timesteps, Figure 4.3 (d). After approximately 400,000 timesteps, the total loss stabilizes around zero, indicating that SAC has learned an effective policy to navigate through the obstacle-laden environment.



(a) Mean episodic reward for PPO.

(b) Total loss for PPO.

(c) Mean episodic reward for SAC.

(d) Total loss for SAC.

Figure 4.3: Performance comparison between PPO and SAC trained in an environment with 15 obstacles in the environment. Subfigures (a) and (c) show the mean episodic rewards for PPO and SAC, respectively, while subfigures (b) and (d) display the total loss for PPO and SAC, respectively.

In conclusion, both the SAC and PPO algorithms demonstrate good learning and navigation in obstacle-filled environments, enabling them to attain their objectives

successfully. Nevertheless, the SAC method exhibits accelerated convergence, enhanced stability, and superior performance, rendering it more appropriate for complicated environments containing obstacles. Although PPO is successful, it exhibits a slower rate of convergence and poorer performance, suggesting that further training or parameter adjustment may be required to attain results equivalent to SAC. Quantitatively, the mean episodic reward achieved by SAC is 50, whereas PPO stabilizes at -125, indicating that SAC outperforms PPO. From a qualitative perspective, the SAC algorithm offers a more consistent and efficient learning process, whereas the periodic spikes observed in PPO indicate a more volatile training process.

Test Environment Results and Analysis The performance of policy trained with PPO and SAC algorithms was evaluated in a random test environment containing obstacles. Additionally, the DWA strategy was tested in this environment. The trajectories achieved by each agent are illustrated in Figure 4.4.

DWA Performance

- Navigation and Stoppage: The trajectory of the DWA-controlled agent (Figure 4.4) (a) shows that it managed to navigate initially but halted in front of obstacles due to improper hyperparameter tuning. The agent could have proceeded more effectively as there was navigable space.

- Tuning Challenges: The performance issues highlight the importance of precise hyperparameter tuning in DWA for successful navigation in complex environments. For example, if the obstacle avoidance parameters are not set correctly, the agent might either stop prematurely or collide with obstacles, as observed in the test scenario.

PPO Performance

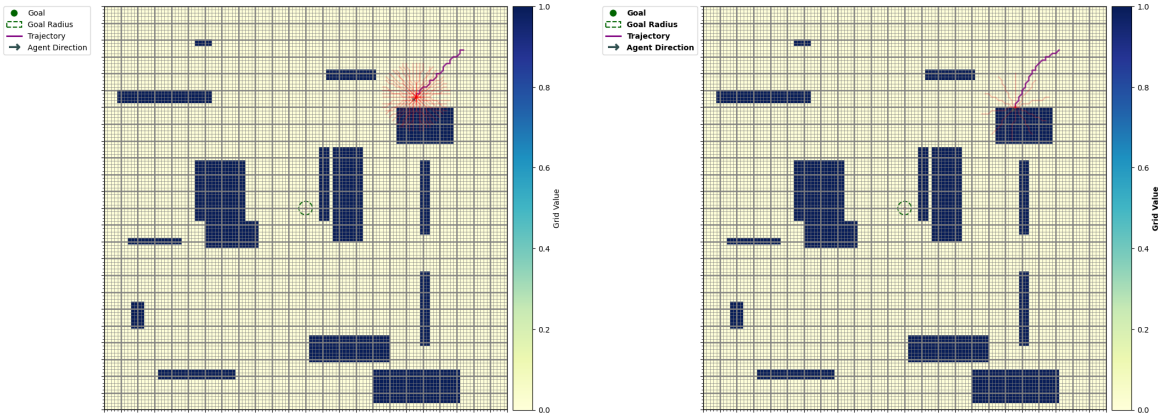
- Navigation and Collision: The trajectory of the PPO-trained agent (Figure 4.4 (b) demonstrates collisions with objects, even when clear pathways are available. This suggests that the decision-making process is not ideal when faced with obstacles.

- Hyperparameter Sensitivity: The results indicate that the fine-tuning of hyperparameters greatly influences the success of the PPO algorithm. Insufficient calibration may result in subpar navigation and collisions with obstructions. Thus, to improve obstacle avoidance and guarantee the agent can identify clear

pathways to the objective, it is necessary to make precise adjustments to parameters such as the clipping range, learning rate, and the coefficient for the value function loss.

