Contents

1 Introduction **3**

1.1 Introduction 4

2 Swarm Robotics **6**

2.1 Swarm Intelligence and Social animal inspiration 7

2.2 Swarm robotics and multi robot systems 7

2.3 Swarm robotics 7

2.4 Swarm robotics properties 8

2.5 Domain of application 8

2.5.1 Tasks that cover a region 9

2.5.2 Search and rescue missions 9

2.5.3 Cleaning of oil speals 9

2.5.4 Exploration 9

2.5.5 Agriculture 9

2.6 Swarm robotics basic tasks and problems 9

2.6.1 Aggregation 9

2.6.2 Dispersion 10

2.6.3 Pattern formation 10

2.6.4 Cordinated movement 10

2.6.5 Hole avoidance 10

2.6.6 Foraging 10

3 Deep reinforcement learning **11**

3.1 Introduction 12

3.2 Machine learning 12

3.3 Reinforcement learning 12

3.4 Q learning 13

3.5 Deep Q learning 14

3.6 Deep Deterministic policy gradient 15

3.7 Related Work 16

4 Problem formulation And Solution Conception **18**

4.1 Introduction 19

4.2 Goal defintion 19

4.3 Problem Formulation 19

4.3.1 Environement 19

4.3.2 Shape generation 20

4.3.3 Shape formation Algorithm 20
4.3.4 Explanation 20
4.4 State Representation 21
4.4.1 Introduction 21
4.4.2 State Representation 21
4.5 Reward Function Design 22
4.6 Shape Formation And Deep Reinforcement Learning 24
4.6.1 Introduction 24
4.6.2 The learning flow 24

5 Implementation **27**
5.1 Introduction 28
5.2 Roboting Operating system 28
5.2.1 Ros Basics 28
5.3 TurtleBot 29
5.3.1 TurtleBot3 components 29
5.4 Simulation 30
5.5 Ros implemation 31
5.6 Ros implemation: Training phase 32
5.7 Ros implemation: Test phase 33
5.8 Neural network implementation 34

6 Result and analysis **38**
6.1 Introdution 39

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].

This feild has mainly focus on a set of probelms that normaly a swarm of robots face. The mais porblems are: aggregation, dispersion, pattern formation, cordinated mouvement, hole avoidance, foraging.

The problem that will be studied in this thesis is the pattern formation. Where the agents (robots) try to form diffrent gemoetric shapes like squeres,trinagles and circles in order to perform a specific task like lifting somthing, encirling some target or for rescue operations.

We can solve this problem using two different aproches. The first one is a completely centrelized method where there exist a centel unit which controles the swarm in all their actions and have global state access. However, this can be easy and much simpler but suffer but it is less robust if the central unit fails. The second aproch is a dicentrilzed one, where each robot have uses local commucintion and have access only to his local state which is much scalable and robust but is more complex and require more time.[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. we will use not use a complety centrlized method where each robot will move autonomusly but a central robot is needed to monitor and cordinate the others. We will be using deep reinforcement learning algorithms to train the agents to achive their task.

This thesis is organised as following: In the first chapter we will define swarm robotics, it's domains, application and common probelms it's try to solve.

In the second chapter, we will define reinfocement learning, the most used algorithms in this domain and how they work, finnaly how diffrent papers have appilied these algothims in learing navigating autonomous robots.

The thrid chapter will discuss the problem formulation in reinforcement learing and how we have designed our solution.

The fourch chapter will be about the implemation of our solution in the Ros framwork and the gazebo simulator, and how we set networking between the robots.

Finnaly, the fifth chapter is the result analysis of our experiments and cite the goals we have achived and the problems that consists.

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 capacity of the swarm to continue operating even if certain group members die or some system components fail.
Scalability: The system's capacity to function successfully in both small and big group sizes without affecting the swarm's performance.

Flexibility The swarm's capacity to govern and adapt to environmental changes.
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

Swarm robots is most suited for handling issues that encompass an area of the space because of the swarm's extensive sensing capabilities, such as monitoring the environment of a lake or surveillance of a particular region.[6][8]

2.5.2 Search and rescue missions

Swarm robots can be used for these sorts of tasks in various accidents or catastrophes, such as earthquakes, when human involvement is challenging. These robots include the M-TRAN, Polyobot, and Swarm Bot.

2.5.3 Cleaning of oil speals

Such problems may be handled more quickly and affordably with a swarm of robots. Seaswarm, a robot created by the senseable team at MIT, is an example of one used for such activities.

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 are used to enhance agriculture and keep an eye on the health of crops.

2.6 Swarm robotics basic tasks and problems

SR tasks may be reduced to simple operations that the swarm often performs in an effort to achieve its goal. Aggregation, dispersion, pattern development, coordinated movement, hole avoidance, and foraging are among these activities.[3][4].

