UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingeniería

Desarrollo y evaluación de un entorno de simulación multi-robot para carreras utilizando técnicas de Deep Reinforcement Learning y métodos tradicionales

Mateo Nicolás Pozo Ruiz

Ingeniería en Ciencias de la Computación

Trabajo de fin de carrera presentado como requisito
para la obtención del título de
Ingeniero en Ciencias de la Computación

Quito, 19 de diciembre de 2024

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingeniería

HOJA DE CALIFICACIÓN
DE TRABAJO DE FIN DE CARRERA

Desarrollo y evaluación de un entorno de simulación multi-robot para carreras utilizando técnicas de Deep Reinforcement Learning y métodos tradicionales

Mateo Nicolás Pozo Ruiz

Daniel Riofrío, Ph.D.

Quito, 19 de diciembre de 2024

© DERECHOS DE AUTOR

Nombres y apellidos: Mateo Nicolás Pozo Ruiz

Código: 00320780

Cédula de identidad: 0450083670

Lugar y fecha: Quito, 19 de diciembre de 2024

ACLARACIÓN PARA PUBLICACIÓN

Nota: El presente trabajo, en su totalidad o cualquiera de sus partes, no debe ser considerado como una publicación, incluso a pesar de estar disponible sin restricciones a través de un repositorio institucional. Esta declaración se alinea con las prácticas y recomendaciones presentadas por el Committee on Publication Ethics COPE descritas por Barbour et al. (2017) Discussion document on best practice for issues around theses publishing, disponible en http://bit.ly/COPETheses.

UNPUBLISHED DOCUMENT

Note: The following capstone project is available through Universidad San Francisco de Quito USFQ institutional repository. Nonetheless, this project – in whole or in part – should not be considered a publication. This statement follows the recommendations presented by the Committee on Publication Ethics COPE described by Barbour et al. (2017) Discussion document on best practice for issues around theses publishing available on http://bit.ly/COPETheses.

RESUMEN

La navegación de vehículos autónomos en contextos competitivos y dinámicos, como las carreras de robots, requieren de adaptabilidad y decisiones en tiempo real. Los entornos de simulación actuales para competencias como F1-tenth, Formula SAE Driverless y AWS DeepRacer son útiles, pero a menudo son rígidos para probar técnicas novedosas. En este sentido, el proyecto actual supera estas limitaciones mediante el diseño y la evaluación de una simulación de carreras multi-robot en Gazebo. En particular, se examina el funcionamiento de agentes DRL, cuyo comportamiento es generado con el algoritmo SARL*, mientras que en paralelo se investigan agentes que emplean los algoritmos clásicos A* o Dijkstra combinados con Dynamic Window Approach (DWA). Este trabajo abarca el proceso de refinamiento, entrenamiento y prueba de los modelos DRL, junto con su aplicación en la plataforma robótica Turtlebot3, en el entorno de Gazebo utilizando ROS Melodic, como una fase introductoria para integrar sistemas más avanzados. En este sentido, se resalta la importancia del aprendizaje profundo por refuerzo (DRL) en relación con la navegación autónoma, haciendo énfasis en la capacidad de respuesta en un entorno cambiante. Además, el uso del algoritmo A* y Dijkstra con DWA proporcionan un baseline para la comparación con los enfoques basados en DRL.

Las métricas de rendimiento incluyeron el tiempo de navegación, el número de colisiones y la calidad de la trayectoria. Además, para la fase de entrenamiento, se evalúan la tasa de éxito y la recompensa acumulada. Estas medidas ayudan a evaluar los modelos en situaciones donde se requiere competir. Los resultados muestran que los enfoques basados en DRL superan a los métodos tradicionales en términos de tiempo de navegación y adaptabilidad, dado un sólido proceso de entrenamiento que garantiza la capacidad de generalización del modelo.

Palabras clave: Multirobot, Deep Reinforcement Learning (DRL), Competitivo, Carreras, SARL*, Gazebo

ABSTRACT

Navigation of autonomous vehicles in competitive and dynamic contexts, such as robot racing, requires real-time decision-making and adaptability. Present simulation environments for competitions like the F1-tenth, Formula SAE Driverless, and AWS DeepRacer are useful but are often rigid to use for testing novel techniques. In this regard, the current project overcomes these setbacks by designing and assessing a multi-robot racing simulation in Gazebo, where the robots move on their own with DRL models or using traditional navigation approaches. In particular, the operation of DRL agents, whose behavior is generated with an adapted Socially Attentive Reinforcement Learning algorithm (SARL*) is examined, while agents employing the classical A* or Dijkstra algorithms combined with the Dynamic Window Approach (DWA) are investigated in parallel. This work encompasses the process of refining, training and testing the DRL models, along with their application on the Turtlebot3 robotic platform placed in the Gazebo environment using ROS Melodic, as an introductory phase for integrating more advanced systems. In this regard, this work emphasizes the significance of deep reinforcement learning (DRL) with respect to robotic navigation by improving responsiveness in a changing environment. Furthermore, the use of the A* algorithm and Dijkstra with DWA provides a baseline for comparison against DRL approaches. Performance metrics included navigation time, the number of collisions, and trajectory quality. In addition, for the training phase, the success rate and accumulated reward are evaluated. These measures help to assess models in situations where they are required to compete. Results show that DRL-based approaches outperform traditional methods in terms of navigation time and adaptability, given a strong training process that ensures the model's generalization ability.

Key words: Multirobot, Deep Reinforcement Learning (DRL), Competitive, Racing, SARL*, Gazebo

TABLE OF CONTENTS

Introduction....10
Motivation and Problem Definition....10
State of the Art....12
Materials and Methods....18
Simulation and Framework....18
Gazebo Simulator....18
Robotic Operating System (ROS)....19
Navigation and Control Algorithms....19
Global Planners: A* and Dijkstra....20
Dynamic Window Approach....21
SARL* Algorithm....22
Training Environment....23
Reward Function Design....25
Evaluation Criteria....27
Training Phase....27
Simulation Phase....27
Experimental Setup and Hardware Configuration....28
Results and Discussion....30
Training....30
Simulation....33
Traditional Agents....33
Intelligent Agents....35
Comparative Analysis....38
Conclusions and Future Work....40
Aknowledment....41
Bibliography....42

TABLE INDEX

Table 1: Simulation environment parameters for training.....24
Table 2: Neural network training parameters 25
Table 3: Comparison of training results across three environments.....30
Table 4: Navigation times for robots using traditional methods.....34
Table 5: Navigation times for SARL* controlled robots.....36

FIGURE INDEX

Figure 1: The ROS navigation stack.....20
Figure 2: The Dynamic Window Approach Visualization 21
Figure 3: Scenarios used in the OpenAI Gym training environment.....24
Figure 4: Simulated racing circuit track within the Gazebo environment. 28
Figure 5: Success rate and reward across training episodes for different scenarios.....31
Figure 6: Experimental Results for Traditional Navigation Agents 34
Figure 7: Experimental Results for DRL Agents.....36

INTRODUCTION

Motivation and Problem Definition

The creation of mobile robotic systems has attracted considerable research interests in recent years. Applications such as delivery robots, automated guided vehicles for warehouses, service robots, and self-driving cars have become a reality and are shaping the future (Zhu & Tao, 2021). Furthermore, the emergence of autonomous racing competitions like the F1tenth require adaptable and efficient navigation techniques, making them ideal to test the latest algorithms for autonomous navigation. Among the latest advancements in autonomous vehicle research, deep reinforcement learning (DRL) approaches are promising because they are able automatically extract important features from the environment and learn appropriate navigation policies through trial and error (Li, 2023) (Li, et al., 2019).