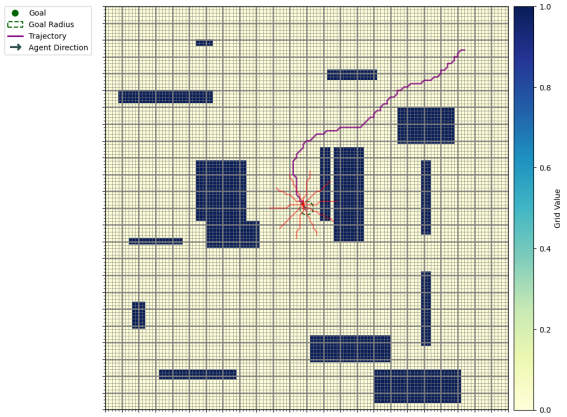
SAC Trained Policy Performance

- Navigation and Goal Achievement: As shown in Figure 4.4 (c), the SAC-trained agent successfully navigated through the environment and reached the goal, even when the goal was positioned behind obstacles.
- Adaptability: SAC demonstrated robust adaptability, effectively handling the complexities of the environment and finding paths around obstacles to reach the goal.



(a) DWA performance on random test environment.

(b) Trained PPO policy trajectory on test environment.



(c) SAC trained agent trajectory on test environment.

Figure 4.4: Visualizations of trained agent trajectories tested in a random environment.

In conclusion, the results demonstrate that SAC excels in adaptability and effective navigation in environments with obstacles, showing resilience to suboptimal initial conditions. In contrast, PPO and DWA significantly depend on hyperparameter tuning for optimal performance.

4.2.3 Lidar Data Downsampling Scenario

In this scenario, the agent equipped with a 100-beam Lidar sensor is trained using SAC in an environment containing 15 obstacles. Two variations are considered: one agent uses the full 100 beams, while the other uses downsampled Lidar data to 10 beams. Figure 4.5 display the learning and training curves for both configurations, respectively.

The results highlight the trade-offs between sensory data resolution and learning efficiency. The agent with downsampled Lidar data to 10 beams exhibits quicker learning and stabilization of mean episodic reward, a shown in Figure 4.5 (a), indicating effective policy adaptation with limited sensory input. However, this configuration shows higher variability in total loss, exhibited in Figure 4.5 (b), reflecting potential instability due to reduced data.

In contrast, the agent with the full 100-beam Lidar data demonstrates more consistent and stable learning, albeit with a slower convergence rate. The lower variability in total loss post-stabilization suggests that more extensive sensory data contributes to a more robust and stable policy.

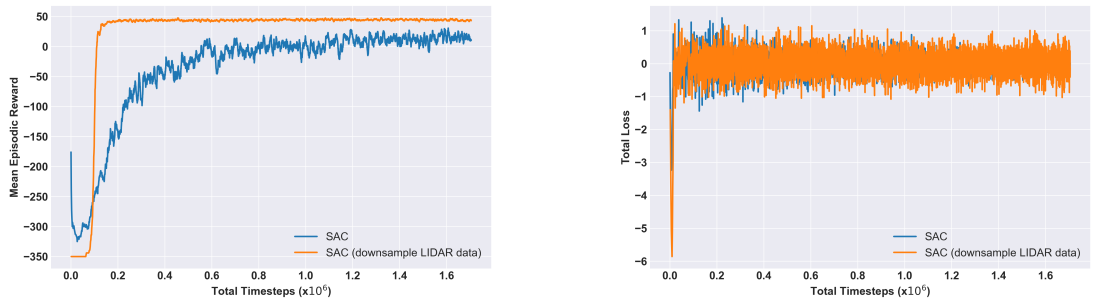
To conclude, while downsampled Lidar data allows for quicker adaptation and efficient learning, the full Lidar data configuration provides more stability and robustness in policy learning, making it preferable in environments where stability is critical. The choice between these configurations should be guided by the specific requirements of the task, balancing the need for rapid learning against the benefits of stable and consistent policy performance.