2.6.1 Aggregation

The robots are combined in order to complete a task or share information. In a centralized system, this task could be simple, but in a decentralized one, it might be challenging.

(a) fish swarm

(b) birds swarm

Figure 1: swarm animals

2.6.2 Dispersion

Sometimes, in order to increase the group sensing capacities during exploration, the swarm must span a large region without losing contact with its members.

2.6.3 Pattern formation

In order to carry things or navigate hallways, the swarm must sometimes create precise patterns such as circles, squares, or lines.

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.
ref:https://www.nationalgeographic.com/]] https://www.researchgate.net/

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]

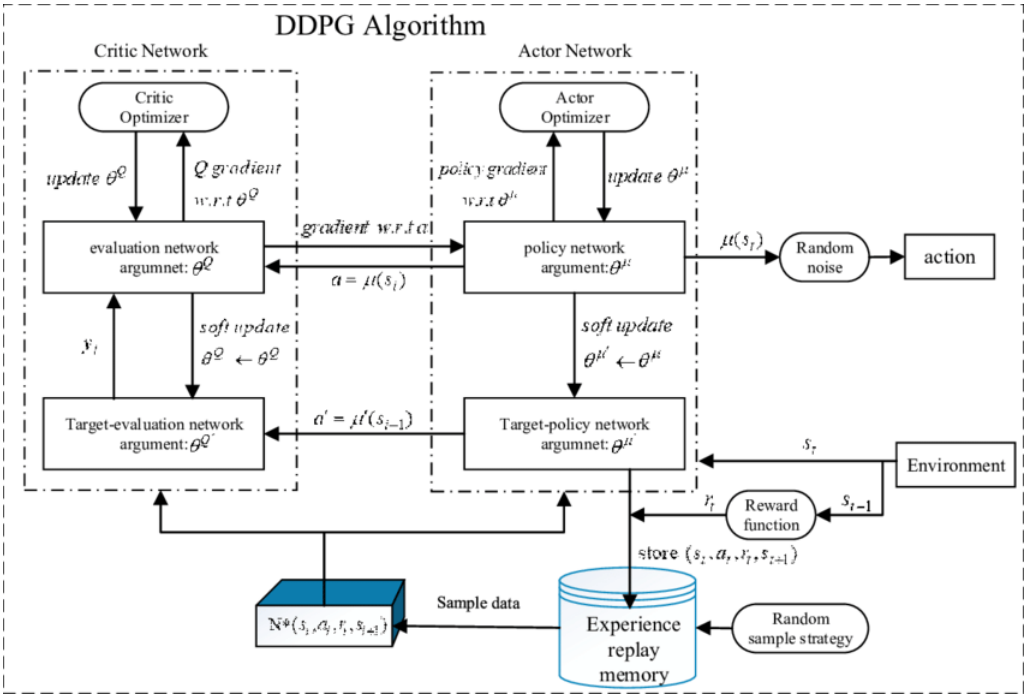

Figure 2: ddpg **ref**:https://www.researchgate.net/]]

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 environment as an MDP where S are the set of states that each robot will receive. A are the set of actions that it can take , 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 collisions with the wall and other robots. We will use the DDPG algorithm to control the robots in a continuous manner.

For the environment, we will Consider a scenario in which there are n ($n > 2$) robots moving in a 2D environment. Each robot will try to reach his target position which is described by the homogeneous coordinates (x,y) that is be set by the central robot.

Figure 3: formation

4.3.2 Shape generation

In order to form the different shapes, The central or leader robot will first go to a point in the space and from it , he will inform other robot with the desired shape to be formed by giving them their target position relative to his position by being at a certain distance and angle.

4.3.3 Shape formation 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 chooses a random postion and stay on it.
The central robot broadcasts a message containing his position with the desired shape by using generaing a set of cooardinates that each robot must reach.
while $c! = N$ **do**
    for robot in robots **do**
        Take an action.
        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

4.3.4 Explanation

The first robot will choose a random position and place himself in it. Then he broadcast a message 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 is received by other robots, each one of them will try to reach the target goal autonomously by avoiding collision 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 and declare the task as complete if the shape is formed correctly .

4.4 State Representation

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

The state of the robots will be the inputs for our neural networks and based on them, we can calculate the rewards that the robot will get when acting in the environment.

In our implementation we have defined six kind of state information, the distance to goal, angle to goal, angular velocity,linear velocity ,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 lds mesurments**: In order to detect other robots and avoid collision, we use an LDS which gives us an array of elements where each value is the minimum distance detected and each index is the angle of detection.

  The orinal lds take 360deg mesurement all around the robot with a max length of 3.5 meters. But in our problem we don't need such precision as we have constased that just 40 samples are enough because there are no thin or small object that require such precision. In addition, we also take just 20 of them which will presnt the front of the robot.

