SIM-TO-REAL REINFORCEMENT LEARNING FOR ROBOT NAVIGATION

Company : Factory AI

Division : Vision AI Laboratory

Lead's LAB: Mr. CHIM Thenggy

Prepare by : LANG Bandithvipho

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1. What is Reinforcement Learning?

The idea behind Reinforcement Learning is that an agent (an AI) will learn from the environment by interacting with it (through trial and error) and receiving rewards (negative or positive) as feedback for performing actions. Learning from interactions with the environment comes from our natural experiences.

Example: For instance, as shown in **Figure 1** imagine putting your little brother in front of a video game he never played, giving him a controller, and leaving him alone.

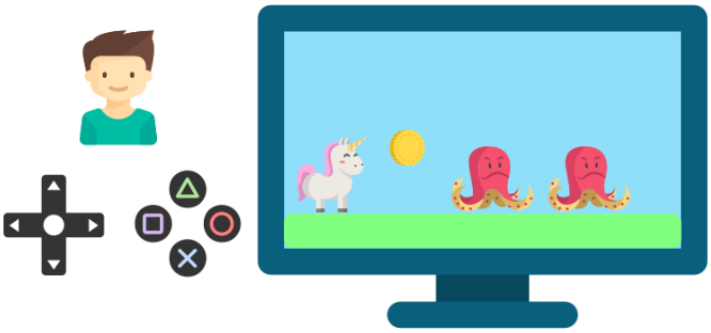


Figure 1.1.

Your brother will interact with the environment (the video game) by pressing the right button (action). He got a coin, that's a +1 reward. It's positive, he just understood that in this game he must get the coins.

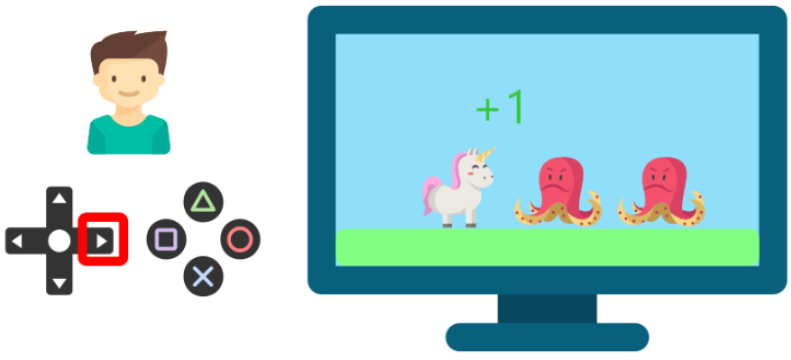


Figure 1.2.

But then, he presses the right button again and he touches an enemy. He just died, so that's a -1 reward.

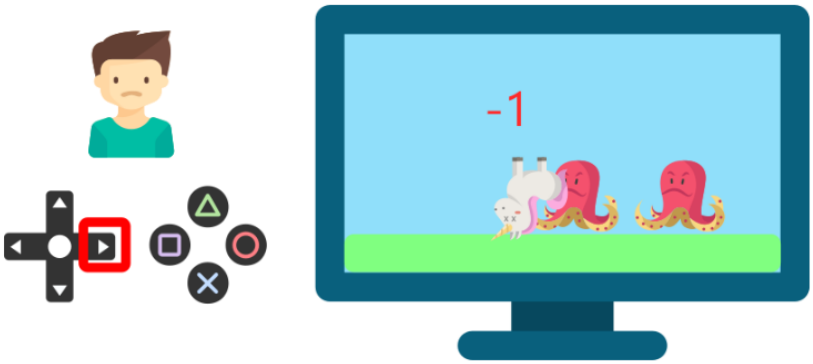


Figure 1.3.

By interacting with his environment through trial and error, your little brother understands that he needs to get coins in this environment but avoid the enemies. Without any supervision, the child will get better and better at playing the game. That's how humans and animals learn, through interaction. Reinforcement Learning is just a computational approach of learning from actions.

A formal definition: Reinforcement learning is a framework for solving control tasks (also called decision problems) by building agents that learn from the environment by interacting with it through trial and error and receiving rewards (positive or negative) as unique feedback.

