Contents

1 Introduction

1.1 Introduction

These days, mobile robots have taken place in many fields like industry automation, planetary exploration, entertainment, and construction ..., for their ability to work in extreme environments with high precision and without fatigue[1].Even so, a robot occasionally needs the support of other robots because it is impossible or difficult for them to perform some tasks on their own. For that, a new field has emmerged to deal with these problems, swarm robotics.

Swarm robotics is relatively a new research topic that has gained more attraction in the last few years. It is about studying how a large number of simple robots (a swarm) can collaborate and work together to achieve predefined objectives and tasks that are often difficult or impossible to do for a single robot.[2].

One of the main challgenes that swarm robotics researchs face, is pattern formation. Where the agents (robots) try to form diffrent gemoetric shapes like squeres,trinagles and circles in order to perform a specific task.

We can solve this problem using two different aproches. The first one is a Centrelized method where there exist a centel unit which controles the swarm and give global state access. However, implementing this apporch can be coslty and less robust to faillures. The second aproch is a dicentrilzed one, where each robot have uses local commucintion and have access only to his local state.[3]

Our primary objective in this thesis is to implement an RL algorithm in a system made up of a group of robots in order to form some specific geometric patterns using a decentralized method. Each robot will only have access to its local state and will interact and communicate with its neighbors in order to form the desired shapes.

2 Swarm Robotics

2.1 Swarm Intelligence and Social animal inspiration

Social animal and insects behavior in groups like the bees dancing, wasp's nest-building, ant's collaboration , bird flocking and fish schooling has caught the attention of researchers for their ability to archive complex tasks and working in coordination, which demonstrate some form of swarm intelligence that researchers have taken inspiration from to design and implement swarm robotics systems.[4]

They also report that social insects were able to accomplish thier goals like searching for food, alerting the presence of an enemy, or collaborate to lift heavy objects, without having access to the global state or having a leader to guide them. They were only able to accomplish this by utilizing local interactions and communication, which spread to other members and and prompted group-wide cooperation.[4]

2.2 Swarm robotics and multi robot systems

The early 1980s are when multi robots systems first gained popularity. As the name suggests, multi robot systems introduce the idea of teamwork in order to complete tasks that are challenging or impossible to complete alone by the robots. Seven topics of study have been identified in this feild which includes:

- Biological Inspirations;
- Communication
- Localization, mapping, and exploration;
- Object transport and manipulation;
- Motion coordination;
- Reconfigurable robots.

Swarm robotics is a subfield of multi-robot systems that differs from other multi-robot systems in some ways.[5]

2.3 Swarm robotics

Swarm robotics has no one definition because it is a rapidly developing area and new research is constantly being done in it. But we can take this defintion from a cited paper. "swarm robotics is the study of how large number of relatively simple physically embodied agents can be designed such that a desired collective behavior

emerges from the local interactions among agents and between the agents and the environment."[6].

The main characters of swarm robotis are:

- The robots in the swarm must be autonomous.
- The number of robots in the swarm is large.
- Homogeneity is required in robots. A small number is acceptable if not.
- Robots must be incompetent with regard to the primary task they must complete, otherwise, they will fail or perform poorly.
- Robots are limited to local communication and sensing. It makes sure that coordination is spread, making scalability one of the system's characteristics.[4]

2.4 Swarm robotics properties

Swarm robotics have some properties that describe the system's current condition. Bellow are some of these properties:

Robustness: the ability of the swarm to still function even with the loss of some members of the group or the faillure of some parts of the system.
Scalability: The ability of the system to perform well on smaller or larger group sizes without impacting the performance of the swarm.

Flexibility It is the capability of the swarm to adapt and manage the new changes that occur in the environment

Autonomus: Implys that there is no central authority controlling the behavior of the swarm and each individual is independent of the others.

Local communication The communication among swarm members is local since they don't have access to the swarm's overall state. [7][8]

2.5 Domain of application

In this section, we will list some of the significant fields domains where swarm robotics fits in, and can impact in solving the domain problem.

2.5.1 Tasks that cover a region

: Because of the widespread sensing capabilities that the swarm has, swarm robotics is most suited for tackling problems that cover an area of the space, such monitoring the environment of a lake or surveillance of a specific area.[6][8]

2.5.2 Search and rescue missions

In different types of accident or disasters that happen like earthquakes where the human intervention is difficult, swarm robots can be deployed for these type of missions. examples of such robots are: polyobot, swarm bot and M-TRAN.

2.5.3 Cleaning of oil speals

A swarm of robots can reduce the cost and time in such incidents. an exemple of robots that were deployed to such tasks is: Seaswarm, which was developed by the senseable group at MIT.

2.5.4 Exploration

To investigate Mars, swarm robots like Marsbees have been created. the same holds true for the CoCoRo swarm, which was employed for in-depth underwater investigation.

2.5.5 Agriculture

As they can be used to improve the agriculture and monitor the status of crops.

2.6 Swarm robotics basic tasks and problems

SR problems can be broken down into fundamental tasks that the swarm frequently carries out in an effort to accomplish its objective. this tasks are: aggregation,d ispersion,pattern formation, coordinated movement, hole avoidance and foraging[3][4].

2.6.1 Aggregation

Aggregation of the robots is performed in order to accomplish some form or to exchange information. This problem can be easy in a centralized system, but difficult in a decentralized one.

2.6.2 Dispersion

For exploration purposes, sometimes, the swarm must cover a wide range of area without losing the connection between the members, in order to expand the group sensing capabilities.

2.6.3 Pattern formation

Sometimes, the swarm must form some specific patterns like circles,squares or lines in order to lift some objects or traverse some ways corridors.

2.6.4 Cordinated movement

It is making an effort to coordinate the group movements by maintaining the established pattern between the robots.

2.6.5 Hole avoidance

As suggested by the name, the group makes an effort to avoid stepping into holes.

2.6.6 Foraging

The swarm aims to locate objects, pick them up, and position them where needed.

3 Deep reinforcement learning

3.1 Introduction

In this chapter we will provide an overview about the feild of reinforcement learning, including its applications, various techniques, and how it relates to swarm robots. Before that,we will briefly discuss machine learning since it is the foundation of RL before we move on.