Test Environment Results and Analysis The agent trained using downsampled Lidar data with 10 beams shows a tendency to keep a safe distance from obstacles along its path, as depicted in Figure 4.6 (b). This cautious behavior helps prevent the agent from getting too close to obstacles, promoting safety even though it may result in longer paths. The downsampled Lidar data agent's trajectory appears erratic compared to the full Lidar data agent, indicating a less smooth navigation ex-

perience due to more frequent adjustments and deviations. Despite these deviations, the downsampled Lidar data agent manages to reach its destination by effectively avoiding obstacles and displaying goal-oriented behavior.

On the other hand, the agent equipped with complete Lidar data featuring 100 beams also adeptly avoids obstacles but does so with a smoother trajectory. The richer sensory information enables precise navigation decisions leading to paths that bring the agent nearer to obstacles, as displayed in Figure 4.6 (a), while maintaining safety. The trajectory of the Lidar data agent is noticeably smoother, with fewer disturbances, showcasing stable and efficient navigation. This agent also reaches its goal successfully, demonstrating an approach between obstacle avoidance and path efficiency.

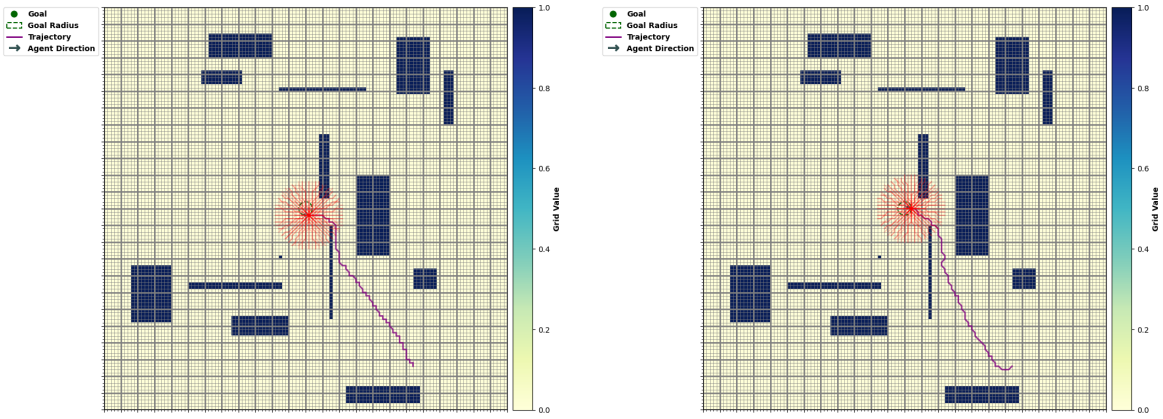
The comparison highlights that the downsampled Lidar data agent prioritizes safety by keeping a buffer from obstacles, which can be advantageous in dynamic or uncertain environments. However, this method leads to a disrupted and less effective route towards the objective. Conversely, the complete Lidar data agent enhances path efficiency by maneuvering closer to obstacles, demonstrating better accuracy with fewer deviations. Both agents manage to reach the goal, underscoring SACs' effectiveness in training resilient navigation strategies across different sensory inputs. The decision between these methods should account for operational needs while balancing safety, effectiveness, computation, and environmental navigational complexity.



(a) Mean episodic reward over time steps.

(b) Total loss over time steps.

Figure 4.5: Training curves for SAC algorithm with Lidar sensors having 100 beams. (a) Mean episodic reward comparison between downsampled and non-downsampled scenarios. (b) Total loss comparison between downsampled and non-downsampled scenarios.



(a) (b)

Figure 4.6: Trajectories of SAC trained agents tested in a random environment. (a) Trajectory with 100 beams in the Lidar sensor, (b) Trajectory with 100 beams downsampled to 10. In (b), lidar beams before downsampling have been shown.

4.3 Summary of key findings

The main findings of the study are summarized as follows:

- **Algorithm Performance in Obstacle-Free Environments**:
  - SAC: Demonstrated high stability and swift convergence, effectively adapting to the environment. This indicates SAC's suitability for dynamic navigation tasks.
  - PPO: Showed effective learning but exhibited slower convergence compared to SAC. Periodic spikes in loss, caused by the clipping mechanism, highlighted the need for precise hyperparameter tuning. Despite these fluctuations, PPO achieved stable navigation over time.
  - DWA: Navigated efficiently and steadily, demonstrating a direct and seamless path to the goal.
- **Algorithm Performance in Environments with Obstacles**:
  - SAC: Adapted well to obstacle-laden environments, successfully reaching the goal despite obstacles. The average reward per episode steadily increased, and overall loss remained consistent, showcasing SAC's capability to handle complex situations and navigate efficiently.