The RL Process:

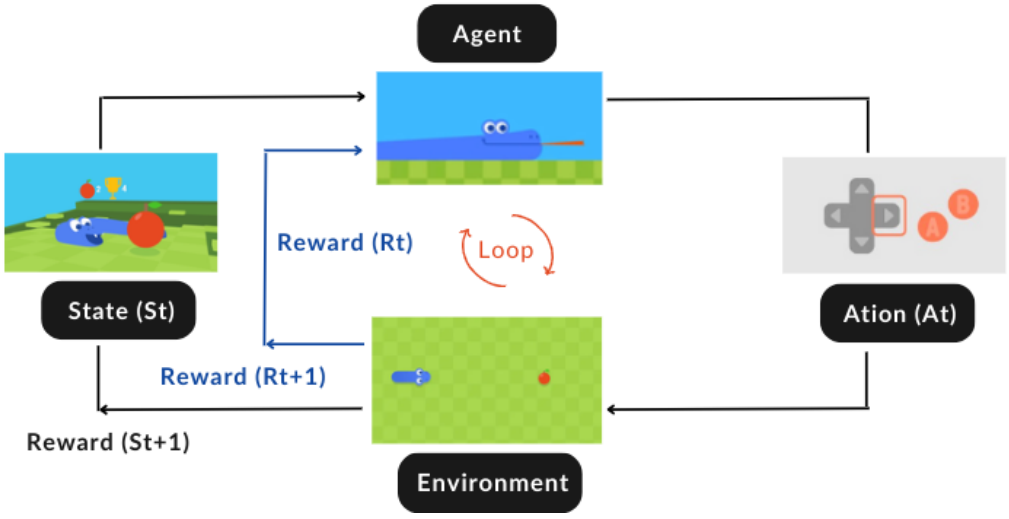


Figure 1.4. Example of RL Diagram for Snake Game

The goal of reinforcement learning is for the agent to learn how to make better decisions by choosing actions that maximize rewards over time. As the agent interacts with the environment repeatedly, it gradually improves its strategy to achieve its goal more effectively.

1.2. Objective

- Self-navigate from current position to a predefined target point with unknown Environment by using Reinforcement Learning
- SIM-TO-REAL Reinforcement Learning with Differential Robot Robot movement visualization Performance of RL model in real robot visualization.

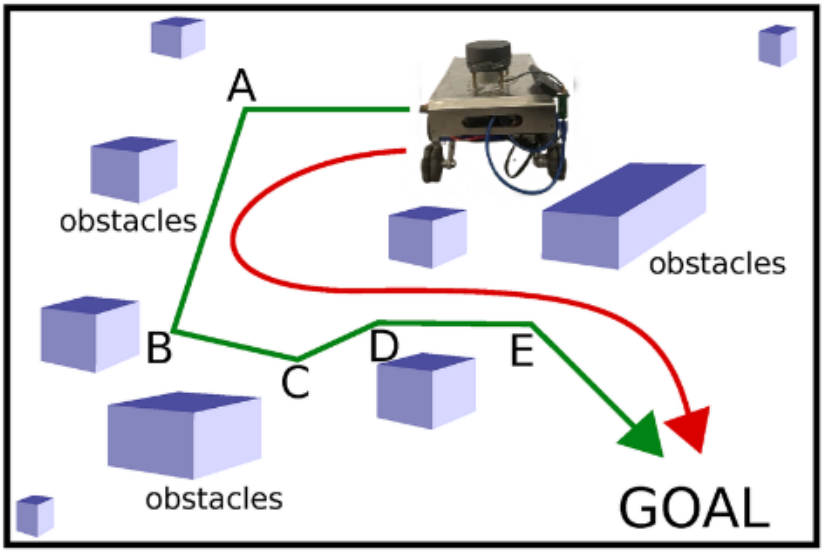


Figure 1.5. Example of robot navigation

1.3. Problem Statement

- Challenges in Robot Navigation:
  - Real-world navigation is complex due to noise, obstacles, and dynamics
  - Training RL model directly in real world can cause:
  - Large amount of real-world data
  - Time consuming to collect
  - Hardware damage
  - Safety risks, hard to manage and control
  - High cost and high computational resource
- Why Sim-to-Real or Zero-shot Sim-to-Real?
  - Sim-to-real enables training in simulation and transferring the knowledge to real robot.

1.4. Project Overview

This project inspire from the project base “Tommaso Van Der Meer” which is using reinforcement learning for robot navigation in “Hospital world” using ROS2, and Gazebo frame work for simulation.

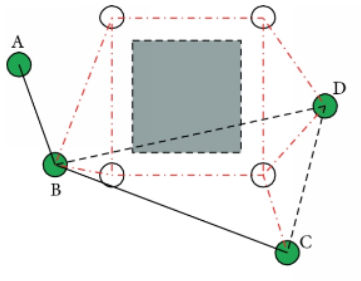
Figure 1.6. Environment from Hospitalbot

Environment of “Hospitalbot”, including some features with difference scenarios, such as:

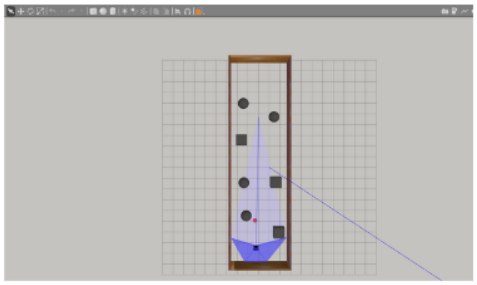
- Random Robot location / Target
- Fix Robot location / Target

