TRAINING A VIRTUAL REINFORCEMENT LEARNING AGENT FOR OBSTACLE AVOIDANCE AND TRANSFERRING IT TO A REAL MOBILE ROBOT

Author: Carrera Fresneda, Javier

Co-Director: Boal Martín-Larrauri, Jaime

Co-Director: Zamora Macho, Juan Luis

ABSTRACT

In a world with drones becoming a common sight in our skies and roads increasingly shared with autonomous vehicles, the quest for efficiency, safety, and innovation takes center stage.

The increasing demand for autonomous mobile robots (AMRs) across diverse sectors such as industrial automation, healthcare, and military has pushed the development of intelligent algorithms that enable these systems to navigate and execute tasks in varying environments without human intervention. Reinforcement learning can provide the backbone for these systems to traverse complex environments, recognize obstacles, detect anomalies, and adapt in real-time to varying situations.

However, the path to creating these systems is filled with challenges, primarily due to the complexities and risks of experimenting in real-world environments. This is where the value of simulation to reality (sim-to-real) becomes pronounced. Developing, testing, and refining autonomous algorithms in simulated environments presents numerous advantages, including accelerated development, cost efficiency, and safety. This thesis aims to explore the application of reinforcement learning to the development of autonomous navigation behaviors in a differential vehicle and a drone, addressing the simulation-to-real gap and providing insights into the challenges and complexities of real-world navigation.

I. INTRODUCTION AND CONTEXT

This section discusses the significance of reinforcement learning in addressing the challenges and complexities of real-world navigation. In the realm of mobile robotics, the unique challenges that reinforcement learning can tackle are highlighted, and its position relative to other techniques is assessed. By showcasing how each segment of a robotic system relates to reinforcement learning principles, this section sets the groundwork for the detailed discussions on methodologies and results in the following sections.

State of the Art

Autonomous mobile robots (AMRs) have been gaining rapid traction in diverse sectors due to the significant advantages they bring in terms of efficiency, precision, and safety. They leverage sensors, actuators, and intelligent algorithms to navigate and execute tasks in varying environments without human intervention.

Industrial Automation and Manufacturing

AMRs are used for material handling, picking, and sorting, increasing the speed and accuracy of production lines [1].

Healthcare

Patient Assistance: Robots in hospitals can help in transporting medicines, meals, or lab specimens [2].

Disinfection: Given the current global health scenario, robots equipped with UV lights or liquid disinfectants have been used in hospitals and public places to prevent the spread of contagious diseases [3].

Military and Defense

Reconnaissance: Robots can be used for surveillance purposes, scouting areas without putting human lives in danger [4].

Bomb Disposal: Specialized robots can be utilized to defuse or safely detonate explosive devices [5].

Other Sectors

Among the aforementioned sectors, AMRs provide versatile solutions in **warehouse and logistics** for inventory management [6] and material handling [7], **search and rescue** [8], **agriculture** for precision farming [9] and harvesting [10], **maintenance and inspection** [11] and **space exploration** [12].

Reinforcement learning provides the backbone for these systems to navigate complex environments, recognize obstacles, detect anomalies and adapt in real-time to varying situations. However, the path to creating these systems is filled with challenges, primarily due to the complexities and risks of experimenting in real-world environments.

This is where the value of simulation to reality (sim-to-real) [13] becomes pronounced. Developing, testing, and refining autonomous algorithms in simulated environments presents numerous advantages such as accelerated development, cost efficiency and safety.

General Scheme of Reinforcement Learning

Basic Components of the Process

Agent: This refers to the learner or decision-maker that interacts with the environment.

Environment: The environment is everything that the agent interacts with and learns from. It provides feedback to the agent based on the agent's actions.

Actions: Actions represent the set of all possible moves that the agent can make. These can be discrete, continuous, or a combination of both.

States: The state of an environment at any given time refers to the current configuration or situation of that environment.

Rewards: Rewards are feedback from the environment based on the agent's actions. A positive reward indicates a favorable action, whereas a negative reward indicates an unfavorable action. The goal of the agent is to maximize its cumulative reward over time.

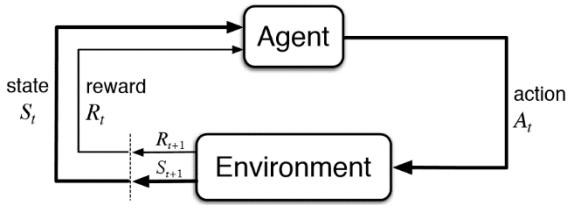


Figure 1: Reinforcement Learning General Scheme
Image Source: 2103.15781.pdf (arxiv.org)

Fundamental Concepts

Policy (π): A policy defines the agent's behavior. It is a mapping from states to actions, determining what action the agent should take in each state. The policy can be deterministic or stochastic. A deterministic policy provides a specific action for each state, while a stochastic policy provides a probability distribution over actions for each state. Neural networks can be used to approximate these deterministic policies.