  - PPO: Encountered difficulties with obstacles, often colliding when alternative routes were available. This highlights the algorithm's sensitivity to hyperparameter configurations. Proper tuning of parameters like clipping range, learning rate, and value function loss coefficient is crucial for enhancing obstacle avoidance and navigation.
  - DWA: Initially managed navigation but paused in front of obstacles due to inadequate hyperparameter tuning. This emphasizes the importance of precise parameter adjustments for successful navigation.

- **Impact of Lidar Data Downsampling**: SAC maintained performance despite reducing Lidar beams from 100 to 10, demonstrating resilience to limited environmental information.

- **Hyperparameter Tuning Significance**:
  - SAC: Exhibited resilience to suboptimal conditions and reduced reliance on hyperparameter tuning, maintaining consistently strong performance across scenarios.
  - PPO and DWA: Showed significant reliance on accurate hyperparameter tuning. Poor tuning led to performance issues like collisions in PPO and premature stops in DWA, underscoring the need for meticulous adjustments to achieve optimal performance in complex environments.

Chapter 5

Conclusion

This final chapter serves as a comprehensive overview of the thesis, summarizing the research work. Certain limitations were encountered during the course of work, paving the way for future directions, which has been enumerated as potential future work.

5.1 Summary of research questions

The primary aim of this study was to examine the following questions in order to understand how Deep Reinforcement Learning (DRL) may be utilized for effective autonomous navigation:

1. **How can Deep Reinforcement Learning be effectively applied to achieve autonomous navigation in scenarios with limited environmental knowledge?**

   Answer: SAC demonstrated that deep reinforcement learning (DRL) can successfully explore and adapt in contexts with limited information by immediately stabilizing and obtaining substantial rewards. The algorithm's resilience and capacity to process limited sensory data were crucial to its successful implementation. Although the results were tested in simulation, the findings indicate significant potential for real-world applications where sensory data might be limited or incomplete.

2. **What is the impact of different reward functions on the learning performance of DRL algorithms in autonomous navigation tasks?**

Answer: The learning performance of reinforcement learning algorithms is significantly influenced by the design of reward functions, mainly when these functions prioritize obstacle avoidance and goal efficiency. This focus leads to improved navigation outcomes. Various reward function designs were initially experimented with, utilizing dense and sparse reward systems. Later, upon successfully training agents using Soft Actor-Critic (SAC) from the Stable Baselines3 library, a more simplified reward system was adopted. The study demonstrates that well-designed reward functions can substantially enhance the learning process and efficacy of deep reinforcement learning (DRL) algorithms.

3. **How do SAC, PPO, and DWA compare in terms of their ability to navigate environments of varying complexity?**

   Answer: SAC demonstrated superior performance compared to both PPO and DWA in terms of stability, convergence speed, and efficiency in obstacle-laden environments. The effectiveness of PPO relied on meticulous hyperparameter tweaking to prevent conflicts and provide optimal performance, whereas the specific parameter configurations greatly influenced DWA's performance. The constant performance of SAC in many situations has emphasized its outstanding flexibility and resilience.

5.2 Conclusions drawn from the research

The study provided several critical insights into the application and performance of DRL algorithms for autonomous navigation. Despite limited environmental knowledge, SAC demonstrated exceptional adaptability and stability, making it highly effective for autonomous navigation. PPO, while effective, required precise hyperparameter tuning to achieve optimal performance. Regarding stability, convergence speed, and overall navigation efficiency, SAC outperformed PPO and DWA. DWA's performance was susceptible to hyperparameter tuning, and PPO showed periodic instability due to its clipping mechanism. The contributions of this research to the field of autonomous navigation and DRL are multifaceted. The study enhanced how SAC and PPO can be leveraged for autonomous navigation, particularly in obstacle-laden environments. The research also demonstrated SAC's robustness in handling

limited sensory information, maintaining performance even with significant Lidar data downsampling. These findings underscore the significance of precise hyperparameter tuning for PPO and DWA, providing valuable insights for future algorithmic tuning and optimization efforts. To encapsulate, this research establishes a foundational framework for initiating the application of DRL in autonomous navigation.