1.5. Scope of Works

- Create way-points for Path Planing on mobile robot navigation
- Setup world for model training base Real World Dimensions (4x15m)
- Configure robot’s parameters for model training according to Algobot
- Training RL Model in Simulation (Gazebo World)
- Hardware Implementation -> Deploy RL model into Algobot



a). Way Point construction



b). World Base Real World Dimension

c). Algobot

2. METHODOLOGY

2.1. Soft Actor Critic (SAC)

SAC concurrently learns a policy π_θ and two Q-functions Q_{ϕ_1}, Q_{ϕ_2}. There are two variants of SAC that are currently standard: one that uses a fixed entropy regularization coefficient α, and another that enforces an entropy constraint by varying α a over the course of training. For simplicity, Spinning Up makes use of the version with a fixed entropy regularization coefficient, but the entropy-constrained variant is generally preferred by practitioners.

Exploration vs. Exploitation:

SAC trains a stochastic policy with entropy regularization, and explores in an on-policy way. The entropy regularization coefficient α explicitly controls the explore-exploit tradeoff, with higher αcorresponding to more exploration, and lowerα corresponding to more exploitation. The right coefficient (the one which leads to the stablest / highest-reward learning) may vary from environment to environment, and could require careful tuning.

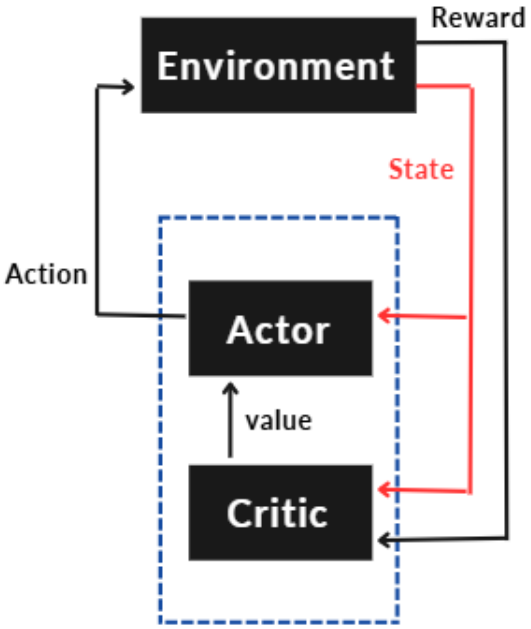


Figure 2.1. Actor-Critic

For reinforcement learning algorithm we're using SAC:

- SAC is a off-policy "actor-critic" algorithm, it learn from experiences generated by a different policy rather than the current one.

  - Actor: Choose best action to execute.
  - Critic (Q): Tell actor how good this choice was.
- Soft Actor-Critic (SAC): It uses a replay buffer to store past experiences, and allowing the model to learn from this data improving sample efficiency and training speed.

Pseudocode:

Algorithm 1 Soft Actor-Critic

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
repeat
Observe state s and select action $a \sim \pi_\theta(\cdot|s)$
Execute a in the environment
Observe next state s', reward r, and done signal d to indicate whether s' is terminal
Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
if s' is terminal **then**
Reset environment state
end if
if it's time to update then **then**
for j in range(however many updates) **do**
Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
Compute targets for the Q functions:

$$y(r, s', d) = r + \gamma(1 - d)\left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', a') - \alpha \log \pi_\theta(a'|s')\right), \quad a' \sim \pi_\theta(\cdot|s')$$

for $i = 1, 2$ **do**
Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2$$

end for
Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s)|s)\right)$$

where $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot|s)$ which is differentiable w.r.t θ via the reparameterization trick.
Update target networks with

$$\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1 - \rho)\phi_i \quad \text{for } i = 1, 2$$

end for
end if
until convergence

2.2. Environment Setup

In reinforcement learning (RL), setting up the environment is crucial as it defines the interactions between the agent and its surroundings. The two main components used in this setup are the Gymnasium framework and the Stable-Baselines3 library, each serving a specific purpose to facilitate the RL training process.

- **Gymnasium framework:** (an updated version of OpenAI Gym) is a framework that standardizes the creation, manipulation, and interaction of RL environments. This framework allows for a consistent interface across various types of environments, making it easier to train, test, and benchmark RL algorithms across different setups.
- **Stable-Baselines3 library:** is a popular Python library for implementing and managing reinforcement learning algorithms. It provides robust and optimized implementations of many commonly used RL algorithms, including Soft Actor-Critic (SAC), which is particularly well-suited for continuous control tasks like robotic movement.