Multi-robot racing tasks provide a greater challenge than simple navigation tasks. In such environments the robots not only are to arrive at the goal, but they must also do this while interacting with competing agents that share the goal. In addition, the classic collision avoidance problem is extended to a multi-robot system, and the decision process is complicated, because agents must be capable of predicting and reacting to unanticipated agents (Zhu & Tao, 2021). Competitive environments provide the opportunity to evaluate the generalization ability of the most recent navigation techniques, which may lead to safer and more stable navigation. In addition, the type of competitive environments offers a powerful mechanism to improve and adjust current DRL algorithms.

There is currently a lack of multi-robot simulation environments specifically for racing situations. This poses a significant learning curve for students and researchers who wish to race autonomous robots. By creating a flexible simulation platform, this work aims to make robotics

and autonomous racing more accessible. In addition, by completing the entire algorithm workflow within the simulation, it is possible to decrease the risk associated to real robots, obtain a high-fidelity simulation and increase the ease of transferring the validated models to real-world scenarios effortlessly.

STATE OF THE ART

Reinforcement learning (RL) has recently gained considerable attention as a powerful branch of machine learning, particularly for tasks where an agent must learn to optimize its behavior through interactions with the environment. Unlike supervised or unsupervised learning, RL enables agents to discover optimal actions by trial and error without the need of large datasets or labelling, much like humans learn from experience. This approach allows the agent to progressively refine its actions based on feedback, maximizing a numerical reward to achieve a specific goal (Sutton, et al., 2018). The recent advancements in deep learning and the improvement in computational power have expanded the capabilities of RL to tackle more complex problems, giving rise to Deep Reinforcement Learning (DRL) models. The DRL agents, for instance, have been trained to play Atari 2600 games with the only input of image, even beating human performance in many scenarios (Mnih, 2013).

Due to its ability to represent and experience learning, DRL possesses a potentially exciting future in fields such as mobile robotics. Autonomous navigation is a fundamental problem in mobile robotics, and includes several sub-problems including obstacle avoidance, path planning, localization, and mapping. One of the most frequent strategies to overcome the autonomous navigation task is to design systems composed of various algorithms as solutions for each subtask as pointed out in (Zhu & Tao, 2021). Yet, such approaches often rely on accurate sensor readings and every stage in the autonomous navigation pipeline will be contaminated with computational errors compounded over time to generate suboptimal and reactive navigation policies in a dynamic or uncertain environment (Feng, et al., 2021), (Zhu & Tao, 2021). By contrast, DRL based methods can learn superior policies that enhance the performance of agents, even in dense and adverse situations (Feng, et al., 2021).

While DRL-based approaches are promising solutions for autonomous driving, as pointed out by (Zhu & Tao, 2021), there are still challenges. These include the partial observation problem, where the agent has limited visibility of its environment, leading to the learning of suboptimal policies; the sparse reward problem, where reward functions may only be effective at the end of an episode, making the learning process inefficient; and poor generalization, as training typically occurs in simulated environments to address safety and time constraints, resulting in a simulation-to-reality gap (Bosello, et al., 2022). To address these issues, various solutions have been proposed, including techniques such as recurrent neural networks for better handling partial observability (Yuan, et al., 2019), reward shaping to tackle sparse rewards (Pico, et al., 2023), and domain randomization strategies to reduce the simulation-to-real gap (Balaji, et al., 2020).

A separate strategy for overcoming the constraints of DRL-based approaches is to design hybrid solutions. Stand-alone DRL approaches have limitations including high training time, the absence of long-term memory, and strong propensity to fall into local optima (Wang, et al., 2023), (Cimurs, et al., 2022). From the other side, classical approaches such as A or Dijkstra's algorithm provide an asymptotically efficient and invariant solution for navigation in static environments, but they are not flexible enough for navigation in dynamic environments. To address these limitations, hybrid frameworks combining the reliability of traditional methods with the adaptability of DRL approaches have been proposed.

An innovative autonomous robot navigation framework was proposed in (Wang, et al., 2023). This consisted of a global and a local planner. The global planner employed a conventional navigation strategy, and generated an environment map, calculating a globally optimal route. Afterwards, several landmarks are found in the global path. They will finally be used for the DRL local planner to come to the goal. This approach demonstrated higher

efficiency, safety, and robustness than conventional approaches and full-end-to-end DRL approaches.

Similarly, in (Li, et al., 2019) a hybrid approach called SARL* is introduced, integrating a dynamic local goal setting mechanism. This framework computes a global trajectory using Dijkstra's algorithm from an environment occupancy grid map. Waypoints are subsequently established along this trajectory and the dynamic local goal mechanism chooses the farthest waypoint within a determined radius from the agent. Achieving its goal is reached through following these waypoints. This mechanism enhances the generalization ability of the policy, decreasing its dependence on the individual distances learned in the process and enabling the agent to generalize to changing environments.

With the advancement of DRL, the complexity of robot systems demands rises. There is growing interest in possible solution-oriented alternatives for cooperative navigation and communication among intelligent agents in applications such as indoor navigation, multi-robot navigation, and social navigation (Zhu & Tao, 2021). Doing the same in such complex environments, however, presents various issues that have led to sophisticated, advanced DRL techniques and frameworks (Balaji, et al., 2020) to enable robots to build and adapt as they go through the environment, while maintaining safe and efficient navigation.

The distributed multi-robot collision avoidance approach based on the hybrid control scheme presented in (Fan, et al., 2020) is a new approach to distributed multi-robot collision avoidance in cluttered environments. In addition, policy-gradient algorithm, proximal policy optimization (PPO), can be applied to allow the robot to learn from and adapt its navigation policy autonomously, in real time without using intercommunication. Additionally, the centralized learning/decentralized execution framework is adopted, producing experiences

from all the robots that trained on the training set. The advantage of this method is that it does not require a central controller which is computationally infeasible for large-scale multi-robot systems and provides the robots a significant opportunity to cooperate efficiently and to not collide. Furthermore, the simulation-to-real error is shown to be reduced.

An alternative approach, the Cooperative Deep RL (CDRL) framework, is presented in (Kim, 2024). CDRL introduces a policy switching mechanism to balance exploration and exploitation in unknown environments. This framework allows robots to dynamically switch from safe exploration of the unknown to idea-driven exploration and deployment in goal-oriented navigation. Policy coupling increases the robot's capacity to navigate the complex, dynamic environment without the need for retraining and therefore leads to a flexible robot for varying environments.

As regards autonomous exploration, authors in (Cimurs, et al., 2022) proposed an autonomous approach based on DRL with a Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm to achieve goal-driven exploration in arbitrary environments. This approach learns and dynamically selects way points for surveying around interesting points (POI) and can efficiently exploit to survey unknown environments. In addition, experimental results demonstrate that the proposed scheme achieves the combined effect of reactive local and global navigation strategies.