3.2 Machine learning

Machine learning is a sub field of artificial intelligence, which focuses on developing algorithms and models that make prediction and take decisions without being explicitly programmed to do so based on the input data they receive. Generally speaking, there are three types of machine learning algorithms, suppervised algorithms which obtain labeled data and attempt to categorize or anticipate the unknown one. Unsupervised learning algorithms, where the data is unlabeled and it is the job of the ml algorithms to extract hidden patterns in it. And finally, reinforcement learning algorithms where the algorithm learns to make decisions based on trial and error and rewards, by interacting with an environment.

3.3 Reinforcement learning

Reinforcement learning algorithms are based on a agent that interact and observe an environment in order to take actions that maximize the reward for a given task.

The goal of the agent is to learn an optimal policy (Π) which maps state to actions in order to get the best possible action to take in a given situation. This can be achieved by maximizing the expected reward that the agent receive from the environment on each step he takes.[9]

Formally, RL can be described as a Markov Decision Process (MPD) where the future state depends only on the current one. An MPD consists of:

- A set of states S
- A set of actions A
- A transition dynamics $\rho(S_{t+1}|a, S)$. which give the probability of being in state S_{t+1} based on the current state S and the performed action A.
- A reward function $R(S_t, A_t, S_{t+1})$ which outputs a scalar reward $r_t \subset R$.
- A discount factor $\gamma \in [0, 1]$ which emphasis either immediate or future rewards.

From the above definition, the policy function Π will a map a given state to a probabilty distribution over actions: $\Pi : S \longrightarrow \rho(A = a|S)$. The goal of the agent to find the optimal policy Π which maximize the expected return :

$$\Pi^* = argmax_\Pi E[R|\pi]$$

If the MPD is episodic, then the agent will accumulates a set of reward at the end of each episode which is called a return:

$$R = \sum_{t=0}^{T-1} \gamma^t r_{t+1}$$

Setting $\gamma < 1$ will ensure the convergence of the return in case of non episodic MPDS.[9].

3.4 Q learning

Q-learning is a type of model free reinforcement learning algorithm based on the dynamic programming technique, where the agent tries to maximize it rewards by finding the optimale policy Π in an iterative manner. In order to acheive this, the agent uses a type of equation called value function that gives it the value or the reward of being in a state s and taking action according to it policy Π.

$$V^\Pi(S) = E[r(s, a) + \gamma V^\Pi(s')]$$

In order to find the optimal value function, hence finding the optimal policy, the equation must satisty the bellman optimality condition which states that:

$$V^*(S) = max_a E[r(s, a) + \gamma V^*(s')]$$

The same goes for the action-value function which gives the return if the agent chooses the action a and follow the optimale policy thereafter.

$$Q^*(S, a) = E[r(s, a) + max_a \gamma Q^*(s', a)]$$

[10].

3.5 Deep Q learning

The problem with traditional Q-learning or Rl algorithms in general, is the inability to work in a high dimensional input states where discritization and hand-crafted features extraction is performed in order to it to work. For this, deep learning was combined with traditional Q-learning in order to overcome these challenges.[11]

Deep Q learning was firstly introduced in an article by the deepmind group, where it was tested in atari games in which the agent was given raw input images of the emulator and it was the goal of the agents to win the games by using the Q learning algorithm.

Besides using the bellmen equation to converge and calculate the loss, an experience replay memory was used to store the the experiences of the agents during the game and then sampled randomly to be fed to the neural network to learn and in order to break the correlation that apears when fedding them sequential data.[11]

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights
for `episode = 1, M` **do**
`Initialise sequence s1 = {x1} and preprocessed sequenced` $\phi_1 = \phi(s_1)$
for `t = 1, T` **do**
With probability ϵ select a random action a_t
otherwise select $a_t = max_a Q^*(\phi(s_t, a; \theta)$
Execute action a_t in emulator and observe reward r_t and image x_{t+1}
Set $s_{t+1} = s_t$, a_t , x_{t+1} and preprocess $\phi_{t+1} = \phi(s_{t+1})$
Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
Sample random minibatch of transitions $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
set

$$y_j = \begin{cases} r_j, & \text{for terminal } \phi_{j+1}. \\ r_j + \gamma max_a Q(\phi_{j+1}, a'; \theta), & \text{for non-terminal } Q(\phi_{j+1}). \end{cases} \tag{1}$$

Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))$
end for
end for

3.6 Deep Deterministic policy gradient

One of the major problems of DQN is that he works only in discreate action spaces. For this, tasks that require continuous control, such as the linear and rotational movements of robots, require discretization. In most cases, this can result in a large action space, which slows convergence and causes instability.. [12]

For this, there is another type of DRL algothims named Deep Determinisitc policy gradient, which is a policy based method actor critic algoithm, that takes the benifites of both methods (policy based method and value based method, and useses the actor critic to reduce the bias (underfitting) that occurs ...

DDPG in composed of two neural networks: the actor and the critic. The job of the actor is to select an action based on the input state. And it is the critic who will determine if that action was suitable or not by observing the rewards returned when taking that specific action in a specific state.

The critic network is learned using bellman equation same as q-learning.

The DDPG algothim also useses a replay buffer to store agent exepriences in order to sample from them later when training the netwroks.

Like in Q-learning and to enable better exploration for the agent, a noise is sampled from a process η then added to the output of the actor network. It is common to use the Ornstein-Uhlenbeck process to generate temporally correlated values.[12]

Algorithm 2 Deep Deterministic Policy Gradient (DDPG)

Initialize actor network $\mu(s|\theta^\mu)$ and critic network $Q(s,a|\theta^Q)$ with weights θ^μ and θ^Q
Initialize target networks $\mu'(s|\theta^{\mu'}) \leftarrow \mu(s|\theta^\mu)$ and $Q'(s,a|\theta^{Q'}) \leftarrow Q(s,a|\theta^Q)$ with weights $\theta^{\mu'} \leftarrow \theta^\mu$ and $\theta^{Q'} \leftarrow \theta^Q$
Initialize replay buffer R
for episode $= 1$ to M **do**
    Initialize a random process for action exploration
    Receive initial observation state s_1
    for $t = 1$ to T **do**
        Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}t$ according to current policy and exploration noise $\mathcal{N}t$
        Execute action a_t and observe reward r_t and new state $st+1$
        Store transition $(s_t, a_t, r_t, st+1)$ in R
        Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
        Update critic by minimizing the loss: $\mathcal{L} = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
        Update actor policy using the sampled policy gradient: $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|s = s_i, a = \mu(s_i)\nabla\theta^\mu \mu(s|\theta^\mu)|_{s_i}$
        Update target networks: $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$ and $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
    end for
end for

3.7 Related Work

In this section, we present some of the work that have applied RL algorithms in training agents and robotics systems.