5.3 Recommendations for further research

To build on the findings of this research, several prospective research directions are suggested as follows:

- **Extended Hyperparameter Studies**: Further studies on hyperparameter optimization for DRL algorithms could enhance their performance and stability in complex environments. Investigating different tuning strategies and their impacts can lead to more robust algorithms.
- **Exploration of Reward Structures**: Investigating alternative reward structures and their impact on DRL algorithm performance could provide deeper insights into optimizing learning outcomes. Understanding how different rewards influence behavior can help in designing better training regimes.
- **Real-World Testing**: Implementing the trained models in real-world scenarios, like dynamic obstacles, would validate the robustness and applicability of the algorithms beyond simulated environments. This step is crucial for transitioning from theoretical research to practical applications.
- **Multi-Sensor Integration**: Incorporating additional sensors and integrating data from multiple sources could improve navigation accuracy and decision-making in DRL-based autonomous systems. Exploring sensor fusion techniques could enhance the robustness and reliability of these systems.

Bibliography

[1] Metin Sitti. "Miniature soft robots—road to the clinic". In: *Nature Reviews Materials* 3.6 (2018), pp. 74–75.

[2] Jing Li et al. "Magnetically-driven medical robots: An analytical magnetic model for endoscopic capsules design". In: *Journal of Magnetism and Magnetic Materials* 452 (2018), pp. 278–287.

[3] Deepak Patil et al. "A survey on autonomous military service robot". In: *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*. IEEE. 2020, pp. 1–7.

[4] Abhijith Valsan et al. "Unmanned aerial vehicle for search and rescue mission". In: *2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184)*. IEEE. 2020, pp. 684–687.

[5] Joseph A Starek et al. "Spacecraft autonomy challenges for next-generation space missions". In: *Advances in Control System Technology for Aerospace Applications*. Springer, 2015, pp. 1–48.

[6] Avijit Banerjee et al. "Resiliency in Space Autonomy: a Review". In: *Current Robotics Reports* 4.1 (2023), pp. 1–12.

[7] Binbin Xie et al. "Research progress of autonomous navigation technology for multi-agricultural scenes". In: *Computers and Electronics in Agriculture* 211 (2023), p. 107963.

[8] Hassan Nehme et al. "Lidar-based structure tracking for agricultural robots: Application to autonomous navigation in vineyards". In: *Journal of Intelligent & Robotic Systems* 103.4 (2021), p. 61.

[9] Nils J Nilsson et al. *Shakey the robot*. Vol. 323. Sri International Menlo Park, California, 1984.

[10] Hans P Moravec. "The Stanford cart and the CMU rover". In: *Proceedings of the IEEE* 71.7 (1983), pp. 872–884.

[11] Jane Holland et al. "Service robots in the healthcare sector". In: *Robotics* 10.1 (2021), p. 47.

[12] Jose E Naranjo et al. "Automation kit for dual-mode military unmanned ground vehicle for surveillance missions". In: *IEEE Intelligent Transportation Systems Magazine* 12.4 (2018), pp. 125–137.

[13] Humberto Martínez-Barberá and David Herrero-Pérez. "Autonomous navigation of an automated guided vehicle in industrial environments". In: *Robotics and Computer-Integrated Manufacturing* 26.4 (2010), pp. 296–311.

[14] Fatima Khan et al. "Navigation algorithm for autonomous mobile robots in indoor environments". In: *2018 Advances in Science and Engineering Technology International Conferences (ASET)*. IEEE. 2018, pp. 1–6.

[15] Cuebong Wong et al. "Adaptive and intelligent navigation of autonomous planetary rovers—A survey". In: *2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. IEEE. 2017, pp. 237–244.

[16] Levin Gerdes et al. "Efficient autonomous navigation for planetary rovers with limited resources". In: *Journal of Field Robotics* 37.7 (2020), pp. 1153–1170.

[17] Luiz FP Oliveira, António P Moreira, and Manuel F Silva. "Advances in agriculture robotics: A state-of-the-art review and challenges ahead". In: *Robotics* 10.2 (2021), p. 52.

[18] Prasanth Kumar Duba et al. "Estimation of State for GPS-Denied Navigation of Autonomous Underwater Vehicles (AUVs) in Underwater Exploration". In: *2023 International Conference on Sustainable Technology and Engineering (i-COSTE)* (2023), pp. 1–7. URL: https://api.semanticscholar.org/CorpusID:269324953.