Consistent with efforts to improve DRL-based autonomous navigation, the development and implementation of reward functions has generated considerable interest. In (Pico, et al., 2023), the performance of various reward functions for an obstacle avoidance model was evaluated. A new reward function was proposed, which compels the robot to stop in dangerous situations and highlights the importance of finding a balance between simplicity

and interpretability while designing the reward function. In a similar way, in (Zhou, et al., 2023) a novel collision avoidance algorithm known as Context-Aware Safe Reinforcement Learning (CASRL) was presented. One of the main implications of this algorithm was the introduction of two task-related policy gradients, fundamentally breaking the safety mechanisms from the navigation mechanisms. This method relieves the problem of task interference frequently experienced in the design of reward function and yields more satisfactory and efficient learning in the reinforcement of complex scenarios.

With the advancement of DRL approaches for autonomous navigation, their application in autonomous robot racing presents both unique challenges and opportunities. With the need for fast, real-time learning in very dynamic environments, the task of racing translates very well to a testing ground for state-of-the-art DRL algorithms. Autonomous racing requires not only precision in navigation but also the ability to adapt quickly to continuously changing scenarios, pushing DRL models to their limits. Notably, DRL approaches have been proposed for prestigious international racing events, such as the driverless category of Formula SAE (Salvaji, et al., 2022), the F1tenth competition (Bosello, et al., 2022), and the Amazon Web Services sponsored autonomous racing league with the DeepRacer platform (Balaji, et al., 2020), demonstrating the potential of DRL in continuously changing, complex environments.

Even though there have been large breakthroughs in the use of DRL in autonomous racing, current methods, such as those employed in the F1tenth Challenge (Bosello, et al., 2022) and Formula SAE (Salvaji, et al., 2022), these mainly rely on single-robot navigation in a simulator. This limitation restricts the possibility of a complete validation of the collision avoidance features of DRL models in a larger multi-robot scenario, which may involve more agents, and in which their interactions among the agents are important. While the AWS DeepRacer platform (Balaji, et al., 2020) provides multi-robot simulation environments, it is

limited to the DeepRacer ecosystem, and it will not be extendible to other competitive platforms such as F1tenth or Formula SAE. Thus, although current work shows encouraging results, there is a lack of a comprehensive experimental effort to tap into the full insights that DRL can provide in multi-robot autonomous racing tasks in a general, and yet generalizable, simulation environment.

MATERIALS AND METHODS

Simulation and Framework

Gazebo Simulator

Testing on physical hardware in robotics can be expensive, risky, and labor-intensive (Newans, 2024). Furthermore, from the viewpoint of hardware implementation, the related practical issues can hinder the development of fundamental algorithms, resulting in an inefficient process.

To address these problems, Gazebo was created as an open-source, free simulation platform for robots. Similar to game engines, Gazebo provides physically realistic physics simulations of good accuracy and is capable of providing a wide range of virtual sensors, as well as programmable interfaces for both users and system (Open Source Robotics, 2024).

Even though flexibility is a most prominent characteristic of Gazebo, there is another important part of the simulator that lies in its large collection of robot models, sensors, and prebuilt environments. This heterogeneity permits the researcher to filter out the specifics of simulated hardware in favor of the testing of robotic algorithms (Newans, 2024).

We can build a realistic, high fidelity simulation environment in Gazebo, which allows us to model a robust environment in which each robot sensor can consistently monitor its environment and communicate with its environment data seamlessly to other simulation

entities. This integration simplifies the testing of algorithms, allowing us to focus directly on algorithm development without being constrained by hardware limitations.

Robotic Operating System (ROS)

Interaction with robotic systems was for a long time a very inefficient task because of the complex hardware integration needed. Infrastructure management and design were frequently dull and repetitive work, which made robotics out of the reach of people without much hardware experience (Tellez, 2022).

The adoption of ROS (Robot Operating System) as an open-source platform greatly enhanced accessibility through easy to use and re-usable code for robotics applications by researchers (Canonical, 2024). ROS is based on a publisher-subscriber architecture using the ROS API for communication, which is structured by topics, services, action servers, and message formats to provide a standardized interface to hardware peripherals (Tellez, 2022). With this abstraction, developers are very much relieved of the low-level hardware details, thus it allows them to concentrate on the algorithm development more.

In this study, we exploit the Turtlebot platform in the simulation space. Turtlebot is uniquely tailored for ROS with the clear goal of teaching developers how to test and develop applications in a simple way, so that developers can concentrate on algorithms rather than getting trapped into hardware details.

Navigation and Control Algorithms

The techniques explored in this study utilize the ROS navigation stack and its modified versions, focusing on hybrid navigation approaches. These methods integrate global planners, employing algorithms such as Dijkstra or A*, with local planners that utilize either Deep

Reinforcement Learning or the Dynamic Window Approach. An illustration of the workflow followed by hybrid planners based on the ROS navigation stack is shown in Figure 1 (Gill, 2018).

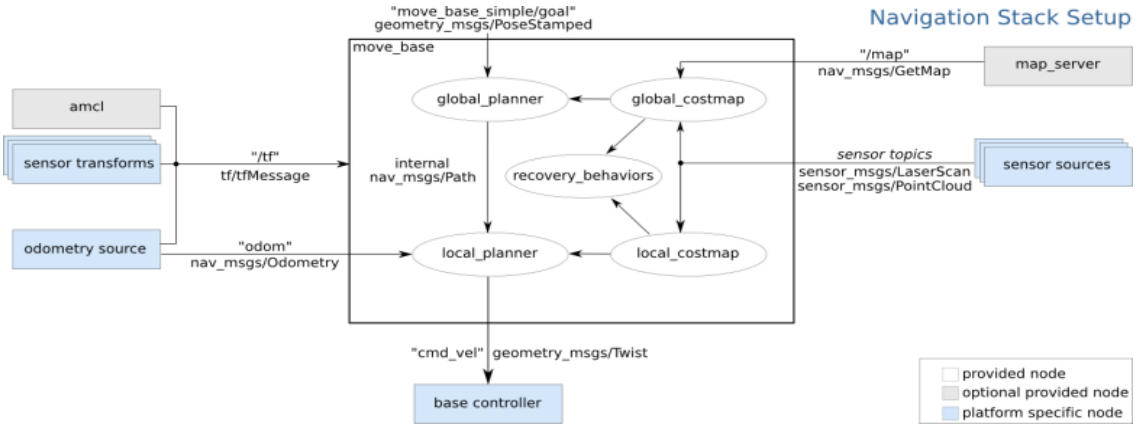


Figure 1: The ROS navigation stack

Global Planners: A* and Dijkstra

The ROS navigation stack used in this work provides several algorithms for global path planning. In this sense, the *move_base* package links a global planner that uses either A* or Djikstra, along with a local planner to accomplish the navigation task (Marder, 2020).

A* and Djikstra are both graph-based search algorithms that are designed for the discovery of shortest paths in a graph between two nodes. In the context of our problem, the graph is the cost map grid. A* is a directed, informed search, and nodes are judged by evaluating the sum of costs to move to a node by the sum of costs from that node to the goal (Russell & Norvig, 2010). Accordingly, the approach gives high weight to paths likely to speed one toward an objective expeditiously, consistently producing expeditious solutions. However, Djikstra's algorithm examines each of the possible paths, thus guaranteeing the shortest path cost even if it results in a less computationally efficient solution (Marder, 2020).