The most used algorithm is Deep Q learning, where [11] have applied it in learning playing atari games by using a convolutionnal neural netwrok by providing raw image pixels as input, where they demonstarted that the agents achived great performance even surpassing human level.

In continous action spaces, [12] have used the ddpg algorithm, which is an extension of the DQN and the DPG algorithms. They have used an actor critic model free policy based algorithm which operate in conitous action space by appliyng it in several simulated physics tasks, and also shows great result achived by the agents.

In field of multi agents. [13] where they used a decentrilized aproach to train multi agent to form some patterns using the DQN network with a central replay buffer that contains agents experinces in order for them to learn cooparitvly.

For robotics, [14] have used the Proxiaml policy optimization algorithm (PPO) wchih is a policy based one , to controll in a decentrilzed manner a group of robot to reach their target by avoidng collison between them. Also in [15] have used the ddpg algorithm to traing mutli robtos to encircle some target in a static position or

in mouvement where the agents (robots) learn to coopretivly achevce their task by using also a central replay memory.

Finnaly, in [16],[17] and [18] have applied the ddpg and dqn algorithms for path playing and postion control of robots in order to achive the target goals.

4 Problem formulation And Solution Conception

4.1 Introduction

In this section, we'll go into detail about the objective that needs to be accomplished by our robots, the difficulties they must overcome, and then our suggested solution.

4.2 Goal defintion

: Our objective in this thesis is to make multi robots form geometric patterns like lines, circles, squares, and triangles while avoiding collisions and moving toward their destination in a quick and efficient manner.

4.3 Problem Formulation

4.3.1 Environement

We will describe our multi robot environemnt as an MDP where S are the set of states that each robot will reacive. A are the set of actions that it can take (left,right,towords ...). and R are the set of Rewards that each robot will get during each episode. The goal of each robot is to maximize his return by reaching the goal target and avoiding collisons with the wall and other robots. . We will use the DDPG alrorithm to controll the robots in a contius manner

For the environemnt, we will Consider a scenario in which there are n ($n > 2$) robots moving in a 2D environment. Assuming that every robot is aware of a central point that they must form shapes around or along it.

Figure 1: formation

We'll utilize the polar coordinates (R,Φ), where R and Φ are the radius and angle from the central point , respectively. After assigning each robot a goal position, we will convert it to homogeneous coordinates so it may be reached by them. The desired shapes will be formed by defining the distance and the angle from the center of the formation and by using each robot id to generate unique goal for it .

4.3.2 Shape generation

In order to form the different shapes, each robot will have an $id = (1...n)$ where n is the number of robots, and he will get assigned a goal based on it using the radious and the distance from the central point. So if the target shape is a triangle the target goals will be as follow:

$d_i = R$ and $\Phi_i = \Phi$

where d_i and Φ_i are the distance and the angle from the central point respectively. So if they want to form triangles ,squares, or circles this will be their positions:

$d_i = R$

$\Phi_i = 2\Pi/id$

For lines

$d_i = R * id$ $\Phi_i = 0$

and we can keep modifying the distance and angle for each robot to get any shape.

And to simplify the job for the robot, we will transform the cordinates from polar to homogonous in order to calculate the distance between the robot and their target goal.

4.4 State Representationn

4.4.1 Introduction

The state and reward function must be well designed in order to generate the desired shape because how they are represented will directly impact our robot's performance towards achieving their goals and the architecture of the neural network.

4.4.2 State Representation

State are the inputs to our neural network. They are the inputs for our neural network and based on them, we can calulate the rewards that the robot will get when acting in the environement.

In our implementation we have defined four kind of state information, the distance to goal, angle to goal, the minimun detected object distance and it's angle for collision avoidance.

- **The distance to goal**: The first component in our state representation is the distance to goal, which will be calculated using the euclidean distance between the robot and the goal .

$$D(P, G) = \sqrt{(Py - Gy)^2 + (Px - Gx)^2}$$

Where P and G are the current homogeneous coordinates of the robot and the target goal respectively.

- **The angle to goal**: The second component is the angle to goal. And to find it we need to calculate the difference between the robot yaw, which is it's orientation around it's vertical axis, and the angle between the goal and the robot positions.

$$\theta(P, G) = yaw - arctan(P/G)$$

- **The minimum detected distance**: In order to detect other robots and avoid collision, we use an LDS (we will talk about it later when we describe the robot equipment) which gives us an array of 24 elements where each value is the minimum distance detected and each index is the angle of detection.

- **The angle of detection**: It is is the angle of the detected object relative to the robot lds.

This is a common state representation used to control a moving robot towards the goal target,as they have provide a very clear guidance for it. [18] [17]

In the next part, we will talk how we calculate the rewards of each robot to guide it in a efficent manner towards it goal based on the state input .

4.5 Reward Function Design

The reward function design is a critical part of any RL system. A well designed reward function will give the robot more feedback when acting in the environment thus making the task more straight forward.

The reward function was designed to engourge and reward the robot more, when he aproches the goal target, and penilize him when he collide with the wall or other robots.

The design of the function is based on the state. We have used the distance , the angle and the LDS to form the final function.

We have divided the function into three parts: The first part contains with the distance reward. We will give the robot more reward when he aporoches the target goal and less when he doesn't. Formally it can be described:

$$R_D = -(D_{goal} - D_{init}) + \alpha \tag{1}$$

Where D_{goal} is the changing distance between the robot and the target goal,D_{init} is the initial distance between the robot and the target goal. and α is a parameter to adjust.

The second part is to the angle of the robot to the goal. formally:

$$R_\theta = -(\theta_{goal}) + +\beta \tag{2}$$

where θ_{goal} ranges between $-\pi$ and π. we added the β which add some reward when the angle aproches zero.