Value Function: The value function approximates the expectation of cumulative future rewards a particular agent can expect to receive starting from a given state or state-action pair. In deep reinforcement learning deep neural networks are employed to represent and approximate this function due to their capability to generalize and handle large or continuous state spaces.

Episode: An episode refers to a sequence of states, actions, and rewards that ends in a terminal state or after a given number of steps.

Markov Decision Processes (MDPs): To apply reinforcement learning algorithms to an environment, it must satisfy the Markov property. According to this property, all subsequent states following a given state are dependent solely on that state and the action taken. Therefore, knowing the current state eliminates the need to consider the previous history of states to make an optimal decision.

Exploration vs. Exploitation: Exploration refers to the act of an agent trying out different actions, especially ones it is not certain about, in order to gather more information about its environment. Exploitation refers to the act of an agent selecting the action it believes will yield the highest reward based on its current knowledge. In continuous spaces, the balance between exploration and exploitation becomes crucial. Strategies like Epsilon-greedy , Softmax action selection or adding noise to the actions can help in efficient exploration.

Comparison and Synergy with Other Techniques

Traditional Control Algorithms: Traditional control algorithms like PID controllers [14] , LQRs (Linear Quadratic Regulators) [15], and MPC (Model Predictive Control) [16] have been standard tools in mobile robotics for years. These methods generally rely on mathematical models of the system and the environment. While they are robust and well-understood, they might not handle complex environments or unexpected scenarios as effectively as RL. However, the synergistic combination of these methods with RL can provide the reliability of traditional controllers with the adaptability of reinforcement learning. For example, a PID controller might maintain the stability of a drone, while the RL agent learns to navigate in complex environments.

Genetic Algorithms (GAs): GAs are optimization techniques inspired by the process of natural selection. In the context of mobile robotics, GAs might be employed to optimize certain parameters of an agent's operation. While GAs are excellent for optimization, RL focuses on learning through interaction. When combined, GAs might determine optimal hyperparameters for an RL agent, while the agent learns the best policy through interaction with the environment.

Project Objectives

Development of Autonomous Behaviors: To design and develop autonomous navigation behaviors for both a differential vehicle and a drone using reinforcement learning.

Simulation-to-Real Transition: To validate the trained models in a simulated environment and then successfully transition and adapt them for real-world deployment.

Integration of Technologies: Seamless integration of technologies such as Unity, Simulink, ROS2, Docker, and reinforcement learning frameworks to achieve a holistic system that can function in both simulated and real-world settings.

II. SIMULATION METHODOLOGY

In the realm of autonomous systems, simulation serves as the foundation upon which real-world implementations are built. This section delves into the technologies and tools employed to simulate and train the autonomous differential vehicle and drone. The deep reinforcement learning algorithm used, the Distributed

Distributional Deep Deterministic Policy Gradient (D4PG) is then discussed. The chapter then transitions into the modeling and implementation of the environments, distinguishing between the Unity and Python sides, and contrasting the simplified environment with the more realistic one implemented in MATLAB and Simulink.

Description of the Technologies

Simulation Software

Unity: Chosen for its versatility in 3D simulations, Unity provided the platform for sensor and actuator implementations. It offers an intuitive interface, advanced physics simulations, and real-time visualization.

Simulink: While Unity serves as a primary interface for visualization and high-level dynamics, Simulink complements it by offering realistic physics simulations, especially for the control aspect. Simulink provides a robust platform for implementing low-level controls seamlessly.

Technologies and Tools

ROS2: The Robot Operating System, version 2, facilitates the communication between different application modules, acting as a middleware. Given the complexity of autonomous systems where various modules and components need to communicate in real-time, ROS2 becomes an indispensable tool.

Docker: To ensure the developed application's reproducibility and compatibility across various platforms, Docker has been used for containerization. Docker encapsulates the application and its dependencies into a 'container' that can run uniformly on any machine, which is critical for collaborative projects and real-world deployment.

Reinforcement Learning Frameworks

PyTorch with low-level D4PG implementation [17]: PyTorch was the framework of choice due to its dynamic computational capacities and intuitive design which makes custom implementations, like D4PG, more straightforward and customizable. The D4PG (Distributed Distributional Deterministic Policy Gradients) algorithm, given its nature, fits well with complex tasks like autonomous navigation which requires both continuous action spaces and distributional value estimates.