Both algorithms have been chosen as the global planner for this project, because of their robustness and ROS compatibility. A* provides an optimal path finding solution by computing

a heuristic, enabling the generation of a navigation plan in a relatively short time, which is useful for large environments. However, by progressively analyzing all nodes, Dijkstra guarantees the shortest possible path, which makes it suitable if an optimal solution is required. Collectively, these algorithms permit a flexible strategy that can be accommodated for the requirements of this work.

Dynamic Window Approach

The Dynamic Window Approach (DWA) is a local planner algorithm applied to robot navigation in real time. In contrast to global planners which find a global path from beginning

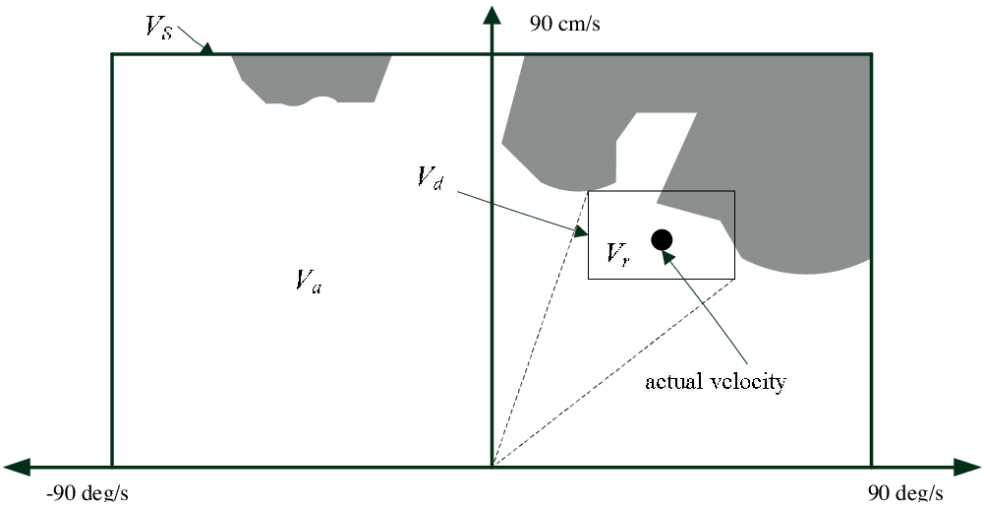


Figure 2: The Dynamic Window Approach Visualization

to end, DWA seeks to find feasible, intermediate trajectories to safely guide the robot through way to the aim while circumventing obstacles. DWA is based on the assessment of the possible robot trajectories in a limited time window, considering both dynamic and kinematic limitations as maximum velocity, acceleration and thus a maximum braking limit. This naturally leads to a dynamic window, in which the robots constrains its action space, as illustrated in Figure 2. DWA generates an optimal trajectory by, respectively, predicting future positions of the robot in this window, as well as distance to obstacles and distance to the

globally optimal trajectory, while constraining the trajectory to produce the final desired goal direction (Fox, et al., 1997).

Besides the ROS integration provided by DWA, to address the requirements of both the dynamic environment and real-time obstacle evasion, this algorithm was selected for this work, as it performs well in dynamic environments, and allows real-time obstacle avoidance; characteristics that are critical for multi-robot scenarios. Due to the algorithm's capacity to continuously adjust to the environment's changes, the robot can take fast decisions, which is suitable for competitive applications. Additionally, DWA integrates easily with the global planners mentioned before, as it can efficiently follow the global path generated by these algorithms while adjusting to obstacles along the way.

SARL* Algorithm

This project features agents using Deep Reinforcement Learning techniques. For this purpose, the SARL* algorithm was chosen as the base algorithm for experimentation. The SARL* (Socially Attentive Reinforcement Leaning*) algorithm enhances the original SARL model, originally designed to safely navigate in crowded environments (Li, et al., 2019). SARL* solves its previous implementation problems by including features designed for long-distance navigation and obstacle avoidance. Therefore, SARL* presents itself as an interesting alternative to experiment with dynamic environments.

Key elements of SARL* are an adaptive, local goal setting mechanism and a map-based safe action space. In addition, SARL* possesses a hybrid planning mechanism based on applying Djikstra's algorithm, which plans globally and gives rise to dynamic goals as the path is being traversed by the robot (Li, et al., 2019). This paradigm allows for adaptability to changing circumstances while assuring a steady trajectory towards the goal. Furthermore, the

safe action space mechanism represents future positions, and prevents actions that will probably result in collision.

Here, the algorithm SARL* is an interesting object of study because of its intrinsic capabilities. Its adaptive local goal setting mechanism and obstacle avoidance, enable SARL* to be used in the context of racing situations where robots need to be responsive and instantly take optimal decisions. Additionally, the studies around this algorithm have led to promising outcomes. For instance, as explained in (Pico, et al., 2023) various reward functions were explored, which resulted in superior performance for the same algorithm.

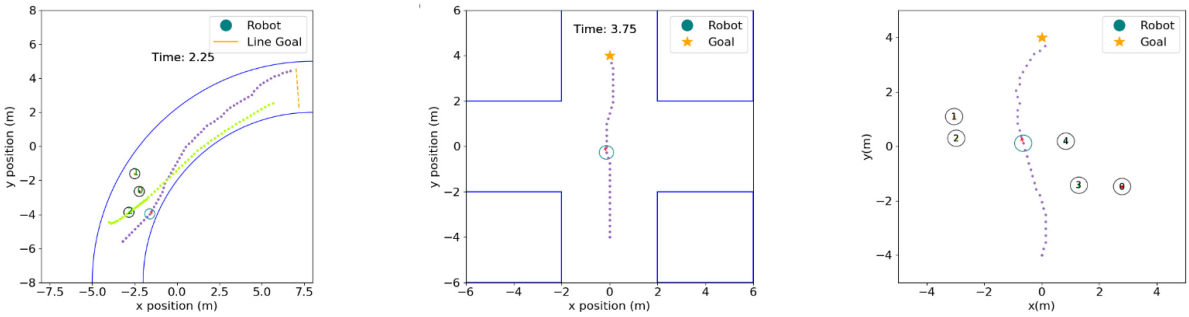
With the comparison between SARL* and conventional navigation approaches, the purpose of this project is to study the robustness and performance of SARL* and DRL in the actual racing conditions. The findings of these experiments will be used to establish the utility of SARL*'s abilities in changing competitive environments and to inform and develop future versions for multi-robot autonomous navigation systems.

Training Environment

For training our DRL models, the work in (Li, et al., 2019) was extended to model competitive situations. This methodology combines the OpenAI Gym library to simplify the development of a value-based deep reinforcement learning framework. The use of the OpenAI Gym library eases the development process, since it provides an API that includes necessary

building blocks like action spaces, observations, reward functions, and key environment data (Sonawane, 2023).

The training environment is designed to mostly represent the real-world scenarios the robot can encounter. Training scenarios for curve paths, straight paths, and intersections are implemented as shown in Figure 3. Also, the movement of other agents in the training environment is simulated with the ORCA algorithm because of its efficient path generation and simplicity. Each of the presented scenarios generates a model that will be further tested in the simulation phase.