And For avoiding collion between the robots and the wall, we have set two values. The first one is set when the robot aproches the object which can lead to a collision. And the second one is when is actualy collide with it.

$$R_c = \begin{cases} -10, & \text{if } mdd < 0.5m. \\ -500, & \text{if } mdd < 0.13m \text{ (a crash).} \\ 0 & \text{else.} \end{cases} \tag{3}$$

where mdd stands for the minimum detected distance. Finally, we can add 1, 2 and 3 to form the finale reward function:

$$R = R_D + R_\theta + Rc \tag{4}$$

4.6 Formation Control Algorithm

Algorithm 3 Shape Formation Algorithm

Initialize $n = N$ number of robots.
Initialize $c = 0$ goal counter.
Assign each robot an id starting from $id = 1...N$
Set the central robot with $id = 1$.
The central robot will choose a random postion and stay on it.
The central broadcast a message containing his position with the desired shape by using the distance d and angle θ from his position.
while $c! = N$ **do**
    for robot in robots **do**
        Reacive the message with the target goal.
        Reach the target goal.
        Inform the central robot with his current postions every $tseconds$.
        if target position reached **then**
            Stop the robot and inform the central robot.
            $c++$
        end if
    end for
end while

4.6.1 Explanation

The first robot will choose a random position and place himself in it. Then he broadcast a msg containing his position and the target positions to other robots in order to form the shape by using the distance and the angle.

Once the message reacived by other robots, each one of them will try to reach the target goal autonomously by avoiding colision with other robots. And once they reach their goal they will inform the central robot with their task completion.

Finally, and if all robots are in place, the central robot will check the state of the formation a and declare the task as complete if all the shape is formed correclty

.

4.7 Formation and Deep Reinfocement Learing

4.7.1 Introduction

Until now, we haven't talk about the role of DRL in this mission.

Each robot will have it's own neural network which will guide him towards the goal positions by avoiding obstacles with other robots.

As mentioned earlier, we have used the DDPG algothim where each robot will have two neural networks, the critic and the actor.

The actor model will get as input the mentioned state and outputs two actions: the angle and the velocity.

The robot will learn to move towards the goal by adjusting the angle and will use the velocity to learn how to move faster in certain conditions and when not. For example the velocity will help him in colision avoidance. If he doesn't detect any robots in front of him he moves faster, else he slows down.

For the critic, he gets as input both the state and the actions, and outputs the q-value which will be used by the actor to choose the suitable actions to take.

4.7.2 The learning flow

It starts by the robot being in a given state, he then takes an action, reacive a reward and transition to a new state.

This informations will be stored in a replay buffer, and once it reach a certain number of experiences the training starts.

If all robots reach their goal, the simulation is reset and a another formation shape is generated in order to prevent overfiting a certain type of shapes.

And if they collide the simulation is reset allso and the robots tries again to form the desired shape.

This learning flow will continue until the models converge and the robots form the shapes many times respectively.

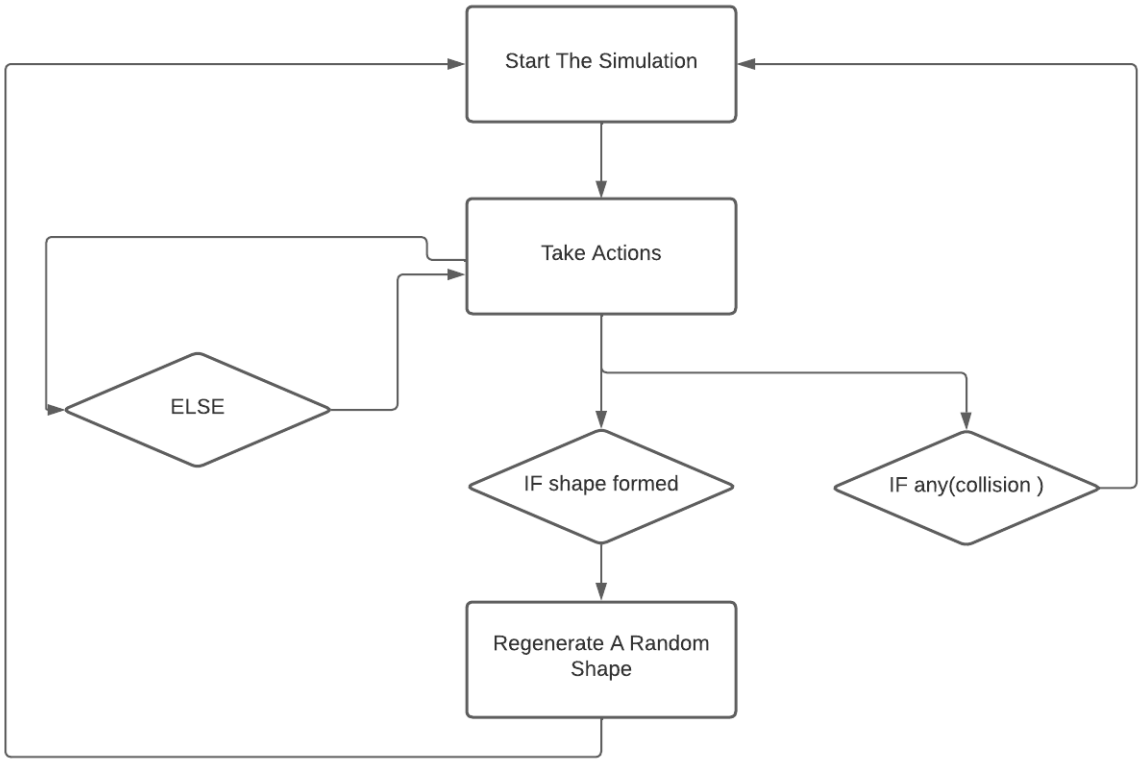

Figure 2: simulation low

5 Implementation

5.1 Introduction

In this section, we will explain how we have implemented the solution , how the neural netwoks are desgined, what frameworks, tools and simulators used to train and test the robots shape formation.

5.2 Roboting Operating system

To implement our work and test it, we used the roboting operating system (ROS), which is a collection of open source software and tools, that simplifies our work by giving us the necessary tools for controlling the robots, determining their positions and angles, gathering sensor data and preparing it for manipulation, and also giving the robots a way to communicate with one another.