[19] Linus Casassa et al. "Co-Axial Helicopter for Autonomous Navigation and Exploration in GPS-denied indoor missions". In: 2011. URL: https://api.semanticscholar.org/CorpusID:174773151.

[20] Erke Shang et al. "A guide-line and key-point based a-star path planning algorithm for autonomous land vehicles". In: *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2020, pp. 1–7.

[21] Mayur Parulekar et al. "Automatic vehicle navigation using Dijkstra's Algorithm". In: *2013 International Conference on Advances in Technology and Engineering (ICATE)*. IEEE. 2013, pp. 1–5.

[22] Luis Garrote et al. "An RRT-based navigation approach for mobile robots and automated vehicles". In: *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*. IEEE. 2014, pp. 326–331.

[23] Duc-Kien Phung et al. "Model predictive control for autonomous navigation using embedded graphics processing unit". In: *IFAC-PapersOnLine* 50.1 (2017), pp. 11883–11888.

[24] Saleh Alarabi, Chaomin Luo, and Michael Santora. "A PRM approach to path planning with obstacle avoidance of an autonomous robot". In: *2022 8th International Conference on Automation, Robotics and Applications (ICARA)*. IEEE. 2022, pp. 76–80.

[25] Egidio D'Amato et al. "A visibility graph approach for path planning and real-time collision avoidance on maritime unmanned systems". In: *2021 International Workshop on Metrology for the Sea; Learning to Measure Sea Health Parameters (MetroSea)*. IEEE. 2021, pp. 400–405.

[26] Muhammad Mudassir Ejaz, Tong Boon Tang, and Cheng-Kai Lu. "Autonomous visual navigation using deep reinforcement learning: An overview". In: *2019 IEEE Student Conference on Research and Development (SCOReD)*. IEEE. 2019, pp. 294–299.

[27] Yang Tang et al. "Perception and navigation in autonomous systems in the era of learning: A survey". In: *IEEE Transactions on Neural Networks and Learning Systems* (2022).

[28] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. "An introduction to reinforcement learning". In: *The Biology and Technology of Intelligent Autonomous Agents* (1995), pp. 90–127.

[29] Chang-Shing Lee et al. "Human vs. computer go: Review and prospect [discussion forum]". In: *IEEE Computational intelligence magazine* 11.3 (2016), pp. 67–72.

[30] Smruti Amarjyoti. "Deep reinforcement learning for robotic manipulation-the state of the art". In: *arXiv preprint arXiv:1701.08878* (2017).

[31] Sarthak Bhagat et al. "Deep reinforcement learning for soft, flexible robots: Brief review with impending challenges". In: *Robotics* 8.1 (2019), p. 4.

[32] Lei Lei et al. "Deep reinforcement learning for autonomous internet of things: Model, applications and challenges". In: *IEEE Communications Surveys & Tutorials* 22.3 (2020), pp. 1722–1760.

[33] Tuomas Haarnoja et al. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.

[34] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017). URL: https://arxiv.org/pdf/1707.06347.

[35] Dieter Foxy Wolfram Burgardy Sebastian Thrunyz. "The dynamic window approach to collision avoidance". In: (1997).

[36] Jorge de Heuvel et al. "Subgoal-Driven Navigation in Dynamic Environments Using Attention-Based Deep Reinforcement Learning". In: *2023 21st International Conference on Advanced Robotics (ICAR)* (2023), pp. 79–85. URL: https://api.semanticscholar.org/CorpusID:257279775.

[37] E. W. Dijkstra. "A note on two problems in connexion with graphs". In: *Numer. Math.* 1.1 (1959), pp. 269–271. DOI: 10.1007/BF01386390.

[38] P. E. Hart, N. J. Nilsson, and B. Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107. DOI: 10.1109/TSSC.1968.300136.

[39] Anthony Stentz. *Optimal and Efficient Path Planning for Unknown and Dynamic Environments*. Technical Report CMU-RI-TR-94-02. Robotics Institute, Carnegie Mellon University, 1994. URL: https://www.ri.cmu.

edu/pub_files/pub3/stentz_anthony__tony__1994_2/stentz_anthony__tony__1994_2.pdf.