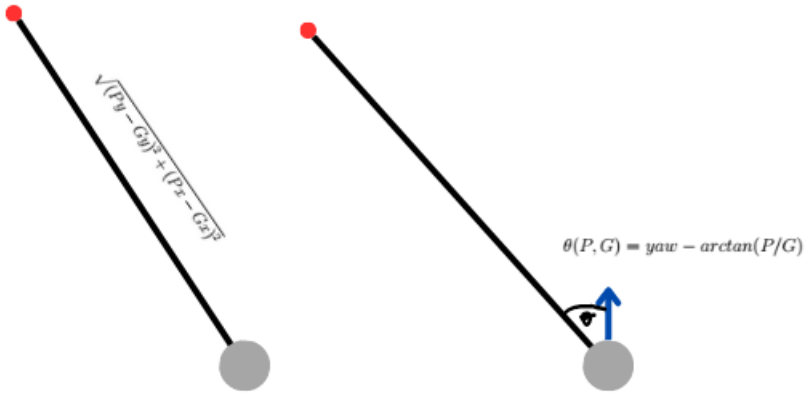

Figure 4: equations

- **The angular velocity**: It is the angular or rotational movement of the robot, and it's between $-\Pi$ and Π.
- **The linear velocitiy**: it is the linear speed of the robot . and we have set it between 0.2 and 0.5 m/s.

So, our final state representation is an array of :
$state = [distance, angle, angularvelocity, linearvelocity, lidarmesurments]$
where lds is a 20 element array.

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 encourage and reward the robot more, when he approaches the goal target, and penalize 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 is the distance reward. We will give the robot more reward when he aporoches the target goal and

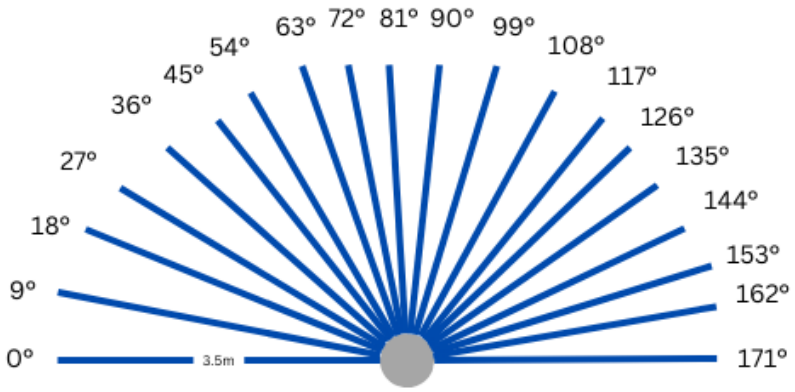

Figure 5: lds mesurements

less when he doesn't:

$$R_D = -D_{goal} \tag{1}$$

Where D_{goal} is the changing distance between the robot and the target goal,

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

$$R_\theta = -\theta_{goal} \tag{2}$$

where θ_{goal} ranges between $-\pi$ and π.

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

$$R_c = \begin{cases} -10, & \text{if } mdd < 0.6m. \\ -500, & \text{if } mdd < 0.13m \text{ (a crash)}. \\ 0 & \text{else.} \end{cases} \tag{3}$$

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 Shape Formation And Deep Reinforcement Learning

4.6.1 Introduction

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

The parameters of both networks will affect how the performance of the robots and the speed of convergence. we will describe in details the parameters that we have chosen to be good for achieving the task.

4.6.2 The learning flow

Here, we will describe the learning flow that the robots will go through to learn how to aproach their goals.

It starts by the robot being in a given state, he then takes an action, reacive sensor data from the sensors like lds and imu, then based on them we calculate the reward function and finaly the robot transition to a new state.

This information (state,action,reward,next state) will be stored in a replay buffer to sample from it during the training of the models , and once he reaches 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 overfitting a certain type of shapes.

And if they collide the simulation is reset also 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.

To provide the robots a notion of what it's like to attain a goal in terms of actions to do and states to be in, and to facilitate exploration, we assign relatively basic objectives to be completed in the beginning, and after our robots are familiar with achieving their destinations, we raise the challenge and randomize the target goals.