5.2.1 Ros Basics

There are five concepts that ROS is built on that will allow us to implement our solution in simple and modular way.

- **Nodes**: Ros is made up of nodes where each node is responsible for a single giving task like moving the robot, getting sensor data, doing some processing ... nodes can communicate between the using topics and services.

  Each node can subscribe or publish to one or more topics. It can contains also one or more services which other nodes can request to get a one time information. This aproach used by ros will allow for modularity and more orginized and clean code.

- **Topics**: topics are a way for nodes to communicate between them. Each node can either subscribe to a topic thus reaciving informations from other nodes, or publish to some topic to send information to other nodes.

  For example: a node can subscribe to a topic that publish the position of the robot from another node which is responsible for providing this information.

  When defining the topics, we first need to define the exact type of information that work with. For instance, we can define topics to accept only string ,numbers, arrays or we can define our custom data type as we will do later.

- **Services**: They are the same as topics but they differ in that data is received when requested by a client in contrast to topics where there are a stream of data updated continuously. This can be helpful in situations where the inforamtion is needed only once or the update frequency of it is low.

For example: We can have a service that generate goals or reset the simulation. Same as topics, we must set the data type that the services work with.

Ros support c,c++ or python as programming languages. in our project, we choosed python. For the operating system, Ros support mainly linux systems, the support for windows was added until recently. We have used ubuntu for the existing support and large ros community .

5.3 TurtleBot

We will be using the turtlebot robot which is developed by open robotics. They same organization behind ros and the gazebo similator.

This robot is used for educational,reaserch and prototyping purpuses, where multiple version have been developed. In this thesis, we will be using the turblebot3-burger version.[19]

5.3.1 TurtleBot3 components

TurtleBot3 is a two wheeled, small size lightweight robot equipped with a variety of sensors, including a 360-degree laser rangefinder for obstacle avoidance, and an IMU for navigation and localization. The robot is powered by a Raspberry Pi single-board computer and it is compatible with ROS.

- **LDS**: LDS, which stands for laser distance sensor. which uses laser to detect the distance between the robot and it's surrounding to allow the robot to navigate in the environment and avoid detected obstacles. The used LDS in this project can sense object in 360 degree with a maximun length of 3.5 meters.

  The (LDS) was one source of data to train our neural networks besides the robots positions. we have used it to detect other robots and to avoid collisions during the navigation. we have minimized the original 360 sample from the LDS as no need for such precision in our project.

- **IMU**: IMU stands for Inertial Measurement Unit, is a type of electronic sensor that is used to measure the orientation, position, and velocity of a moving object. The IMU combines information from many sensors, such as accelerometers, gyroscopes, and magnetometers, to give a thorough picture of the movement of the object in three dimensions. The position of the robot is calulated relative to a fixed refrence frame which either define by the system designer like a closed room or factory or by the robot himself in outdoor environement.

5.4 Simulation

It's hard , time consuming and costly to train the robots in real world. For that, simulation is used to train the robots as it allow us to train in a faster repetable way, let us put the robots and the environement in different conditions for faster experimentations.

Robotics simulation programs like Gazebo, Webots, V-REP, MATLAB Robotics System Toolbox, and CoppeliaSim are some of the more well-liked ones. each one of them has it's own features from programming language support to cost.

In our project we used the gazebo simulator as it is free, open source and integrete well with Ros.

Gazebo is an open source collection of software libraries that have been developed for robotics developers and educationers.

It allows to simulate robotic systems in a virtual environment, providing a safe and cost-effective way to test and refine robot designs before deploying them in the real world. It contains a lot of prebuild models sensors and give the user the ability t build custom one. Gazebo is also able to simulate real world physical phenomena, including physics-based motion, collision detection, and sensor simulation.

5.5 Neural network implementation

Designing the architecture of the neural networks and choosing the right parameters to fit is a crucial part to converge and obtain better performance.

There is a high number of parameters to choose from, hidden layers, number of units, activation functions, weight initlization, regularization ... and each one of them has it's role, and there is no clear rule for the right one to choose, so trial and experimenting is the way to go.

In this section, we will discuss what parameters we choosed and explain the reason behind them.

For the actor network, we set for hidden layers where each layer has 400, 300, 128 and 64 units respectively. Each layer is using the relu activation function and was initialized with Xavier Glorot's exepect for the output layers. The output layers have two units, one for the angular velocity and one for the linear velocity. They was initalized with the tanh activation function which give an output bounded beteween -1 and 1. We choose the angular velocity of the robot to be between $-\Pi$ and Π so we multipiled the result of the output layer to get the desired range. Same goes for the linear velocity we have maped the range from -1 and 1 to 0.2 to 0.5.

And to enable better exploration, we have added the OuNoise to the output of

the actor network when taking actions which will sample a range of values from a unifrom distribution.

We have also used batch normalization layers [20] which a technique for normilizing the output for each layers that have an effect of stabilizing the trainging process and making it converge faster especily if the initial distribution of the input data is varying. It also acts as regulizer so no need for dropout layers.

As For the critic, he gets as input both the state and actions and outputs the q-value. Firstly, we pass the states and actions into seperate layers with number of units of 512 and 256 respectevely. Then, we concatinate them into one layer and add two other hidden layers with 256 number of units with a dropout of 0.2.

For loss calculation, we will be using target networks like described in the algortihms and updating them using soft update with parameters tau of 0.01

The critic loss, will be between the predicted value of the critic network and y:

$$closs = loss(critic(state, actions), reward + \gamma * targetcricic(nextstates, nextactions) * (1 - dones). \tag{5}$$

where the loss function is the mean squared error.

For the actor, it will be the mean predicted value of the critic network multiplied by - as he is trying to maximize the q-value.

$$aloss = -mean(critic(states, actions)) \tag{6}$$

We have used the Adam optimizer for both networks with learning rates of 0.01 and 0.001 for the critic and actor respectively. We set the batch size to 128 and the minimun size of the replay buffer before training start to 2500 samples.

5.6 Ros implemation

In this section, we will describe how we have implemented our solution in ros, how we defined the different nodes, topics and services that makes the system.

The work is divided into phases. The first phase is the training and test phase and the second one is the deployment phase.