Figure 2.2. Reinforcement Learning Framework

2.3. Gazebo World Configuration

To train the reinforcement learning model effectively, it is essential to configure the environment settings within the Gazebo simulation. This involves defining various parameters in the sdf (Simulation Description Format) file, which describes the world in which the robot will operate. The sdf file is a powerful tool for creating realistic and complex environments, enabling you to simulate various physical attributes, environmental conditions, and world properties.

Key Parameters in the SDF file:

- Real time factor (RTF) is the ratio of simulation time to real-world time. It measures how fast the simulation is running compared to real time.
- Max step size defines the time interval of each simulation step, essentially controlling the granularity of the physics calculations.
  - Smaller Step Size, more accurate and realistic simulation results
  - Larger Step Size, faster simulation with less detail and accuracy
- μ and μ_2 allow for more precise control of frictional forces, enabling simulations of surfaces with different frictional behaviors depending on the direction of force application. **Example in Gazebo:** When configuring friction for a robot's wheels, you might set different values for μ and μ_2 to simulate the grip behavior of the tire differently in forward and sideways directions. Common values:
  - 0.1 - 0.3: Low friction (e.g., ice or wet surfaces).
  - 0.5 - 0.8: Medium friction (e.g., wood or regular flooring).
  - 0.9 - 1.2: High friction (e.g., rubber on concrete or asphalt).
- kp (spring stiffness): A value of 100000.0 means the contact points between surfaces (like robot wheels or feet touching the ground) will resist penetration with a high force, simulating rigid surfaces.
- kd (damping coefficient): A low value like 1.0 means the system does not over-damp collisions, allowing for some realistic bouncing or quick response without the robot getting "stuck" or too slow after collisions.

Parameters	Type	Value
Gravity		9.8 (m/s^2)
max_step_size		0.001 (s)
real_time_factor		1
sensor_noise	"Gaussian"	
Friction coefficient (mu/mu2)		0.9/0.9
Contact Dynamic (Kp, Kd)		100000/1.0

2.4. Robot Configuration

In order to accurately simulate the behavior and capabilities of a real robot in Gazebo, it's essential to configure the robot model to match the real robot's kinematic parameters and sensor specifications. This involves customizing several parameters in the SDF (Simulation Description Format) file or URDF (Unified Robot Description Format) file. Here's an overview of key aspects to consider:

Physical Properties	Value
wheel separation	0.28 (m)
wheel diameter	0.115 (m)
max_wheel_torque	5 (N/m) -> approximate
wheel acceleration	0.6 (m/s^2) -> approximate

Sensor Specification:

To accurately simulate the RPLidar A1 sensor in Gazebo, you need to configure the sensor's parameters in the SDF (Simulation Description Format) file for the robot. These parameters should closely match the real RPLidar A1's specifications to achieve realistic sensing and perception

Parameters	Value
Sample	360 (degree)
update rate	10 Hz
Resolution	0.017
max_angle	6.28 (2Pi)
max_range	12

- Sample: This typically refers to the number of measurements or data points taken by the LiDAR sensor within a specific time frame or a single scan.
- Update Rate: This is the frequency at which the LiDAR sensor updates its data. It is usually measured in Hertz (Hz), meaning the number of times per second the sensor captures and sends a new scan or data point.

- Resolution: The resolution of a LiDAR sensor refers to the level of detail in the measurements or the density of points in a scan.
- Max/Min Angle: These parameters define the angular range over which the LiDAR sensor scans.

2.5. Model Training

Setup scenario for model training:

We train the RL model in a Gazebo world that includes:

- Random waypoints
- Fix robot Position
- Static obstacles
- A waypoint-based layout:
  - The robot is required to navigate along three predefined paths, consist of 10 points.
- Reward system:
  - Reach target => reward + 10
  - Hit Obstacle => reward - 5