Programming Languages

Python: Given its simplicity and wide array of support libraries, Python was used primarily for reinforcement learning, data analysis, and general scripting.

C#: As Unity's primary scripting language, C# was employed for implementing the logic within the 3D simulation environment.

MATLAB and Simulink: Primarily for control system design and low-level dynamics simulations.

System Description

Differential Vehicle System

The differential vehicle system follows a conventional architecture. As seen from the simulation perspective, it consists of two primary elements:

Mathematical Model: The mathematical model replicates the physical design of the vehicle, which features two motorized wheels at the back that deliver the required torque for movement. Additionally, a stabilizing wheel at the front ensures three contact points, and therefore stability.

Control Mechanism: The vehicle's motion is governed by two conventional PID controllers, one for linear velocity (with a nominal velocity of 0.4 m/s) and the other for angular velocity (with a nominal value of π rad/s). This dual PID control structure facilitates precise control over both speed and direction.

Actuators: The actuators for the differential vehicle system are the two motorized wheels located at the back. By individually controlling the speed and direction of each motorized wheel, the differential vehicle can achieve both forward motion and turning actions.

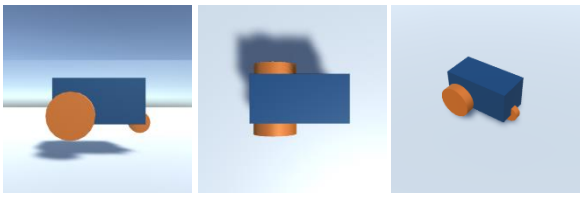


Figure 2: Differential Vehicle Simulation Model

Drone System

The simulation of the drone follows the design of a quadrotor in a '+' structure. The primary components of the drone are:

Mathematical Model: The mathematical model of the drone is carefully designed to emulate a quadrotor's dynamics, with smooth movements ensuring that pitch and roll angles remain close to zero. This simplification facilitates ease of control by considering the system as holonomic, allowing for more intuitive control over the vehicle.

Control Mechanism: The control system for the drone is more intricate, cascading three distinct control loops:

- **Angle Control Loop:** A low-level control loop responsible for maintaining the drone's orientation.
- **Linear Velocity Control Loop:** This loop controls the linear velocities of the drone, allowing for controlled movements in various directions.
- **Position Control Loop:** The top-level control loop utilizes an LQR (Linear Quadratic Regulator) architecture to manage the three-dimensional positioning of the drone.

Actuators: For the quadrotor drone system, the actuators are its four rotors. Each rotor is powered by a dedicated motor that provides thrust. Two diagonally opposite rotors spin in one direction and the other two in the opposite direction. The collective thrust from all rotors allows the drone to ascend or descend, while the differential thrust between them enables controlled movements in pitch, roll, and yaw.

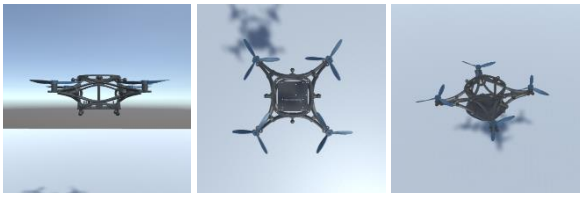


Figure 3: Drone Simulation Model

Sensors

Motion Capture Cameras: These cameras are utilized to obtain accurate positioning data, including both position and orientation. The cameras track specific markers or features within the environment.

Inertial Measurement Unit (IMU): The IMU integrates accelerometers and gyroscopes to measure the vehicle's specific force and angular rate. This aids in stability and navigation.

LiDAR: LiDAR sensors emit laser beams to measure distances to objects within their range. In the context of this project, a 2D rotatory LiDAR is used for both the vehicle and drone, providing a two-dimensional "slice" of the surrounding environment. This technology is vital for obstacle detection and avoidance.

Definition of the Deep Reinforcement Learning Algorithm

The complexity of continuous action spaces and the need for stable convergence require the use of advanced reinforcement learning algorithms. The Deep Deterministic Policy Gradient (DDPG) [18] is designed explicitly for environments with continuous action spaces, making it suitable for real-world problems like these autonomous systems. However, while DDPG offers a robust solution, advancements in the field have led to the development of an even more refined algorithm: the Distributed Distributional Deep Deterministic

Policy Gradient (D4PG) [19]. D4PG not only inherits the strengths of DDPG but also introduces several enhancements that make it particularly well-suited for this project's requirements. The following sections will present the details of DDPG and subsequently explore the improvements introduced by D4PG.