In the first phase we implemented the solution in ros in a way to facilitate the training process that will require a lot of modifcation of code and simulation replay.

For the second phase, and after the models are trained we will deploy them in each robot and each robot will choose his actions based on his model, and also can communicate with the central robot.

5.7 Ros implemation: Training phase

We have followed the openai gym implemation of diffrent rl problms using the python programming language.

In the training phase, we have defined two nodes and one python class for the model.

The first node is the environemnt node. This node is responsible for getting sensor data, postions and command robots to move in the simulation.

The node have one service, subscribe to two topics and publishing to one. The fisrt topic he subscibe to is the Laser topic which gets the laser mesurments of each robot every step. the data retunred by this topic is an array where each element of it contains the distance of the detected object and the index of this element is the angle of detection.

The second topic he susbcribe to is the positions topic, which will get the x,y cordinates of the robots and also their orientation. For the topic, the node will publish to it insted of subscbing. he will publish him the actions that the robot must take based on the output of the neural network.

The service that this node offer for the main node is a function that will take that actions that the robots should take , get the states,calculates the rewards and then passing back the response to the main node.

This node is also responsible for calculating the rewards based on the state.

We called the second node the main node. This node is responsbile to start the main traning loop,getting actions from the models passing them to the environement node in order to reacive the next states and the rewards, he then store these samples in the replay memory and finally train the models. The loop will continue until the models converge and the shape is been formed. This node will connect to a python class which contains our models.

This python class defines the models and the networks architecture, contains the replay memory and describes the trainnig process.

References

[1] F. Rubio, F. Valero, and C. Llopis-Albert, "A review of mobile robots: Concepts, methods, theoretical framework, and applications," *International Journal of Advanced Robotic Systems*, vol. 16, no. 2, p. 1729881419839596, 2019.

[2] L. Bayındır, "A review of swarm robotics tasks," *Neurocomputing*, vol. 172, pp. 292–321, 2016.

[3] L. Bayindir and E. Şahin, "A review of studies in swarm robotics," *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 15, no. 2, pp. 115–147, 2007.

[4] I. Navarro and F. Matía, "An introduction to swarm robotics," *Isrn robotics*, vol. 2013, pp. 1–10, 2013.

[5] T. Arai, E. Pagello, L. E. Parker, *et al.*, "Advances in multi-robot systems," *IEEE Transactions on robotics and automation*, vol. 18, no. 5, pp. 655–661, 2002.

[6] E. Şahin, "Swarm robotics: From sources of inspiration to domains of application," in *Swarm Robotics: SAB 2004 International Workshop, Santa Monica, CA, USA, July 17, 2004, Revised Selected Papers 1*, pp. 10–20, Springer, 2005.

[7] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo, "Swarm robotics: a review from the swarm engineering perspective," *Swarm Intelligence*, vol. 7, pp. 1–41, 2013.

[8] I. Olaronke, I. Rhoda, I. Gambo, O. Ojerinde, and O. Janet, "A systematic review of swarm robots," 2020.

[9] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A brief survey of deep reinforcement learning," *arXiv preprint arXiv:1708.05866*, 2017.

[10] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, pp. 279–292, 1992.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[12] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[13] E. A. O. Diallo and T. Sugawara, "Multi-agent pattern formation: a distributed model-free deep reinforcement learning approach," in *2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2020.

[14] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning," in *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 6252–6259, IEEE, 2018.

[15] J. Ma, H. Lu, J. Xiao, Z. Zeng, and Z. Zheng, "Multi-robot target encirclement control with collision avoidance via deep reinforcement learning," *Journal of Intelligent & Robotic Systems*, vol. 99, pp. 371–386, 2020.

[16] Y. Dong and X. Zou, "Mobile robot path planning based on improved ddpg reinforcement learning algorithm," in *2020 IEEE 11th International Conference on software engineering and service science (ICSESS)*, pp. 52–56, IEEE, 2020.

[17] H. Gong, P. Wang, C. Ni, and N. Cheng, "Efficient path planning for mobile robot based on deep deterministic policy gradient," *Sensors*, vol. 22, no. 9, p. 3579, 2022.

[18] F. Quiroga, G. Hermosilla, G. Farias, E. Fabregas, and G. Montenegro, "Position control of a mobile robot through deep reinforcement learning," *Applied Sciences*, vol. 12, no. 14, p. 7194, 2022.

[19] https://emanual.robotis.com/, "Gazebo." `https://emanual.robotis.com/`.

[20] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, pp. 448–456, pmlr, 2015.