[40] Sven Koenig and Maxim Likhachev. "Incremental A*". In: *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic (NIPS'01)*. MIT Press, 2001, pp. 1539–1546. URL: https://proceedings.neurips.cc/paper/2001/file/7e1246a62a726fa4ec7693e925a7c46f-Paper.pdf.

[41] Sven Koenig and Maxim Likhachev. "Fast replanning for navigation in unknown terrain". In: *IEEE transactions on robotics* 21.3 (2005), pp. 354–363.

[42] Martin L Puterman. "Markov decision processes". In: *Handbooks in operations research and management science* 2 (1990), pp. 331–434.

[43] Oliviero Vouch et al. "On the adaptivity of unscented particle filter for gnss/ins tightly-integrated navigation unit in urban environment". In: *IEEE Access* 9 (2021), pp. 144157–144170.

[44] Xiaoji Niu et al. "IC-GVINS: A Robust, Real-Time, INS-Centric GNSS-Visual-Inertial Navigation System". In: *IEEE Robotics and Automation Letters* 8 (2023), pp. 216–223. URL: https://api.semanticscholar.org/CorpusID:260490531.

[45] Gunasekaran Raja et al. "PFIN: An efficient particle filter-based indoor navigation framework for UAVs". In: *IEEE Transactions on Vehicular Technology* 70.5 (2021), pp. 4984–4992.

[46] Wikipedia contributors. *Hardware-in-the-Loop*. https://es.wikipedia.org/wiki/Hardware-in-the-loop. Accessed: 2024-05-29. 2024.

[47] Simon J. Julier and Jeffrey K. Uhlmann. "Unscented filtering and nonlinear estimation". In: *Proceedings of the IEEE* 92 (2004), pp. 401–422. URL: https://api.semanticscholar.org/CorpusID:9614092.

[48] Karl Berntorp, Tru Hoang, and Stefano Di Cairano. "Motion planning of autonomous road vehicles by particle filtering". In: *IEEE transactions on intelligent vehicles* 4.2 (2019), pp. 197–210.

[49] Francisco Curado Teixeira et al. "Robust particle filter formulations with application to terrain-aided navigation". In: *International journal of Adaptive control and signal processing* 31.4 (2017), pp. 608–651.

[50] Junhong Xu et al. "Kernel-based diffusion approximated Markov decision processes for autonomous navigation and control on unstructured terrains". In: *The International Journal of Robotics Research* 0.0 (0), p. 02783649231225977. DOI: 10.1177/02783649231225977. eprint: https://doi.org/10.1177/02783649231225977. URL: https://doi.org/10.1177/02783649231225977.

[51] Alex Rutherford et al. "Motion Planning in Uncertain Environments with Rapidly-Exploring Random Markov Decision Processes". In: *2021 European Conference on Mobile Robots (ECMR)*. IEEE. 2021, pp. 1–6.

[52] Matthew Budd et al. "Markov decision processes with unknown state feature values for safe exploration using gaussian processes". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 7344–7350.

[53] Chengzhuo Ni and Mengdi Wang. "Maximum Likelihood Tensor Decomposition of Markov Decision Process". In: *2019 IEEE International Symposium on Information Theory (ISIT)* (2019), pp. 3062–3066. URL: https://api.semanticscholar.org/CorpusID:203566931.

[54] Francisco Melo and Plinio Moreno. "Socially reactive navigation models for mobile robots". In: *2022 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. IEEE. 2022, pp. 91–97.

[55] Diego F. Paez-Granados et al. "Pedestrian-Robot Interactions on Autonomous Crowd Navigation: Reactive Control Methods and Evaluation Metrics". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2022), pp. 149–156. URL: https://api.semanticscholar.org/CorpusID:251279835.

[56] Neşet Ünver Akmandor and Taşkin Padir. "A 3d reactive navigation algorithm for mobile robots by using tentacle-based sampling". In: *2020 Fourth IEEE International Conference on Robotic Computing (IRC)*. IEEE. 2020, pp. 9–16.

[57] Michael T. Ohradzansky et al. "Reactive Control and Metric-Topological Planning for Exploration". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 4073–4079. DOI: 10.1109/ICRA40945.2020.9197381.

[58] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. "Reinforcement learning: A survey". In: *Journal of artificial intelligence research* 4 (1996), pp. 237–285.

[59] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), pp. 529–533.