(a) Curve path scenario (b) Cross path scenario (c) No-wall scenario

Figure 3: Scenarios used in the OpenAI Gym training environment

The simulation and robot parameters are shown in Table 1.

Parameter	**Value**
Number of Agents	3-5
Robot Radius	0.3 m
Agent Radius	0.3 m
Robot Velocity	1 m/s
Agent Velocity	1 m/s
Kinematics	Unicycle

Table 1: Simulation environment parameters for training

As for the training parameters, these include variables that are essential for the robot learning process, including the agent's policy, discount factor, learning rate, number of episodes, and other important parameters as outlined in Table 2.

Parameter	**Value**
Time Step	0.25 s
Test Size	3000
Agent's Policy	ORCA
Discount Factor	0.9
RL Learning Rate	0.001
Train Episodes	6000
Train Batches	100

Table 2: Neural network training parameters

Reward Function Design

The reward function is designed to drive the learning process of the DRL agents. It is a critical component as it directly influences the behavior of agents and their ability to navigate effectively. The reward design is focused on key objectives including:

- Encouraging the robot to reach the goal as quickly as possible.
- Penalizing collisions with obstacles to promote safety.
- Rewarding efficient path trajectories to minimize detours.
- Penalizing unnecessary stops, promoting continuous motion.

Considering the experiments regarding different reward functions investigated in (Pico, et al., 2023) a reward function using dense rewards was chosen. Therefore, the robot receives

a reward or penalization at each time step given a joint state J, and an action a, according to equation R1.

$$R1(J,a) = R_d + R_{pf} + R_g + R_c + R_{km} + R_{cw} + R_w + R_s + R_t$$

From the equation, R_d refers to the robot's proximity to the surrounding obstacles and penalizes the robot if it comes too close to other robots. R_{pf} encourages the robot to come closer to the goal at each step, giving it a reward when the distance to the goal is reduced. R_g refers to the robot reaching the goal, and it's greater than other rewards, since it represents the final purpose of the navigation task. R_c refers to collisions with other robots and penalizes the robot if a collision happens. R_{km} in a similar fashion to R_{pf} encourages the robot to keep moving, but without considering any direction. R_{cw} and R_w are related to the interaction with the limits of the track, penalizing the robot if it reaches a limit or if it comes too close to it. R_s is useful because it penalizes the robot if it stops, since the behavior of a competing agent is desired. Finally, R_t is a penalty in case the robot doesn't reach the goal in a certain time interval, encouraging responsiveness.

Also, to test the adaptability of the SARL* algorithm in simulation, the no-wall scenario shown in Figure 3 is trained using a conditional reward structure, instead of a sum of rewards as outlined in (Pico, et al., 2023). This approach results in better computational efficiency, as

only one reward is given per episode of training. In this case, the reward function is defined by equation R2.

$$R2(J,a) = \begin{cases} R_g & if\ Reach\ the\ goal, \\ R_t & elif\ Timeout, \\ R_c & elif\ Collision, \\ R_{ud} & elif\ Uncomfortable, \\ R_{hg} & otherwise \end{cases}$$

The new reward elements in this equation are R_{ud} that results in a penalty if the robot moves closer than $0.15m$ to an obstacle, and R_{hg} that refers to a reward when the robot reduces its initial distance to the goal.

Evaluation Criteria

To evaluate the simulation framework, we define distinct evaluation criteria for the two key stages of the workflow: the training phase and the simulation phase.

Training Phase

The evaluation is focused on success rate and accumulated reward during training. These metrics provide information about the learning process in terms of efficiency and performance.

Simulation Phase

The evaluation is conducted in the Gazebo environment, this stage evaluates the applicability and reliability of the trained models. The criteria include:

- Trajectory Quality: Assessed based on the smoothness, safety and efficiency of generated paths.

- Trajectory Completion Times: Measured to determine the speed of the models across different path lengths.

Experimental Setup and Hardware Configuration

To evaluate agent performance on the environment proposed in this project, a Gazebo-based environment was constructed, as described in Figure 4. This configuration simulates real-world racing track conditions, such as straight paths, acute turns, and roundabouts to validate the robustness and effectiveness of both DRL-based, and conventional navigation algorithms. However, due to interference between the AI planners and the traditional ROS navigation stack, separate tracks were used for AI-based agents and those using the ROS navigation stack. This approach prevents interference between planners and allows for a better experimentation process.

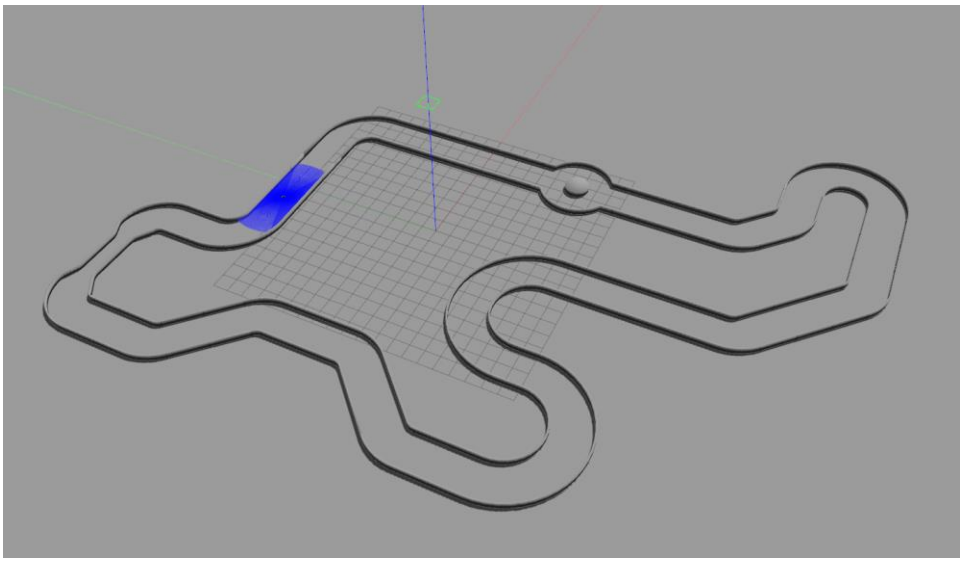


Figure 4: Simulated racing circuit track within the Gazebo environment.

In this environment agents are represented as TurtleBot 3 Burger robots due to their simplicity and ROS compatibility. The TurtleBot 3 platform provides necessary modules for

autonomous navigation, such as laser scanners to identify obstacles, and odometry sensors to provide real-time state information.

Furthermore, individual parameters of the configuration were optimized to respond to the changing requirements of a competitive environment. Goal tolerance was defined to $0.2m$ to ensure robots arrive at their targets, and global costmap inflation layer radius was defined to $0.25m$ in order that the best trajectories can be planned while keeping a safe distance from obstacles.

Additionally, the traditional ROS Navigation stack is implemented in the simulation to serve as a baseline for evaluating our DRL models.

RESULTS AND DISCUSSION

In this section, we first evaluate the results of the training scenarios outlined in Figure 3 using the metrics discussed in the previous section. We will then evaluate the performance of these models in a multi-robot context during the simulation phase.