Deep Deterministic Policy Gradient (DDPG)

Actor-Critic Architecture: The actor defines the policy of the agent and therefore is responsible for determining the best action given a particular state. The actor takes the current state as input and outputs a continuous action or a set of continuous actions in multi-dimensional action spaces. The critic evaluates the action taken by the actor based on the current state and provides a value estimate (Q-value). This value estimate is used to update both the actor and the critic.

Off-Policy Learning: DDPG learns from past experiences stored in a replay buffer. This buffer stores tuples of experiences (state, action, reward, next state). During training, random samples are drawn from this buffer to update the networks, decoupling the correlation between consecutive experiences and stabilizing the learning process.

Deterministic Policy Gradient: Unlike traditional policy gradient methods that work with stochastic policies, DDPG uses a deterministic policy gradient. This means that for any given state, the actor outputs a specific action without any randomness.

Exploration vs. Exploitation: Since DDPG is designed for continuous action spaces, traditional exploration methods like epsilon-greedy can't be applied. Instead, DDPG adds noise to the policy, typically Ornstein-Uhlenbeck noise is used due to its temporally correlated nature.

Target Networks and Soft Updates: DDPG employs the concept of target networks, borrowed from DQN, to stabilize learning. There are target versions of both the actor and critic networks. Instead of copying the weights directly from the main networks to the target networks, DDPG uses "soft updates." This means the target network weights are a blend of the main network weights and their own, ensuring smooth transitions and further stabilizing the learning.

Distributed Distributional Deep Deterministic Policy Gradient (D4PG)

Distributed Experience Gathering: D4PG modifies the standard training procedure to distribute the process of gathering experience. Multiple actors operate in parallel, all contributing to a centralized replay table. A learner process then samples from this replay table, ensuring efficient and diverse data for network updates.

Distributional Perspective: D4PG adopts the distributional perspective on reinforcement learning, a paradigm shift from the traditional approach of estimating the expected value of future rewards to modeling the entire distribution of these rewards. This means that for a given state-action pair, instead of a single value estimate, a distribution over all possible returns is maintained. By modeling the full distribution, this technique captures the inherent randomness and uncertainty in the environment's dynamics.

Modeling of the Environments

The modeling of the environments is divided into two implementations: a simplified environment and a realistic environment. The simplified environment serves as a proof of concept, allowing for quick testing and validation of the fundamental ideas. It is designed with lower computational demands, enabling parallelization. The realistic environment builds upon the simplified environment, incorporating more complex dynamics and control systems to simulate a more realistic scenario. It integrates with an existing system [20] implemented in Matlab and Simulink, providing a more accurate representation of the vehicle's dynamics and control systems.

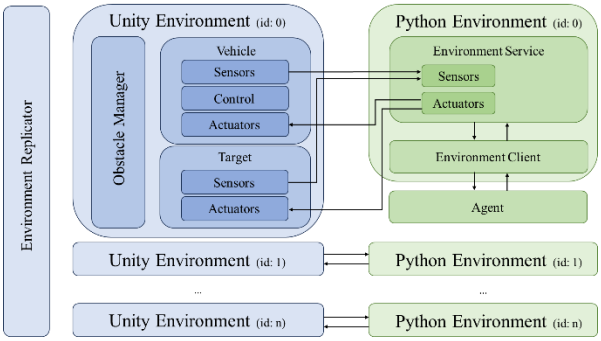


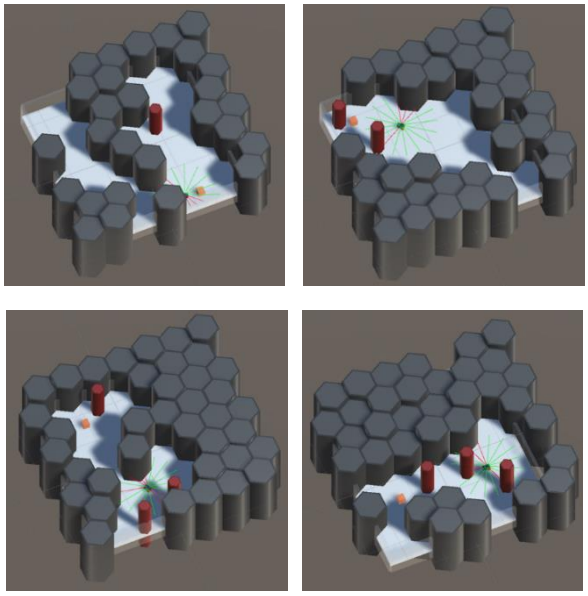
Figure 4: System Implementation Block Diagram

Environment and Environment Replicator Classes: The EnvironmentReplicator class is responsible for creating multiple parallelized instances of a given environment. The environment acts as a container GameObject. Within this container, other GameObjects reside, which allocate the scripts responsible for sensors, actuators, and controls. When the EnvironmentReplicator creates replicas of the environment, it inherently replicates all the contained logic, ensuring that each environment operates independently and in parallel.