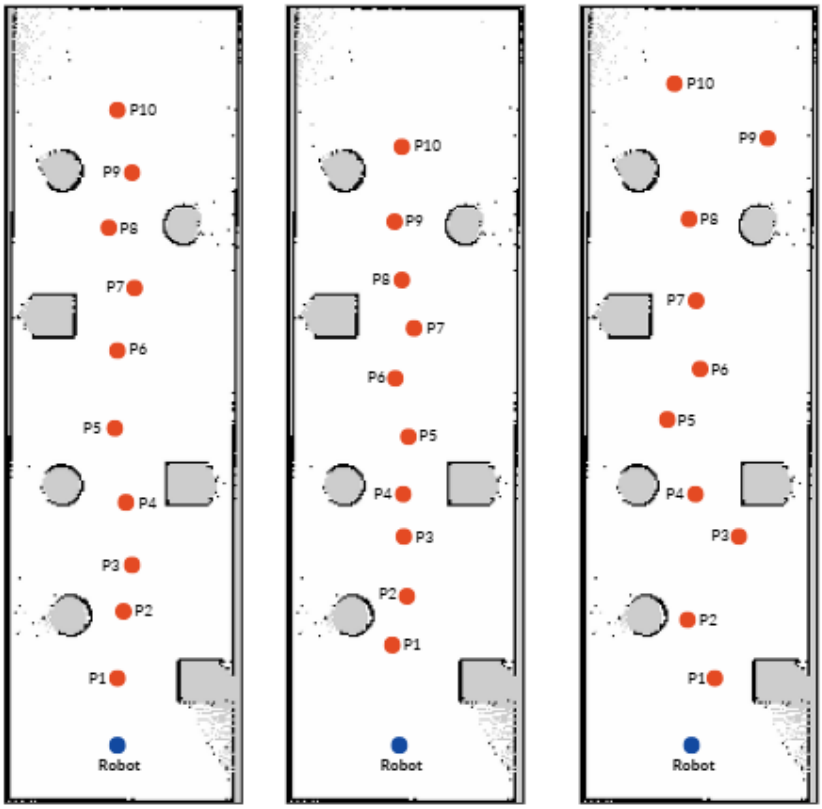


Figure 2.3. Way-points Construction for 3 randoms path

Setup SAC parameters for training:

We utilize the Soft Actor-Critic (SAC) algorithm from the "Stable Baselines3" library to train our reinforcement learning (RL) model. The selected training parameters are outlined in the table below.

Parameters	Value
Type Policy	"MultiInputPolicy"
verbose	1
learning_rate	0.0003
buffer_size	1 000 000
learning_starts	100
batch_size	256
tau	0.05
gamma	0.99
train_freq	10
device	'auto'

- epochs (int): Number of epochs to run and train agent.
- buffer_size(int) – Maximum length of replay buffer.
- gamma (float): Discount factor. (Always between 0 and 1).
- etc...

3. RESULT AND DISCUSSION

In this section, we divide into 2 part of experiment:

- **Simulation:** Train the RL robot in the Gazebo world, which creates the virtual environment and configures it to closely match real-world testing conditions and sensor specifications.
- **Hardware:** Utilize a zero-shot Sim-to-Real approach by directly deploying the model to the hardware (Raspberry Pi 4B) via a local network, without any fine-tuning.

3.1. Simulation

There are 2 models that we use to conduct the experiment in the simulation, “SAC_waypoint02” and “SAC_waypoint03”

3.1.1. For model “SAC_waypoint02”:

The model trains around ”3 million timesteps” with “maximum episode 300”, Using an NVIDIA RTX 3050 Ti GPU.

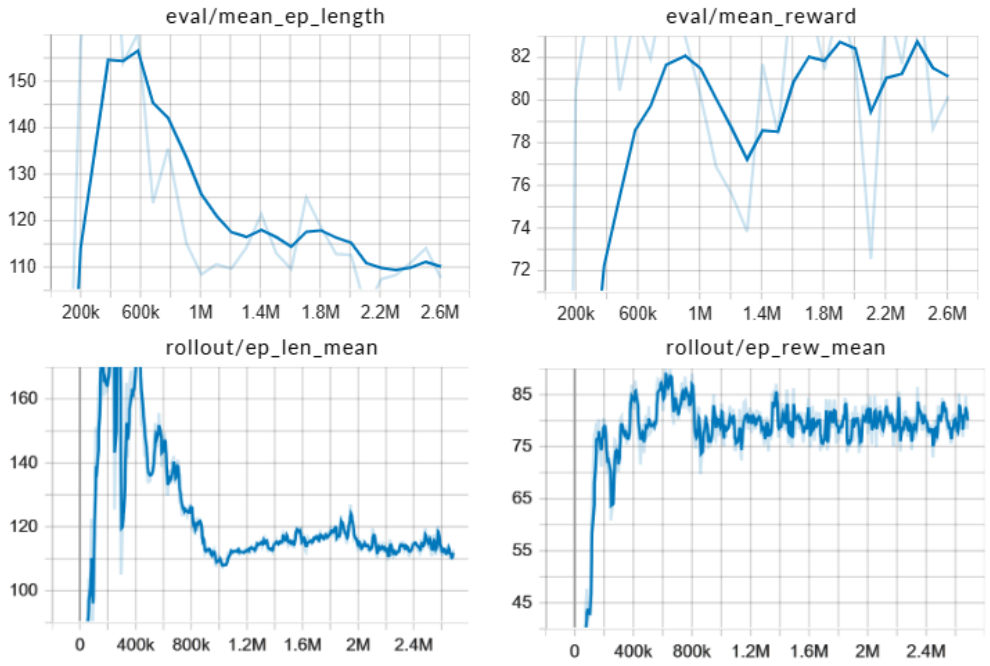
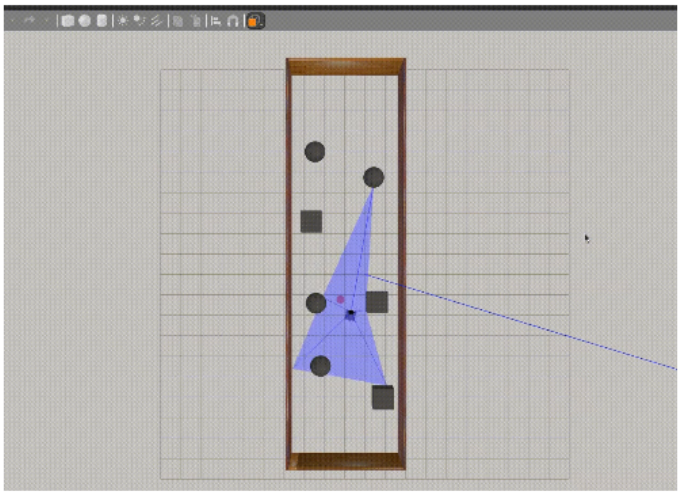