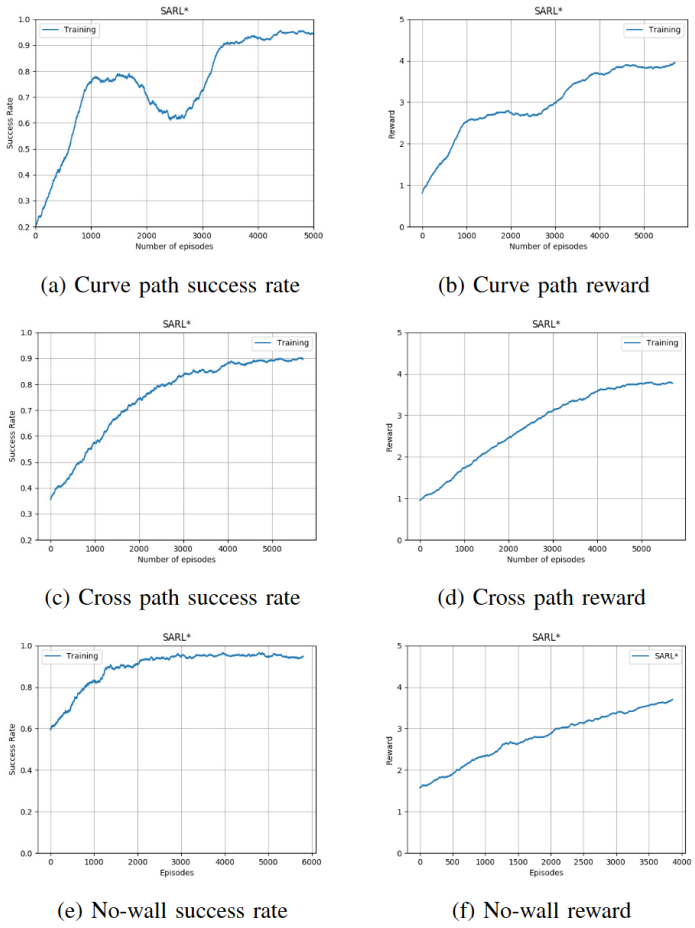
Training

Three scenarios were trained using the OpenAI Gym library for reinforcement learning as shown in Figure 3. The first two of them use a sum of rewards given by $R1$, and the third is a simpler scenario with no walls using a conditional reward approach as explained in $R2$. Table 3 provides an overview of the training results of these models.

Scenario	Success Rate	Collision Rate	Collision Wall Rate	Timeout Rate
Curve Path	0.98	0.00	0.01	0.01
Cross Path	0.94	0.04	0.01	0.01
No-wall	0.96	0.02	-	0.02

Table 3: Comparison of training results across three environments.

Also, the training plots showing the showing the behavior of the models across training episodes in terms of success rate and accumulated reward are shown in Figure 5.



(a) Curve path success rate (b) Curve path reward

(c) Cross path success rate (d) Cross path reward

(e) No-wall success rate (f) No-wall reward

Figure 5: Success rate and reward across training episodes for different scenarios

The training results for the curve path scenario show a progression through different learning phases as shown in Figure 5. Initially, the model shows a rapid improvement, with a success rate that rises from 0.2 to 0.75 within the first 1000 training episodes. However, there is a decline in the success rate at around 2000 episodes, but the model ends up recovering and achieves a final success rate of around 0.95. The reward curve shows a steady climb without major drops, indicating a reliable positive reinforcement over the entire training. The final

reward value of around 4.0 indicates optimized behavior, highlighting the model's overall adaptability and effective learning process despite temporary setbacks.

Similarly, in the cross-path scenario, a consistent learning trend is shown, as seen in Figure 5. Unlike the curve path scenario, the success rate shows a steady increase across all training episodes, reaching a stable value of around 0.9 at 4000 training episodes, and suggesting that the model adapts more smoothly to this scenario. The reward curve also shows a consistent rise, reaching a final value of around 4.0. The consistent behavior in both accumulated reward and success rate indicate that the model effectively learns to navigate in the cross-path scenario without encountering significant challenges during the training process.

In the case of the no-wall scenario, the training results show the highest success rate and the steepest accumulated reward among all training scenarios. The success rate value rapidly rises to over 0.8 at around 1000 training episodes and stabilizes at a value of over 0.9 at around 2000 training episodes. Also, the accumulated reward grows consistently through the training and achieves a final value of around 4.0. The behavior in both metrics is expected in this scenario, since it presents the simplest geometry for the robot. Also, these results highlight the simpler nature of the no-wall scenario, while the robot is still able to achieve its collision avoidance task as shown in Table 3.

Comparing the three scenarios, the no-wall scenario presents the most efficient learning process, whereas the curve path scenario is one with a more intricate learning with some

setbacks during the process. The cross-path scenario shows a consistent trend, with no significant problems, showing the model's adaptability.

Simulation

The simulation evaluation considers environments with AI-controlled agents and traditional agents separately. To deal with hardware limitations, both environments are tested with only three agents running simultaneously.

Due to their general properties, three specific situations are examined for both environments. The first consists of robots going in parallel along a straight path competing for reaching a goal line. The second consists of robots trying to take a curve successfully, potentially recurring to overtaking strategies. Finally, the third situation is a combination of the previous two mentioned before: the robots start in the same line, follow a straight path and take a curve to reach their goal. This setup allows us to precisely evaluate the behavior of the planners.

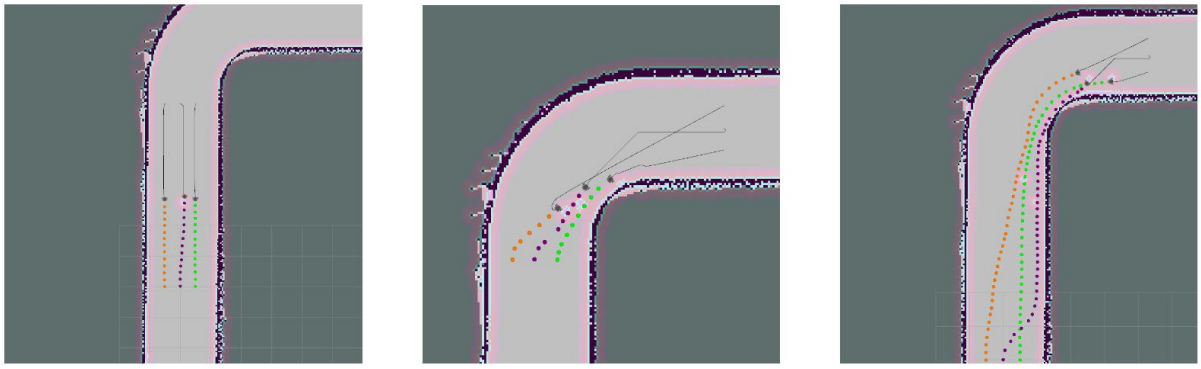
Traditional Agents

To test the behavior of agents in our multi-robot environment, we use the traditional ROS navigation stack. This approach uses Dijkstra/A* as the global planner, and Dynamic Window Approach as the local planner. In our setup, for comparison, we use two agents using Dijkstra and one agent using A* as the global planner. An overview of the results of the

experiments in the described environments is shown in Table 4. Also, the paths taken by the robots in the experiments are shown in Figure 6.