Sensor and Actuator Classes: The Sensor and Actuator classes represent generic sensor and actuator components within the Unity environment. They are designed to be versatile, allowing for the creation of various sensor and actuator types by extending this base class. The sensor communicates with ROS to publish its data while the actuator communicates with ROS to receive actuation commands.

Control Class: The Control class serves as a base class for all controls within the Unity environment. It is responsible for managing the communication with the Robot Operating System (ROS) to receive control references and reset commands.

Obstacle Manager Class: The ObstacleManager class is designed to manage and generate obstacles within a simulated environment. The primary objective of this class is to ensure that the agent does not overfit to a specific environment configuration. By introducing variability and randomness in the environment, especially in the placement and type of obstacles, the agent is encouraged to learn more generalized strategies that are robust to changes.



Simulation Environment Visualization

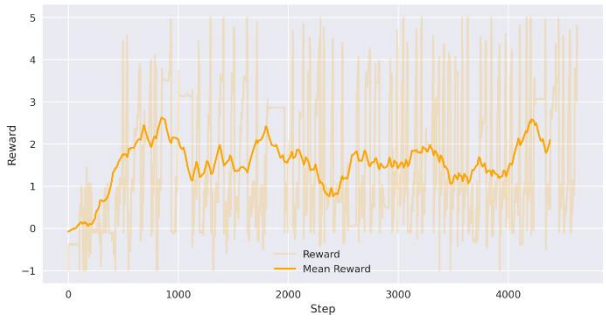
V. SIMULATION RESULTS ANALYSIS

This chapter offers a comprehensive analysis of the simulation results for two agents, by examining the evolution of rewards, the convergence of loss functions, and the overall success rates in different environments. The chapter also includes an analysis of the success rates in agent deployment in both simplified and realistic environments.

Regarding the following presented graphs, it is essential to clarify the distinction between the steps in the reward function and the steps in the loss function. The steps in the reward function represent episode steps for the exploitation worker, while the steps in the loss function represent learning steps. These two metrics are not directly comparable and have no established conversion ratio.

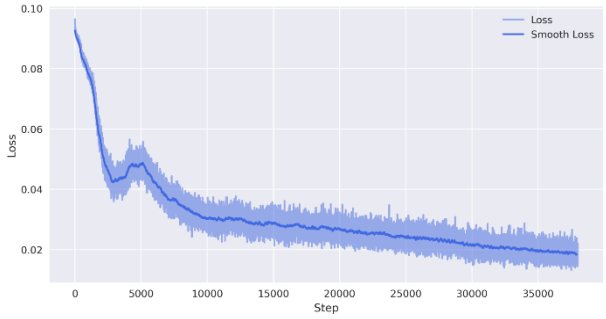
Rewards Evaluation and Policy Validation

Differential Vehicle



Differential Vehicle's Mean Reward per Episode Step

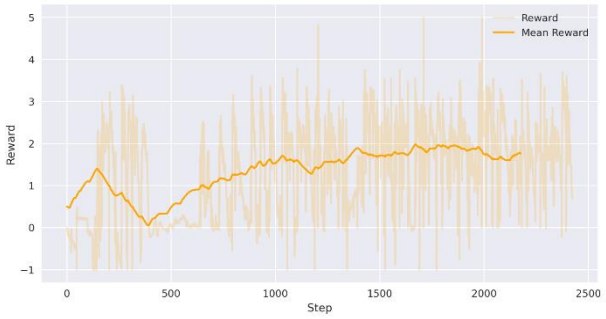
The reward evolution graph for the differential vehicle provides a visual representation of the agent's learning progress over time. As the agent interacts with the environment and learns from its experiences, the reward value increases. By 500 steps, the reward reaches a value of 2.0, suggesting that the agent has quickly adapted to the environment and is making decisions that align with the desired outcomes.



Differential Vehicle's Loss per Training Step

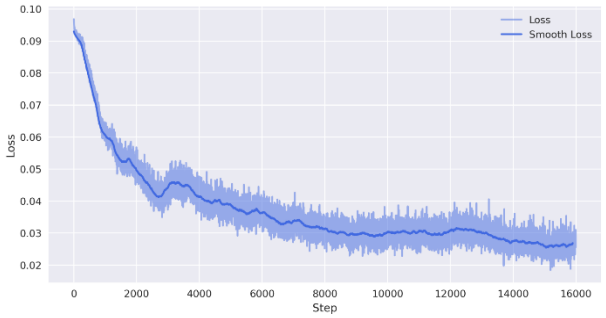
The loss function graph for the differential vehicle provides insights into the convergence of the D4PG algorithm. As the agent learns and updates its policy, the loss value decreases, showing a downward trend. This suggests that the agent's predictions are becoming more aligned with the target Q-values over time.