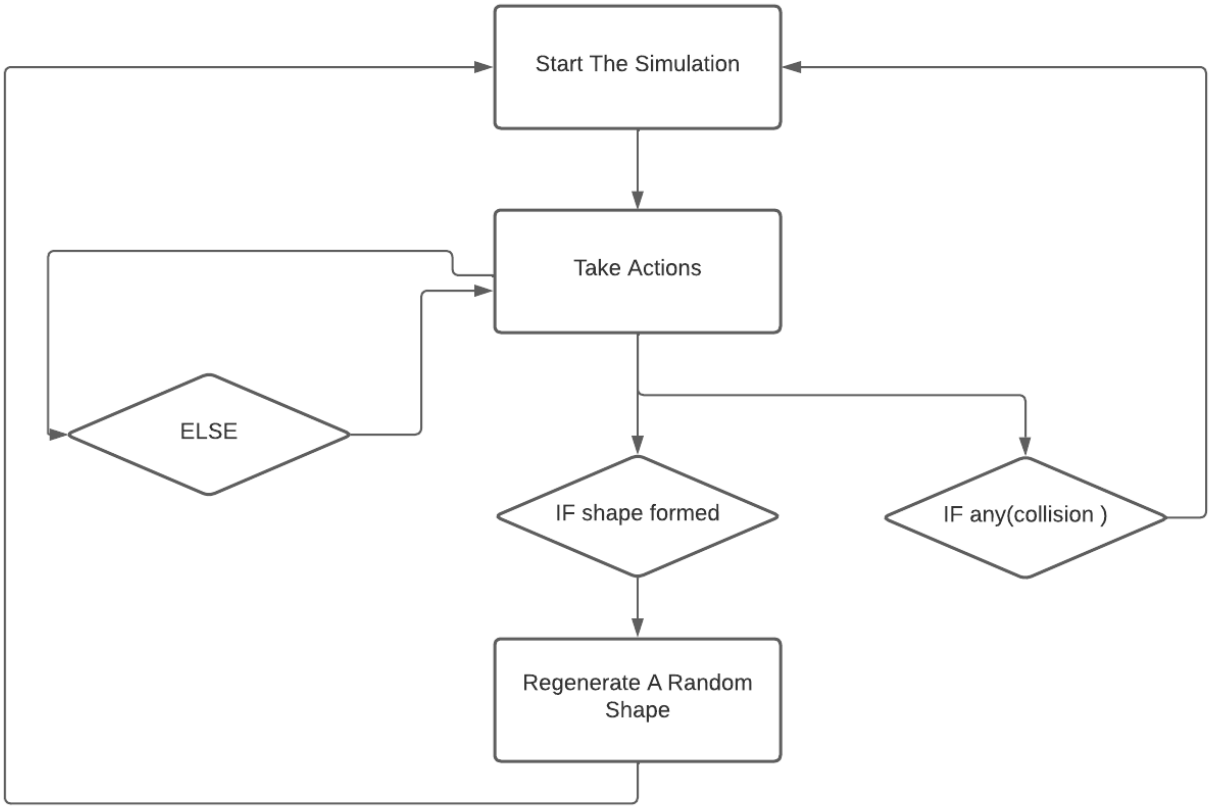

Figure 6: simulation low

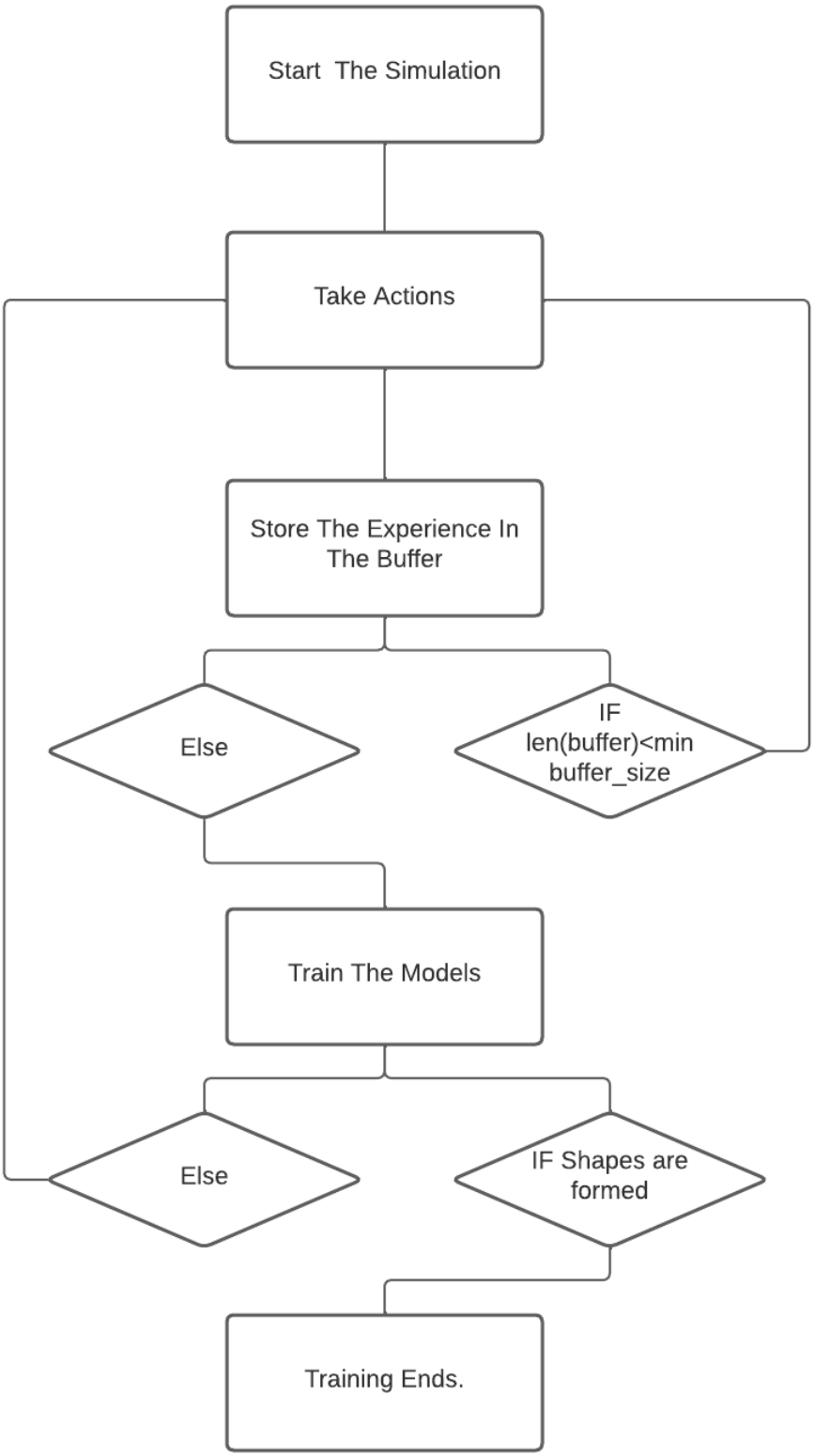

Figure 7: training flow

5 Implementation

5.1 Introduction

In this section, we will explain how we have implemented the solution , how the neural networks are designed, 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 each other using topics and services.

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

- **Topics**: ttopics 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 they will receive. For instance, we can define topics to accept only string ,numbers, arrays or we can define our custom data type..

- **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.

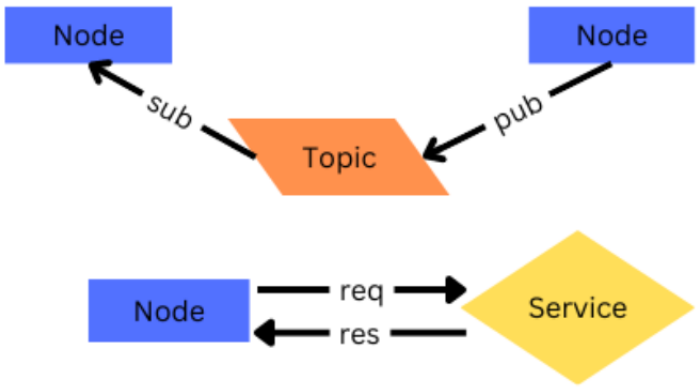

Figure 8: Ros

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. The same organization behind ros and the gazebo simulator.

This robot is used for educational,research and prototyping purposes, 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