Robot	Straight Path	Curve Path	Combined Path
Robot 1 (Dijkstra)	28.01	23.39	52.12
Robot 2 (A*)	28.22	27.36	57.24
Robot 3 (Dijkstra)	28.40	34.13	55.76

Table 4: Navigation times, in seconds, for robots controlled by the ROS Navigation Stack in three different scenarios



(a) Straight path experiment (b) Curve path experiment (c) Complete path experiment

Figure 6: Experimental Results for Traditional Navigation Agents. Each subfigure displays the trajectory paths of three robots using traditional methods. Robot 1's path is shown in green, Robot 2's in purple, and Robot 3's in orange.

For the straight-path experiment, the three robots arrived at the goal within a short amount of time, showing the predictable nature of this scenario. Robot 1 first arrived at the goal in 28.01s, then Robot 2 with a delay of 0.18s, and Robot 3 with a delay of 0.39s with respect to Robot 1. These minor differences indicate that the robots experienced minimal interference from each other, thus highlighting the low complexity of a linear path.

In the curve-path experiment, differences between the planners start to show. As seen in Figure 6, the main differences come from the global planners. In the case of Robot 2, using A* algorithm, the path tends to be less predictable, and the robot's behavior is riskier. Also, there's a significant difference between the robot’s arrival times. Robot 1 arrives at the goal

first, in 23.39s, then Robot 2 arrives with a delay of 3.97s, finally Robot 3 arrives with a delay of 10.74s. Analyzing the behavior of the agents in the Figure, it's evident that Robot 2, controlled by A* generates riskier routes than the other planners. On the other hand, Robot 3 spends more time re-planning, resulting in a longer navigation time. Overall, this environment showcases the natural behavior of planners and outlines clear differences between A* and Dijkstra.

The complete-path experiment provides clear evidence of the performance trade-offs between the planners. In this case, Robot 1 arrived at the goal first at 52.12s, then Robot 3 at 55.76s, and finally Robot 2 at 57.24s.

As can be seen in Figure 6, the most interesting behavior arises from Robot 2 which employs the A* algorithm. This agent repeatedly tries to overtake other agents and maintains the shortest route from the race's start. It does, though, force a direct and hazardous collision, when the global planner devotes its attention to optimality rather than safety, when navigating a curve.

Intelligent Agents

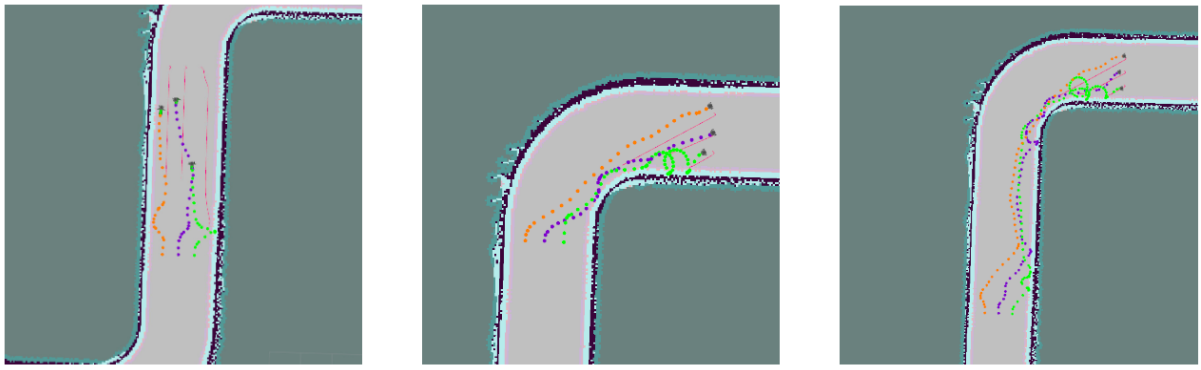
To test the trained DRL models in simulation, a modified version of the traditional ROS Navigation stack is used, extending the work outlined in (Li, et al., 2019). Following this, Dijkstra is used as the global planner, while SARL* is used as the local planner.

In this setup, three agents interact in the Gazebo simulation environment, each of them controlled by a specific trained model. Robot 1 is controlled by the curve path model, Robot 2 is controlled by the no-wall model, and Robot 3 is controlled by the cross-path model. An

overview of the results of the experiments in the three environments described is shown in Table 5. Also, the paths taken by the robots in the experiments are shown in Figure 7.

Robot	**Straight Path**	**Curve Path**	**Combined Path**
Robot	Timeout	44.19	74.63
Robot	22.26	26.29	61.52
Robot	26.71	25.46	48.77

Table 5: Navigation times, in seconds, for robots using SARL as the local planner in three different scenarios*



(a) Straight path experiment (b) Curve path experiment (c) Complete path experiment

Figure 7: Experimental Results for DRL Agents. Each subfigure illustrates the trajectory paths of three robots, with each path corresponding to a specific trained model. Robot 1's path is shown in green, Robot 2's in purple, and Robot 3's in orange.

For the straight-path experiment, the performance of the DRL agents varies significantly, reflecting the differences between trained models. Robot 2, controlled by the no-wall model, accomplishes the navigation task with an optimal behavior, arriving at the goal in just 22.26s. This shows this model's capability to navigate through straightforward, low complexity scenarios. Then, Robot 3, controlled by the cross-path model, reached the goal with a delay of 4.45s. On the other hand, Robot 1, controlled by the curve path model, failed to reach the goal, resulting in a timeout. This indicates a clear limitation of the curve path model, since

it is not generalizing to accomplish the navigation task outside its domain knowledge, even in a relatively simple linear path.

In the curve-path experiment, the results reveal interesting differences in how well each model adapts. Robot 3 reached the goal first, taking just 25.46s, which shows its impressive ability to handle a curve-path environment, even if this is not its specialty. Robot two arrived next, with a delay of 0.83s, showing that the no-wall model is able to tackle challenges different from what it was trained on. Finally, Robot 1 presents problems, arriving much later with a delay of 18.73s with respect to Robot 3, and with several collisions in its way. This suggests that, while the curve-path model is designed for this scenario, it might lack the generalization capability required to accomplish navigation tasks outside of its specific training environment.

The most notable differences and insights about the trained models are highlighted in the combined-path experiment. Robot 3 outperformed the others, completing the experiment in just 48.77s, demonstrating its generalization capabilities, and the strength of its training in the cross-path training scenario. Robot 2 arrived next, with a delay of 12.75s, completing the circuit in 61.52s, also showing its ability to generalize, although in a less efficient manner compared to Robot 3. Again, Robot 1 took the longest time to complete the experiment, arriving at the goal in 74.63s, and with several collisions along its way. In general, this

experiment highlights Robot 3's strong performance in different scenarios, while proving the poor generalization capabilities of the curve-path model, shown by its long completion time.

Comparative Analysis

Experiments conducted using traditional navigation methods along with DRL-based approaches showed some fundamental differences between their performance, adaptiveness, and reliability in multi-robot contexts.

In terms of performance, the conventional approaches to navigation exhibited a uniform and reliable behavior across all experiments. For example, in very simple scenarios, such as the straight-path experiment, all robots were able to accomplish the task, demonstrating very little variation in performance, showing how traditional methods would be optimal for environments of less complexity-needing predictable and safe paths.