[60] Melrose Roderick, James MacGlashan, and Stefanie Tellex. "Implementing the deep q-network". In: *arXiv preprint arXiv:1711.07478* (2017).

[61] Alessandro Giusti et al. "A machine learning approach to visual perception of forest trails for mobile robots". In: *IEEE Robotics and Automation Letters* 1.2 (2015), pp. 661–667.

[62] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[63] Shirel Josef and Amir Degani. "Deep reinforcement learning for safe local planning of a ground vehicle in unknown rough terrain". In: *IEEE Robotics and Automation Letters* 5.4 (2020), pp. 6748–6755.

[64] Mark Towers et al. *Gymnasium*. Mar. 2023. DOI: 10.5281/zenodo.8127026. URL: https://zenodo.org/record/8127025 (visited on 07/08/2023).

[65] John Schulman et al. "Trust region policy optimization". In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[66] Richard S Sutton et al. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press, 1999. URL: https://proceedings.neurips.cc/paper_files/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

[67] Antonin Raffin et al. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

Appendix A

Source Code Samples

Source Code A.1 presents the main function utilized to implement the lidar data downsampling technique mentioned in Subsection 3.1.6 of Chapter 3.

```
def compute_lidar_downsample_params(num_beams, target_resolution):
    """
    Computes the parameters needed for downsampling a LiDAR point cloud.

    This function calculates the parameters such as the minimum and maximum
    range, the number of points, and the resolution for downsampling a LiDAR
    point cloud. The exact parameters calculated can vary depending on the
    specific requirements of the LiDAR sensor and the application.

    Returns:
        dict: A dictionary containing the calculated parameters. The exact
        keys in this dictionary can vary, but common keys might include
        'min_range', 'max_range', 'num_points', and 'resolution'.

    Raises:
        ValueError: If the LiDAR sensor specifications are not valid.
    """
    initial_resolution = 360.0 / num_beams
```

```
19      downsample_factor = int(np.ceil(target_resolution /
            initial_resolution))
20      print
21      if downsample_factor < 1:
22          raise ValueError("Target resolution must be greater than
                the initial resolution.")
23      lidar_feature_dim = (num_beams + downsample_factor - 1) //
            downsample_factor  # Use ceiling division to ensure the
            correct number of beams
24      return downsample_factor, lidar_feature_dim
25
26  def preprocess_lidar_data_by_resolution(lidar_observations,
        target_resolution=36.0, max_range=4.0):
27      """Preprocess LiDAR data by downsampling based on resolution
            and clipping the range.
28
29      Parameters:
30          lidar_observations (np.ndarray): Array of LiDAR distance
                measurements.
31          target_resolution (float): The target angular resolution
                after downsampling in degrees.
32          max_range (float): The maximum range to clip the distances.
33
34      Returns:
35          np.ndarray: Downsampled and clipped LiDAR observations.
36      """
37      num_beams = len(lidar_observations)
38      downsample_factor, lidar_feature_dim =
            compute_lidar_downsample_params(num_beams, target_resolution
            )
39
40      downsampled_observations = []
41
42      # Iterate over the lidar observations in segments based on the
            downsample factor
43      for i in range(0, num_beams, downsample_factor):
44          segment = lidar_observations[i:i + downsample_factor]
45
46          # Treat -1 as a very large distance for min-pooling
                purposes
47          valid_distances = [min(d if d != -1 else float('inf'),
```

```
            max_range) for d in segment]
48
49          # If all distances are -1, set the result to -1
50          if all(d == float('inf') for d in valid_distances):
51              min_distance = -1
52          else:
53              min_distance = min(valid_distances)
54
55          downsampled_observations.append(min_distance)
56
57      # Ensure the length of downsampled_observations matches
            lidar_feature_dim
58      if len(downsampled_observations) != lidar_feature_dim:
59          raise ValueError(f"Downsampled LiDAR data length {len(
                downsampled_observations)} does not match expected
                feature dimension {lidar_feature_dim}.")
60
61      return np.array(downsampled_observations)
```

Code A.1: Lidar Data Downsampling Functions

A.1 Additional Resources

The test results for the thesis work and the custom gymnasium environment source code can be found at the following GitHub repository:

Deep Reinforcement Learning for Robot Navigation

Random Obstacles Gymnasium Environment

List of Figures

List of Tables

List of Codes