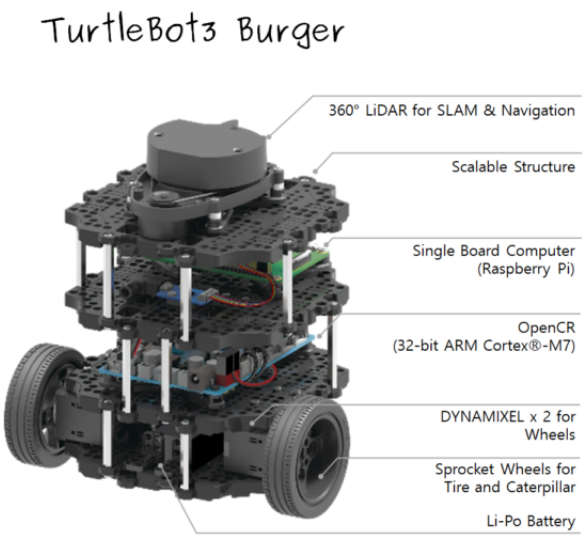

Figure 9: TurtleBot3 [**ref**:https://emanual.robotis.com/]]

this project can sense object in 360 degree with a maximum length of 3.5 meters.

The (LDS) was one of the data source 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 calculated relative to a fixed reference frame which either define by the system designer like a closed room or factory or by the robot himself in outdoor environment.

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 repeatable way, let us put the robots and the environment in different conditions for faster experimentation.

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 fee.

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

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

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 to build custom ones. Gazebo is also able to simulate real world physical phenomena, including physics-based motion, collision detection, and sensor simulation.