On the other hand, DRL-based agents showed a more variable performance between scenarios, depending on their training. For instance, while the no-wall model performed well in the straight path scenario, the curve-path model was not able to reach the goal. This shows the importance of training DRL agents in scenarios designed taking the final task into account, as well as potential performance degradations when acting in scenarios outside of the agent's specialty.

When it comes to adaptability, traditional methods are notably less adaptable across scenarios, since the behavior of the global planner is closely followed. Therefore, in simulations, robots controlled by traditional methods fail to show competitive behavior,

presenting minimal differences between agents. Nonetheless, this conservative behavior is precisely what makes them able to avoid collisions and dangerous situations.

DRL agents, on the other hand, display stronger adaptability between scenarios, depending on the trained model. For instance, the cross-path model outperformed both traditional and DRL-based methods in all scenarios, despite not being a specialist in any of them. This showed that DRL-based methods, when trained in diverse environments, can generalize to unseen situations. Nonetheless, the poor performance displayed by the curve-path model suggests that very specific training scenarios lead to non-optimal behaviors when a good level of generalization is needed.

From the experiments, traditional navigation algorithms implemented by the traditional ROS navigation stack provide a reliable baseline for simple scenarios, presenting an optimal behavior and minimal differences in terms of speed and path quality. However, their performance is affected in more complex environments, where there is a need for adaptability and competitiveness. In contrast, DRL-based approaches show a superior behavior in terms of speed and adaptability in dynamic scenarios, as shown by the cross-path model driven agent across all the experiments. Nonetheless, the reliance of DRL-based methods on well-defined and generalizable training scenarios highlights the importance of a robust model design and validation.

Ultimately, while traditional navigation methods show robustness for predictable scenarios, DRL-based methods show promise in competitive tasks, where adaptability and a faster response time are required.

CONCLUSIONS AND FUTURE WORK

In this work, we present a multi-robot simulation environment designed to test traditional and DRL-based navigation methods in the context of robot racing. The SARL* algorithm was adapted to support competitive agents, and multiple training environments were developed in order to test each of the model's behavior in simulation. Experimental results show that DRL-based approaches can outperform traditional navigation methods in complex scenarios, given an appropriate training process to achieve model generalization. Furthermore, the feasibility of testing multiple agents in simulation is proven, presenting a safe and reliable alternative to test model generalization and validation before deploying in real robots.

Future work will include adapting the kinematics of real racing robots into the simulation. Additionally, a simulation environment where agents using traditional methods, and DRL-based agents interact is proposed to address the present compatibility problem between planners running in the same workspace.

AKNOWLEDMENT

The cooperation of Dr. Nabih Pico, and the SKKU RISE Lab in the development of algorithms and the ROS implementation along this project is appreciated.

BIBLIOGRAPHY

Balaji, B., Mallya, S., Genc, S., Gupta, S., Dirac, L., Khare, V., Roy, G., Sun, T., Tao, Y., Townsend, B., Calleja, E., Muralidhara, S. & Karuppasamy, D. (2020). Deepracer: Autonomous racing platform for experimentation with sim2real reinforcement learning. In *2020 IEEE international conference on robotics and automation (ICRA)* (pp. 2746-2754). IEEE.

Bosello, M., Tse, R., & Pau, G. (2022). Train in austria, race in montecarlo: Generalized rl for cross-track f1 tenth lidar-based races. In *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)* (pp. 290-298). IEEE.

Canonical. (2024). *What is ROS?*. https://ubuntu.com/robotics/what-is-ros

Cimurs, R., Suh, I. H., & Lee, J. H. (2021). Goal-driven autonomous exploration through deep reinforcement learning. *IEEE Robotics and Automation Letters*, *7*(2), 730-737.

Fan, T., Long, P., Liu, W., & Pan, J. (2020). Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios. *The International Journal of Robotics Research*, *39*(7), 856-892.

Feng, S., Sebastian, B., & Ben-Tzvi, P. (2021). A collision avoidance method based on deep reinforcement learning. *Robotics*, *10*(2), 73.

Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, *4*(1), 23-33.

Gill, J. (2018). *Setup and configuration of the navigation stack on a robot.* ROS.org. https://wiki.ros.org/navigation/Tutorials/RobotSetup

Kim, G. W. (2024). Cooperative deep reinforcement learning policies for autonomous navigation in complex environments. *IEEE Access*.

Li, H. (2023). Mobile robot navigation based on Deep Reinforcement Learning: A brief review. In *Journal of Physics: Conference Series* (Vol. 2649, No. 1, p. 012027). IOP Publishing.

Li, K., Xu, Y., Wang, J., & Meng, M. Q. H. (2019). SARL: Deep reinforcement learning based human-aware navigation for mobile robot in indoor environments. In *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)* (pp. 688-694). IEEE.

Marder, E. (2020). *Move_base.* ROS.org. https://wiki.ros.org/move_base?distro=noetic

Mnih, V. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Newans, J. (2024). *Simulating with gazebo: Articulated robotics.* Articulated Robotics. https://articulatedrobotics.xyz/tutorials/ready-for-ros/gazebo/

Open Source Robotics. (2014). *What is Gazebo?.* https://classic.gazebosim.org/tutorials?tut=guided_b1

Pico, N., Lee, J., Montero, E., Auh, E., Tadese, M., Jeon, J., ... & Moon, H. (2023). Enhancing Autonomous Robot Navigation based on Deep Reinforcement Learning: Comparative Analysis of Reward Functions in Diverse Environments. In 2023 23rd International Conference on Control, Automation and Systems (ICCAS) (pp. 1415-1420). IEEE.

Russell, S. & Norvig, P. (2010). Artificial Intelligence: A Modern Approach 3rd ed. *Prentice Hall.*

Salvaji, A., Taylor, H., Valencia, D., Gee, T., & Williams, H. (2023). Racing Towards Reinforcement Learning based control of an Autonomous Formula SAE Car. *arXiv preprint arXiv:2308.13088*.

Sonawane, B. (2023). *Getting started with OpenAI Gym.* BuiltIn. https://builtin.com/software-engineering-perspectives/openai-gym

Sutton, R. S., Bach, F., & Barto, A.G. (2018). Reinforcement learning: An introduction. *MIT Press Ltd*.

Tellez, R. (2022). *The beginners guide to ROS.* The Construct. https://www.theconstruct.ai/about-ros-robot-operating-system/

Wang, X., Sun, Y., Xie, Y., Bin, J., & Xiao, J. (2023). Deep reinforcement learning-aided autonomous navigation with landmark generators. *Frontiers in Neurorobotics*, *17*, 1200214.

Yuan, J., Wang, H., Lin, C., Liu, D., & Yu, D. (2019). A novel GRU-RNN network model for dynamic path planning of mobile robot. *IEEE Access*, *7*, 15140-15151.

Zhou, Z., Ren, J., Zeng, Z., Xiao, J., Zhang, X., Guo, X., Zhou, Z. & Lu, H. (2023). A safe reinforcement learning approach for autonomous navigation of mobile robots in dynamic environments. *CAAI Transactions on Intelligence Technology*.

Zhu, K., & Zhang, T. (2021). Deep reinforcement learning based mobile robot navigation: A review. *Tsinghua Science and Technology*, *26*(5), 674-691.