Drone



Drone's Mean Reward per Episode Step

The reward evolution graph for the drone, similar to the differential vehicle, starts at 0.0, indicating that the agent begins with no prior knowledge of the environment. Despite a sudden drop that could be attributed to the agent exploring a new strategy that did not yield favorable results, the reward graph shows an upward trend, indicating that the agent is gradually improving its performance and learning to make better decisions.



Drone's Loss per Training Step

The drone's loss function graph shows a general downward trend, indicating the convergence of the D4PG algorithm. This trend suggests that as the drone continues its interactions with the environment, its predictions of Q-values become increasingly accurate, aligning more closely with the target Q-values.

Success Rates in Agent Deployment

In this section, the success rates of the differential vehicle and the drone in both the simplified and realistic environments for a total of 120 episodes in each environment for each vehicle are analyzed. The success rates are evaluated based on four criteria:

- **Target Reached:** The agent successfully navigated to the target without any collisions.
- **Collision with Static Obstacle:** The agent collided with a static obstacle.
- **Collision with Moving Obstacle:** The agent collided with a moving obstacle.
- **Maximum Time Reached:** The agent took the maximum allowed time for an episode without reaching the target.

In the simplified environment, the differential vehicle achieved a success rate of 83.33%, which slightly decreased to 78.33% in the realistic environment. Collisions with static and moving obstacles were relatively low in the simplified environment but saw a slight increase in the realistic setting.

Differential Vehicle's Success Rates in the Simplified Env.

Criteria	**Epis.**	**%**
Target Reached	100	83.33
Static Obstacle Collision	2	1.67
Moving Obstacle Collision	6	5.00
Maximum Time Reached	12	10.00

Differential Vehicle's Success Rates in the Realistic Env.

Criteria	**Epis.**	**%**
Target Reached	94	78.33
Static Obstacle Collision	4	3.33
Moving Obstacle Collision	10	8.33
Maximum Time Reached	12	10.00

The drone exhibited a higher success rate in the simplified environment, reaching the target in 90% of the episodes. However, this rate dropped to 76.67% in the realistic environment. Notably, the drone experienced a more significant increase in collisions with moving obstacles in the realistic environment compared to the differential vehicle.

Drone's Success Rates in the Simplified Env.

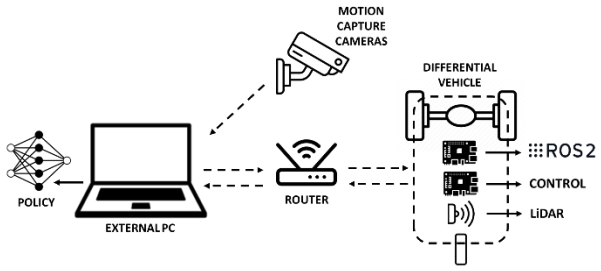
Criteria	**Epis.**	**%**
Target Reached	108	90.00
Static Obstacle Collision	2	1.67
Moving Obstacle Collision	4	3.33
Maximum Time Reached	6	5.00

Drone's Success Rates in the Realistic Env.

Criteria	**Epis.**	**%**
Target Reached	92	76.67
Static Obstacle Collision	6	5.00
Moving Obstacle Collision	14	11.67
Maximum Time Reached	8	6.67

IV. REAL-WORLD METHODOLOGY AND EXPERIMENTAL RESULTS

In the real-world implementation, the differential vehicle is equipped with two Raspberry Pi units, one dedicated to the control system and the other for communication purposes, interfacing with ROS2. The vehicle's perception of its environment is enabled by the LiDAR sensor, which provides a two-dimensional "slice" of the surrounding environment, essential for obstacle detection and avoidance. Additionally, external motion capture cameras are employed for accurate pose estimation. The control systems manage the vehicle's linear and angular velocities.



Real-World Environment Setup

Real-World Experimental Results

In the real-world experimental phase, the vehicle's performance was evaluated based on its ability to navigate autonomously, avoid obstacles, and reach the target destination within a specified time frame. The results of the real-world experiments revealed a success rate of 73.33% in reaching the target without collisions. The vehicle's trajectories were recorded and analyzed, revealing smooth and direct paths towards the destination in successful episodes. However, in episodes where collisions occurred, the trajectories indicated the vehicle's inability to navigate around obstacles effectively. The observed discrepancies can be attributed to several factors, including differences in obstacle geometry, vehicle geometry and dimensions, sensor noise, and the inherent complexities of the real world.