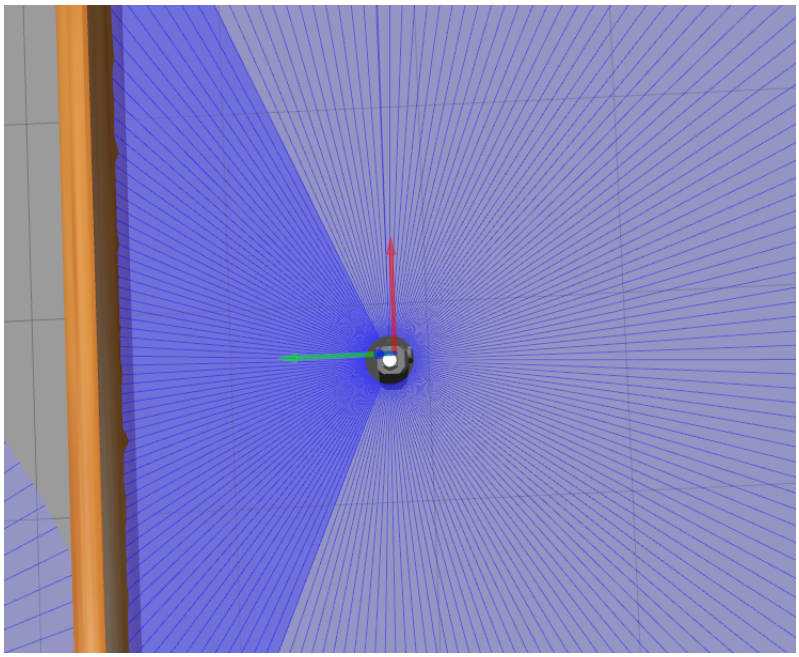

Figure 10: TurbleBot3 Lds

5.5 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 up the system.

The work is divided in two phases. The first is the training and test phase and the second 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 modification of code and simulation replay. In this phase, we haven't set the networking between the robot as we assume that each robot knows his target position and we assume that we can access the global state.

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 will communicate with the central robot for coordination.

5.6 Ros implemation: Training phase

We have followed the openai gym implementation of rl problems using the python programming language.

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

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

This node have one service, subscribe to two topics and publish to one. The first topic he subscribe to is the Laser topic which gets the laser measurements of each robot every step. The data returned 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 subscribes to is the Odometry topic, which will get the x,y coordinates of the robots and also their orientation. For sending actions to be excuted by the robots, this node will publish to a ROS defined topic called "cmd vel" which accepts a message that contains the linear and angular velocity of the robot.

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

The second node we defined is called the main node. This node is responsible of starting the main training loop, getting actions from the models passing them to the environment 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 shapes are 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 training process.

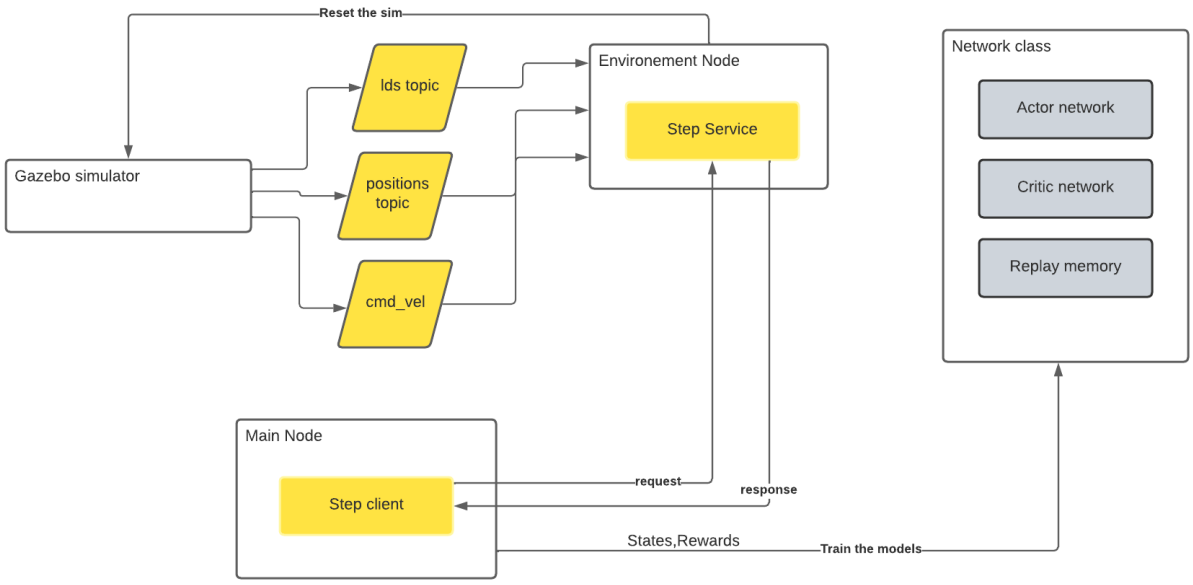

Figure 11: training

5.7 Ros implemation: Test phase

In this phase, and after the training of the models is done, we will deploy just the actor models on each robot (no need for the critic as no training is happening) in order to predict actions based on the input state.