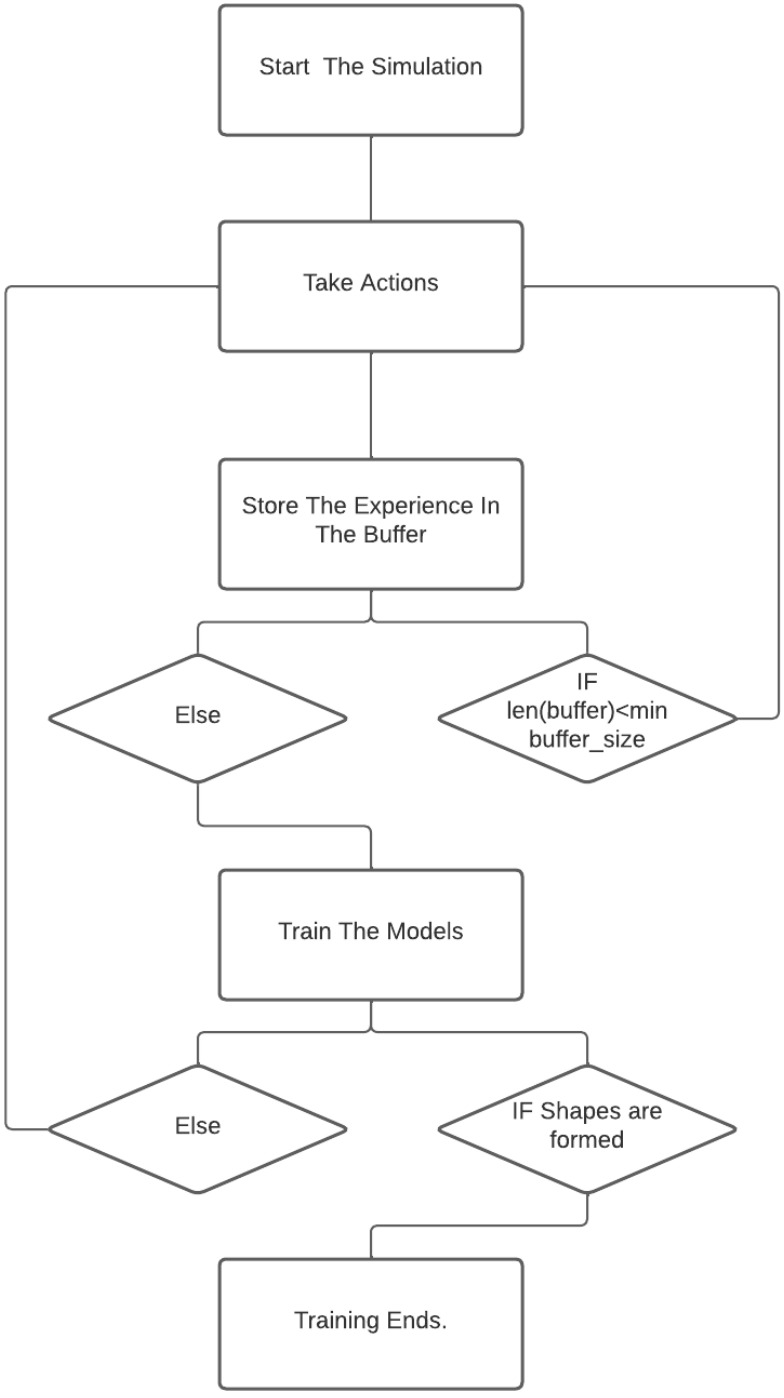

Figure 3: training flow

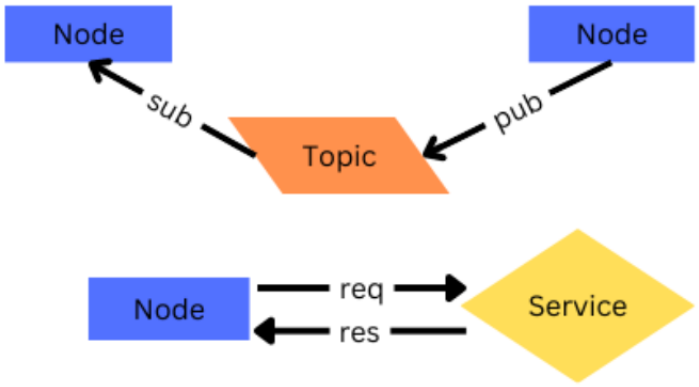

Figure 4: Ros

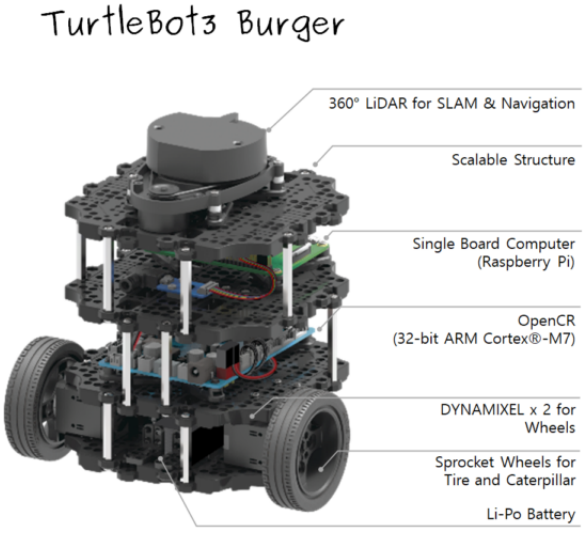

Figure 5: TurtleBot3 [**ref**:https://emanual.robotis.com/]]