Success Rates in Agent Deployment

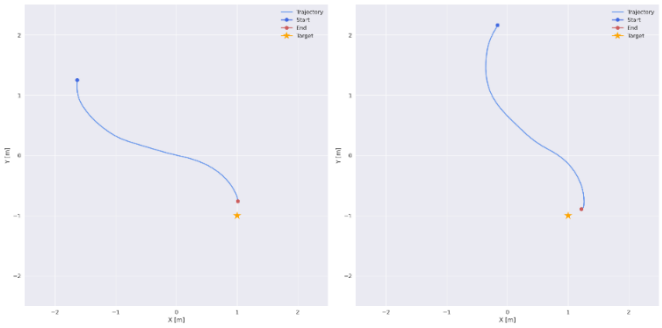
The differential vehicle was subjected to a total of 15 episodes within the real-world environment and successfully navigated to the target without any collisions in 11 episodes, translating to a success rate of 73.33%.

Differential Vehicle's Success Rates in the Real Env.

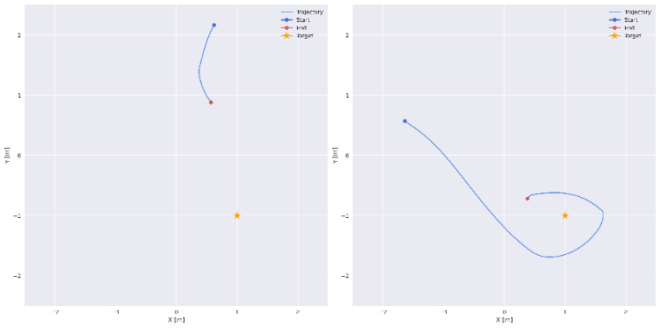
Criteria	Epis.	%
Target Reached	11	73.33
Collision with Obstacle	3	20.00
Maximum Time Reached	1	6.67

Analysis of the Trajectories

Throughout the testing phase, the vehicle's trajectory was recorded. These trajectories were plotted on an xy plane for the most representative episodes.



Episode Trajectories Where Target Was Reached



Episode Trajectory Where Collision Occurred

Episode Trajectory Where Maximum Time Was Reached

V. CONCLUSIONS AND FUTURE WORK

This project successfully demonstrated the feasibility of using reinforcement learning for the development of autonomous navigation behaviors in both a differential vehicle and a drone. The project also addressed the simulation-to-real gap, with the differential vehicle successfully transitioning to the real-world environment. The integration of various technologies, including Unity, Simulink, ROS2, Docker, and reinforcement learning frameworks, facilitated the development and simulation phases and ensured reproducibility and compatibility across platforms.

Future Work

Integration of Additional Sensors: Incorporating cameras as additional sensors could provide richer sensory data, enabling the agents to better perceive their environment. This would imply the use of Convolutional Neural Networks (CNNs) to process the image data and extract relevant features.

Advanced Neural Network Architectures: Introducing Long Short-Term Memory (LSTM) or transformer architectures for the actor and critic networks could enhance the agents' ability to learn and remember temporal dependencies in the environment, potentially improving decision-making in dynamic environments.

Real-World Drone Implementation: Transferring the drone agent to a real-world environment would provide valuable information into the challenges and complexities of real-world aerial navigation. This would also validate the methodologies used for the drone in a real-world context and allow for a direct comparison of the performance of the differential vehicle and the drone in real-world settings.

ANNEX I: ALIGNMENT WITH THE SUSTAINABLE DEVELOPMENT GOALS

Industry, Innovation, and Infrastructure: This project promotes innovation and the adoption of intelligent and sustainable technologies within the industry. The development of autonomous systems, such as drones and differential vehicles, capable of navigating efficiently and safely, can find broad applications across various industries. This includes logistics, agriculture, infrastructure inspection, surveillance, and more. By fostering these advancements, the project contributes to building resilient infrastructure, promoting inclusive and sustainable industrialization, and supporting innovation.

Sustainable Cities and Communities: Autonomous systems have the potential to enhance the sustainability of cities and communities. For instance, drones can be employed for goods delivery, reducing the reliance on ground vehicles, thereby decreasing traffic congestion and greenhouse gas emissions. Autonomous differential vehicles can be employed for sustainable mobility applications. By integrating these technologies, the project aids in making cities and human settlements inclusive, safe, resilient, and sustainable.

Partnerships for the Goals: By utilizing open-source platforms and collaborating across various disciplines, this project encourages cooperation and knowledge and technology exchange. The employment of reinforcement learning techniques and collaboration with the artificial intelligence community contributes to the development of novel solutions and technologies.