We have set a node for each robot, therefore the communication will take place between these nodes by defining custom topics. Each node will get input state and pass them to the actor model to predict the actions to take.

We have defined one topic called the goals topic in which the central robot publish the goals positions and others subscribe to it. We have also defined a service in the central robot that others will send request to either inform him about their current position or if they have reached the target goal in order for the central robot to monitor the state of the formation.

Once all robots are in place, the central robot can then check if the shape is properly formed to declare finally that the task is done by broadcasting a message to other robots in the team.

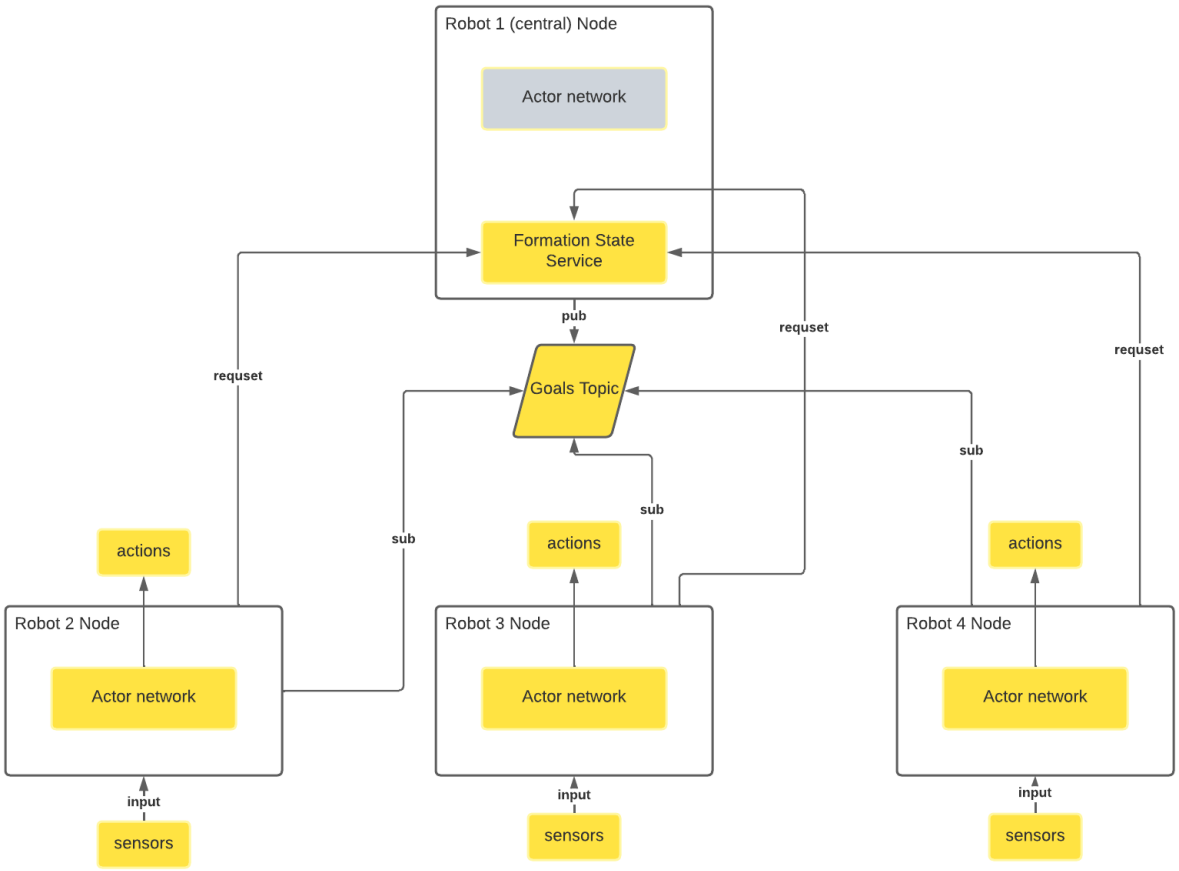

Figure 12: deployment

5.8 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 choose and explain the reason behind them.

The first thing we have done is normalizing the input data, to achievce better convergne and stability of our models. We have nomilized our data to be between 0 and 1 for all the inputs: Distance normalization:

$$norm_d = distance/dist_{diagonale} \tag{5}$$

where $dist_{diagonale}$ is the diagonal distance of the environemnt which is the maximun distance.

Angular velocity and angle to goal normalization:

$$norm_\theta = angle + \Pi/2\Pi \tag{6}$$

And finnaly the lds mesurements:

$$norm_{lds} = lds/3.5 \tag{7}$$

Where 3.5m is the max length of the lidar sensor.