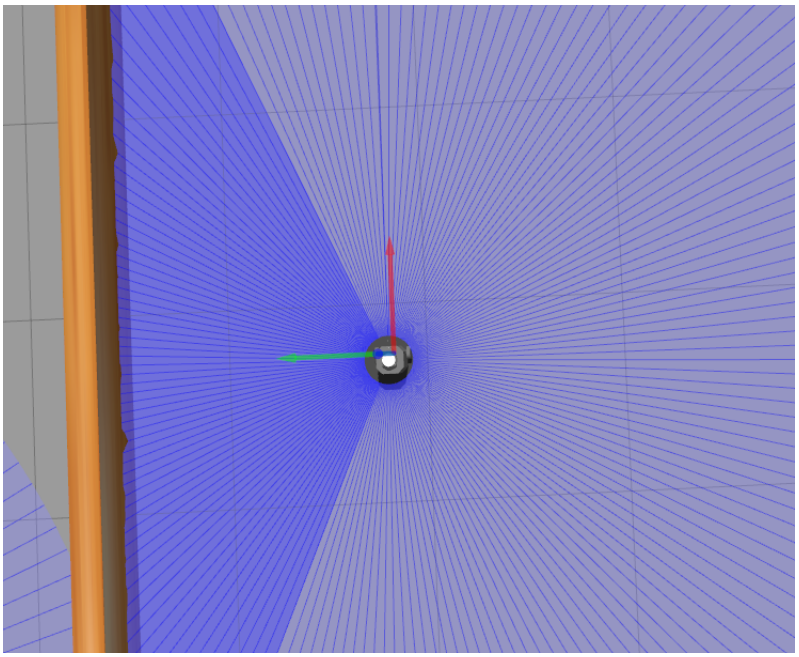

Figure 6: TurbleBot3 Lds

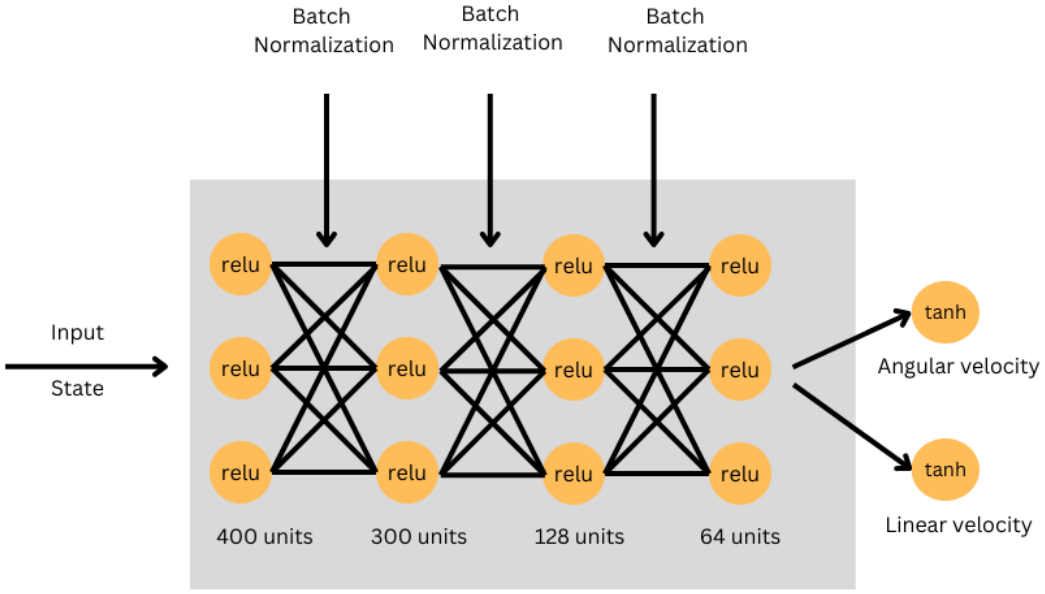

Figure 7: actor network

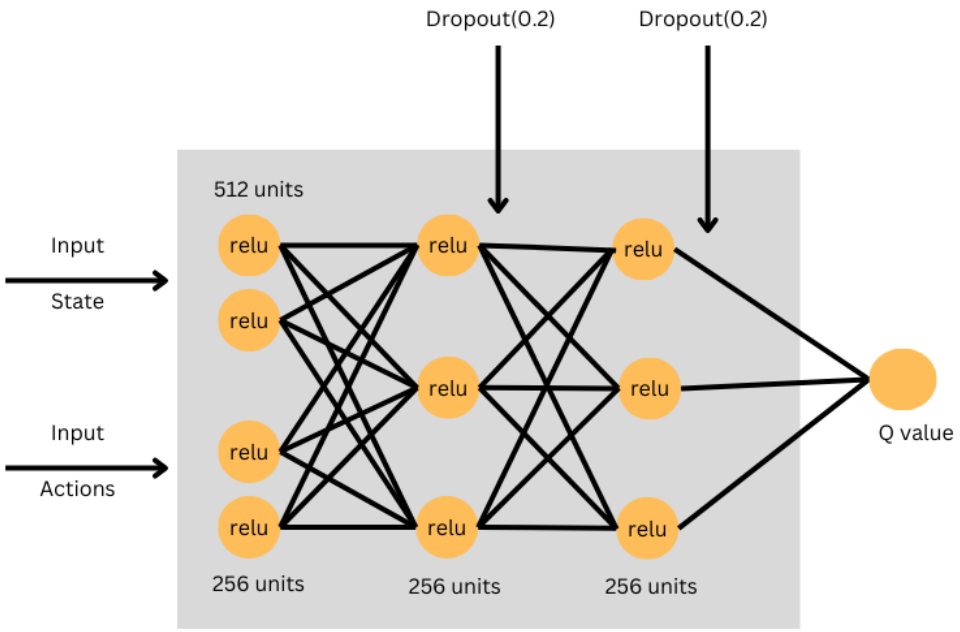

Figure 8: critic network

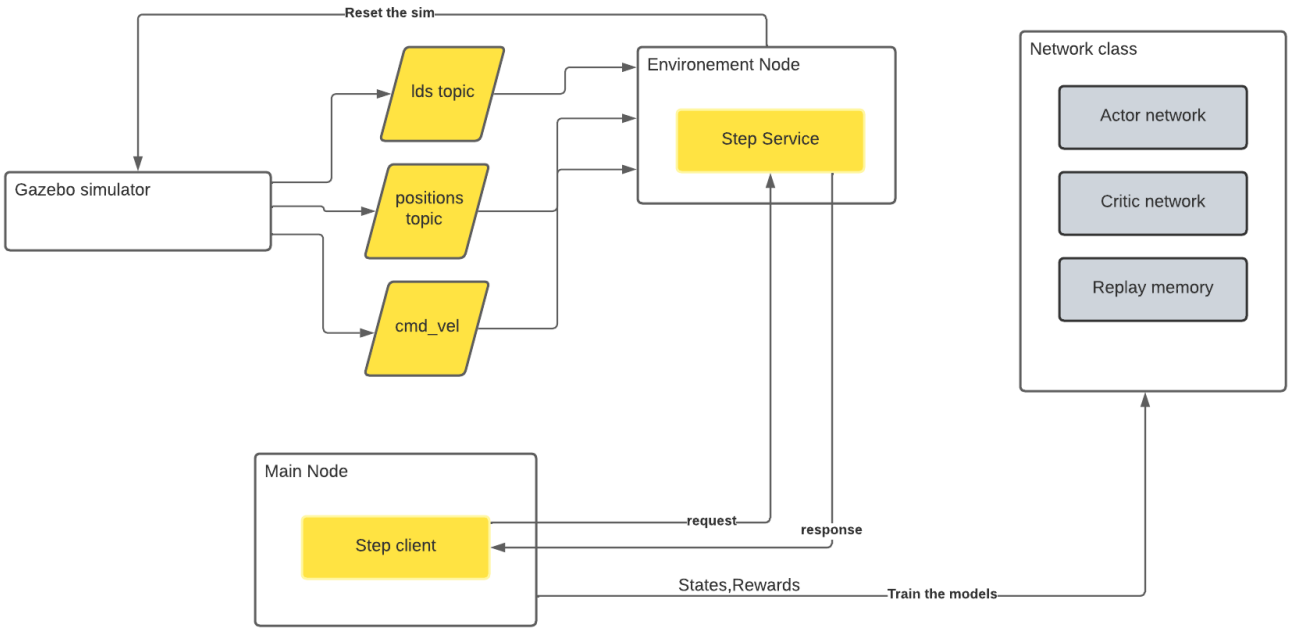

Figure 9: training