Figure 3.1. Evaluation Episode length and Reward

After, finished training model we get result by:

- Deploy “10 episodes” of RL Model After trained 3 Million Time-step in simulation by using default path from training.



Result	Value
Total Episodes	10
Mean Reward	60
Standard Reward	27.477
Max reward	100
Mean episode length	397.1
Completed Path	5

Figure 3.2. Evaluation Result after deploy model in Robot Gazebo

3.1.2. For model “SAC_waypoint03”:

The model trains over ”10 million timesteps” with “maximum episode 500”, to make model more generalize:

- In 3M time-steps: changed positions of the obstacles
- Increase value of friction

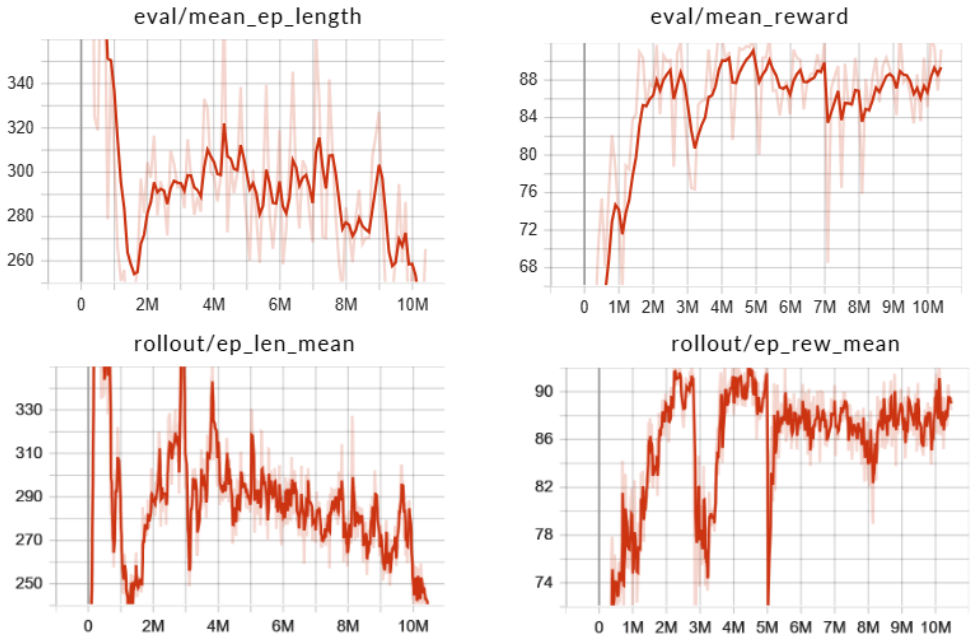
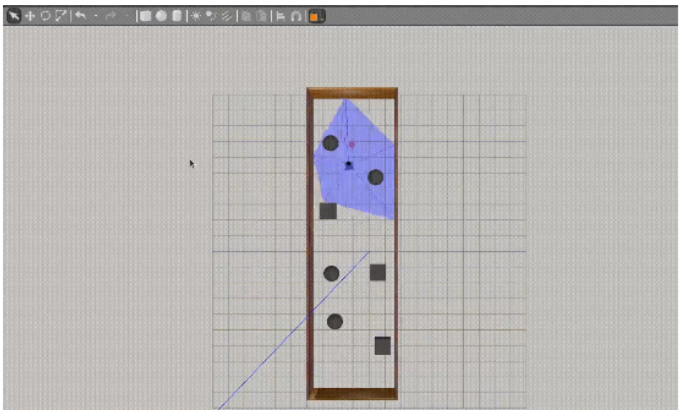