REFERENCES

[1] Guizzo, E. (2018) "How Robots Are Grasping the Art of Grip," *IEEE Spectrum*. Institute of Electrical and Electronics Engineers (IEEE). doi: 10.1109/mspec.2018.8405398.

[2] Broadbent, E. (2017) "Interactions with Robots: The Truths We Reveal About Ourselves," *Annual Review of Psychology*. Annual Reviews, 68(1), pp. 627–652. doi: 10.1146/annurev-psych-010416-044021.

[3] Yang, G. Z. *et al.* (2020) "Combating COVID-19—The role of robotics in managing public health and infectious diseases," *Science Robotics*. American Association for the Advancement of Science (AAAS), 5(40). doi: 10.1126/scirobotics.abb5589.

[4] Murphy, R. R. (2014) "Disaster Robotics," *IEEE Intelligent Systems*. Institute of Electrical and Electronics Engineers (IEEE), 29(4), pp. 25–29. doi: 10.1109/mis.2014.48.

[5] Murphy, R. R. (2011) "Human–Robot Interaction in Rescue Robotics," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*. Institute of

Electrical and Electronics Engineers (IEEE), 41(2), pp. 138–153. doi: 10.1109/tsmcc.2010.2046732.

[6] Wurman, P. R., D'Andrea, R. and Mountz, M. (2008) "Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses," *AI Magazine*. Association for the Advancement of Artificial Intelligence, 29(1), pp. 9–20. doi: 10.1609/aimag.v29i1.2062.

[7] Guizzo, E. (2018) "How Robots Are Grasping the Art of Grip," *IEEE Spectrum*. Institute of Electrical and Electronics Engineers (IEEE). doi: 10.1109/mspec.2018.8405398.

[8] Li, C., Chen, L. and Chen, B. (2022) "Analyzing tracked search and rescue robots," in El-Hashash, A. (ed.) *International Conference on Biomedical and Intelligent Systems (IC-BIS 2022)*. SPIE. doi: 10.1117/12.2661488.

[9] Weyler, J. *et al.* (2023) "Towards domain generalization in crop and weed segmentation for precision farming robots," *IEEE robotics and automation letters*. Institute of Electrical and Electronics Engineers (IEEE). doi: 10.1109/lra.2023.3262417.

[10] Rong, J. *et al.* (2022) "Fruit pose recognition and directional orderly grasping strategies for tomato harvesting robots," *Computers and electronics in agriculture*. Elsevier BV. doi: 10.1016/j.compag.2022.107430.

[11] Disyadej, T. *et al.* (2020) "Smart Transmission Line Maintenance and Inspection using Mobile Robots," *Advances in Science Technology and Engineering Systems Journal*. ASTES Journal. doi: 10.25046/aj050361.

[12] van Hecke, K. *et al.* (2017) "Self-supervised learning as an enabling technology for future space exploration robots: ISS experiments on monocular distance learning," *Acta astronautica*. Elsevier BV. doi: 10.1016/j.actaastro.2017.07.038.

[13] Zhao, W. *et al.* (2021) "Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey" arXiv [cs.CL]. Available at: https://arxiv.org/abs/2009.13303.

[14] Arab, K. and Mp, A. (2012) "PID Control Theory," in Introduction to PID Controllers - Theory, Tuning and Application. InTech. doi: 10.5772/34364.

[15] Chen C. and Holohan, A. (2016) "Stability robustness of linear quadratic regulators", International journal of robust and nonlinear control. Wiley, 26(9), pp. 1817–1824. doi: 10.1002/rnc.3362.

[16] Schwenzer, M. et al. (2021) "Review on model predictive control: an engineering perspective," perspective," The international journal of advanced manufacturing technology. Springer Science and Business Media LLC, 117(5–6), pp. 1327–1349. doi: 10.1007/s00170-021-07682-3.

[17] D4PG PyTorch Implementation. Available at: https://github.com/schatty/d4pg-pytorch.

[18] Silver, D. *et al.* (2016) "Continuous control with deep reinforcement learning," *arXiv [cs.LG]*. Available at: https://arxiv.org/abs/1509.02971.

[19] Barth-Maron, G. *et al.* (2018) "Distributed distributional deterministic policy gradients," *arXiv [cs.LG]*. Available at: http://arxiv.org/abs/1804.08617.

[20] Cubillo Llanes, D. *et al.* (2022) "Navegación autónoma de un vehículo terrestre mediante una cámara lidar". Repositorio Universidadd Pontificia Comillas. Available at: http://repositorio.comillas.edu/jspui/handle/11531/62114.