Figure 3.3. Evaluation Episode length and Reward

Here, the result when deploy model in Robot Gazebo that using with default waypoints as in training procedure by:

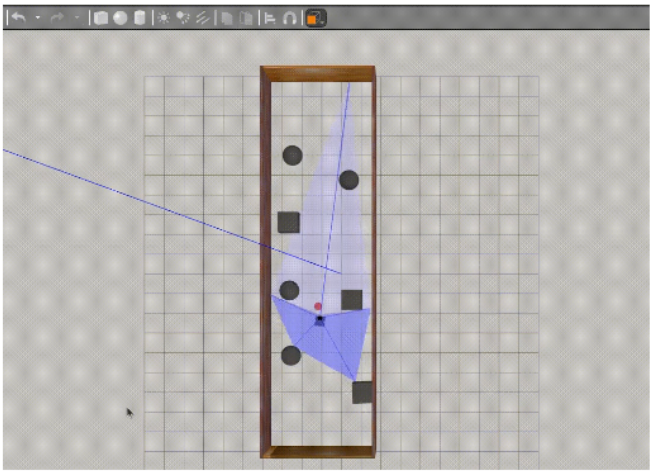
- Deploy “10 episodes” of RL Model After trained 10 Million Time-step in simulation by using default path from training.



Results	Value
Total Episodes	10
Mean Reward	100
Standard Reward	0
Max reward	100
Mean episode length	225.6
Completed Path	10

Figure 3.4. Evaluation Result after deploy model in Robot Gazebo with default path

- Change waypoints, and deploy “10 episodes” of RL Model After trained 10 Million Time-step in simulation by using different path from training.



Results	Value
Total Episodes	10
Mean Reward	85
Standard Reward	30.74
Max reward	100
Mean episode length	285.6 steps per episode
Completed Path	8

Figure 3.5. Result after chagning path

3.1.3. Result Comparison

After conducting experiments with both models, ’SAC_waypoint02’ and ’SAC_waypoint03,’ we can analyze and compare their performance to determine the most suitable model for deployment on the real robot. Key performance metrics, such as reward convergence, training stability, sample efficiency, and success rate in reaching waypoints, are used to evaluate the

effectiveness of each model. Additionally, specific deployment factors, like computational efficiency and robustness under different environmental conditions, are considered.

Result Comparison:

	Total Episode	Mean Reward	Std Reward	Mean episode length	Time for a episode	Completed Path
SAC_waypoint02	10	60	27.477	397.1	39.71s	5
SAC_waypoint03 (default path)	10	100	0	225.6	22.56s	10
SAC_waypoint03 (change path)	10	85	30.74	285.6	28.56s	8

As a result, we can assume that model "SAC_waypoint03" perform better and capable to used for navigating other paths, not just the path it was trained on.

3.2. Hardware Implementation

Execution setup:

We used a Raspberry Pi 4B to run 4 nodes (Micro-ROS, Odom, IMU, and Laser) and publish topics via SSH to a computer. Then computer executed the RL node, deploying the RL model on the Alogobot to navigate through waypoints in a real-world environment.

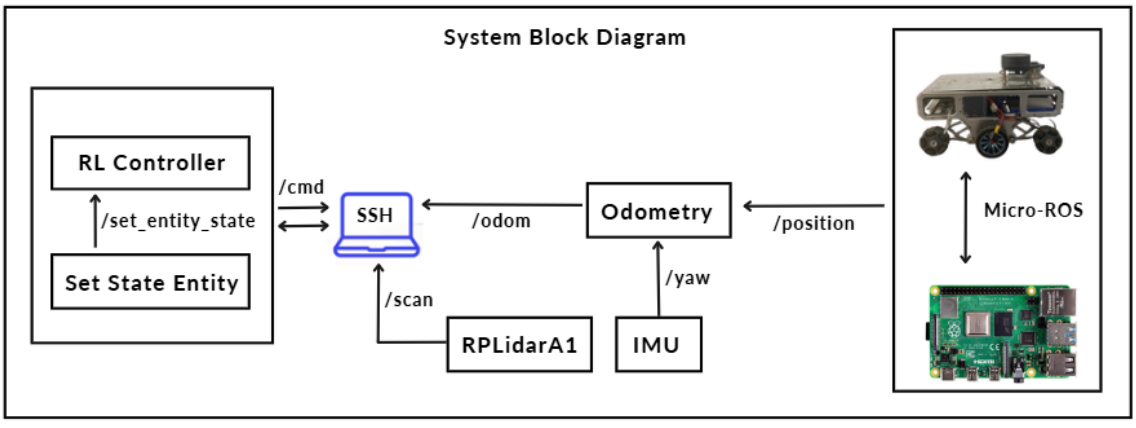


Figure 3.6. Overall System Block Diagram

Deploy RL model on Algobot:

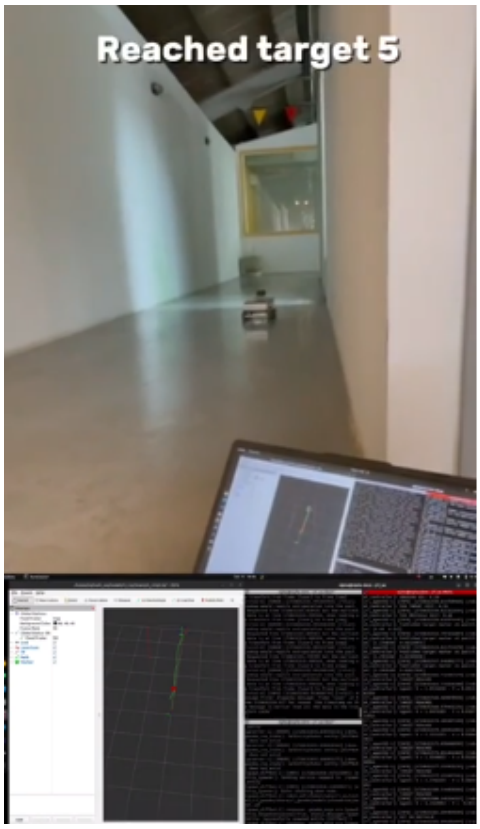


Figure 3.7. Experiment RL model with Algobot

Model Performance on Simulation and Real-World:

After testing in both simulation and the real world, we found that the RL model reached 6 points, indicating potential for effective navigation, even though it did not complete the entire path.

Experiment	Total Episode	Mean Reward	Std Reward	Mean episode length	Time for a episode	Completed Path
Simulation	10	85	30.74	285.6	28.56s	8
Real-world	5	2.8	...	33.66	3.36s	Haft-Path=6 points

As a result, we can see in table above the model performance in simulation better then Real-world even we change the path for it, but in Real-world experiment can achieve only a half-path, the result in real-world seem so bad due to variation of environment, tolerance of electronic component, computational resource and sensor limitation.

4. CHALLENGE AND LIMITATION

4.1. Challenge

Challenges while training:

- Simulation may not fully replicate real-world physics, sensor noise.
- Sensor data is often noiseless
- Slower training times:
- Delay experimentation and model optimization cycles
- Model doesn't generalize well
- Scale and Space Mismatch
- Overestimating distances between waypoints, walls, or obstacles

Challenges while deploy:

- Sim-to-real gap, sensor noise, delay
- friction, dynamic change
- Delayed Response from Network Communication
- Real-Time Constraints

Key improvement:

- For training: Use domain Randomization techniques Test network latency
- Try to optimize data transfer between PI4 and Computer

4.2. Limitation

As a result of this experiment, we can observe that our project has some limitations that affect its overall performance:

- Reality Gap Between Simulation and Real World
- Model Generalization
- Limited Computational Resources
- High Training Time
- Zero-shot sim-to-real, may not scale well to larger or more complex environments Network Latency affect robot response times

5. CONCLUSION

Conclusion:

This project shows the potential of using reinforcement learning (SAC) for autonomous robot navigation, with training in simulation and deployment in the real world. While promising, challenges like the reality gap, model generalization, and hardware limitations need to be addressed for better real-world performance.

Project Insights:

This project lays a strong foundation for future development, with significant potential for autonomous navigation in dynamic, real-world environments. By addressing the challenges and refining the approach, the system can be made more robust and reliable for real-world applications.

REFERENCES

[1] https://stable-baselines3.readthedocs.io/en/master/modules/sac.html

[2] Accelerated Sim-to-Real Deep Reinforcement Learning: Learning Collision Avoidance from Human Player Hanlin Niu, Ze Ji, Farshad Arvin, Barry Lennox, Hujun Yin, and Joaquin Carrasco

[3] Sim-to-Real Transfer with Incremental Environment Complexity for Reinforcement Learning of Depth-Based Robot Navigation Thomas Chaffre1,2 , Julien Moras3 , Adrien Chan-Hon-Tong3 and Julien Marzat3

[4] https://www.catalyzex.com/paper/realtime-collision-avoidance-for-mobile

[5] Wang, B., Liu, Z., Li, Q., Prorok, A. (2020). Mobile robot path planning in dynamic environments through globally guided reinforcement learning. IEEE Robotics and Automation Letters, 5(4), 6932–6939. https://doi.org/10.1109/lra.2020.3026638