For the actor network, we set four hidden layers where each layer has 400, 300, 128 and 128 units respectively with dropout of 0.3 in the first two layers. Each layer is using the relu activation function and was initialized with Xavier Glorot's expect for the output layers, where they were initialized using a uniform distribution. 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 gives 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 uniform distribution.

We have also used batch normalization layers [20] which a technique for normalizing the output for each layers that have an effect of stabilizing the training process and making it converge faster especially if the initial distribution of the input data is varying.

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 1024 number of units with a dropout of 0.2 and l2 regularization of 0.01.

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:

$$closs = loss(critic(state, actions), reward + \gamma * targetcricic(nextstates, nextactions) * (1 - dones). \tag{8}$$

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.

$$aloss = -mean(critic(states, actions)) \tag{9}$$

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 minimum size of the replay buffer before training start to 3500 samples.

```
def create_actor_model(self):

    init1 = tf.random_uniform_initializer(minval=-0.003, maxval=0.003)
    init2 = tf.random_uniform_initializer(minval=-0.0003, maxval=0.0003)

    inputs = keras.layers.Input(shape=(34,))
    out = keras.layers.Dense(400, activation="relu",kernel_initializer=keras.initializers.GlorotNormal())(inputs)
    out=keras.layers.Dropout(0.3)(out)
    out = keras.layers.BatchNormalization()(out)
    out = keras.layers.Dense(300, activation="relu",kernel_initializer=keras.initializers.GlorotNormal())(out)
    out=keras.layers.Dropout(0.3)(out)
    out = keras.layers.BatchNormalization()(out)
    out = keras.layers.Dense(128, activation="relu",kernel_initializer=keras.initializers.GlorotNormal())(out)

    out = keras.layers.BatchNormalization()(out)
    out = keras.layers.Dense(128, activation="relu",kernel_initializer=keras.initializers.GlorotNormal())(out)

    out = keras.layers.BatchNormalization()(out)

    outputs = keras.layers.Dense(1, activation="tanh",kernel_initializer=init1)(out)

    outputs2 = keras.layers.Dense(1, activation="tanh",kernel_initializer=init2)(out)
```

Figure 13: Actor Model

```
def create_critic_model(self):
    state_input = keras.layers.Input(shape=(34))
    state_out = keras.layers.Dense(512, activation="relu",
                                   kernel_initializer=keras.initializers.GlorotNormal())(state_input)

    action_input = keras.layers.Input(shape=(2))
    action_out = keras.layers.Dense(256, activation="relu",
                                    kernel_initializer=keras.initializers.GlorotNormal())(action_input)

    concat = keras.layers.Concatenate()([state_out, action_out])

    out = keras.layers.Dense(1024, kernel_regularizer=keras.regularizers.l2(0.01),
                             activation="relu",kernel_initializer=keras.initializers.GlorotNormal())(concat)
    out=keras.layers.Dropout(0.2)(out)

    out = keras.layers.Dense(1024, activation="relu",kernel_regularizer=keras.regularizers.l2(0.01),
                             kernel_initializer=keras.initializers.GlorotNormal())(out)
    out=keras.layers.Dropout(0.2)(out)

    outputs = keras.layers.Dense(1)(out)

    model = tf.keras.Model([state_input, action_input], outputs)

    return model
```

Figure 14: Critic Model

```
with tf.GradientTape() as tape:
    target_actions = self.target_actor(next_state_batch, training=True)
    target_actions=tf.concat(target_actions,axis=1)

    y = reward_batch + self.discout_factor * self.target_critic(
        [next_state_batch, target_actions], training=True
    )*(1-dones)


    critic_value = self.critic_model([state_batch, action_batch], training=True)

    critic_loss = loss_function(y,critic_value)



critic_grad = tape.gradient(critic_loss, self.critic_model.trainable_variables)
self.critic_optimizer.apply_gradients(
    zip(critic_grad, self.critic_model.trainable_variables)
)
with summary_writer.as_default():
  tf.summary.scalar(f'loss_critic-{self.name}', critic_loss, step=self.critic_optimizer.iterations)
```

Figure 15: Critic Training

```
with tf.GradientTape() as tape:
    actions = self.actor_model(state_batch, training=True)
    actions=tf.concat(actions,axis=1)
    critic_value = self.critic_model([state_batch, actions], training=True)


    actor_loss = loss_actor(critic_value)


actor_grad = tape.gradient(actor_loss, self.actor_model.trainable_variables)
self.actor_optimizer.apply_gradients(
    zip(actor_grad, self.actor_model.trainable_variables)
)
with summary_writer.as_default():
  tf.summary.scalar(f'loss_actor-{self.name}', actor_loss, step=self.actor_optimizer.iterations)
```

Figure 16: Actor Training

6 Result and analysis

6.1 Introdution

In this last section, we will describe the results that we get from our experiment and discuss them.

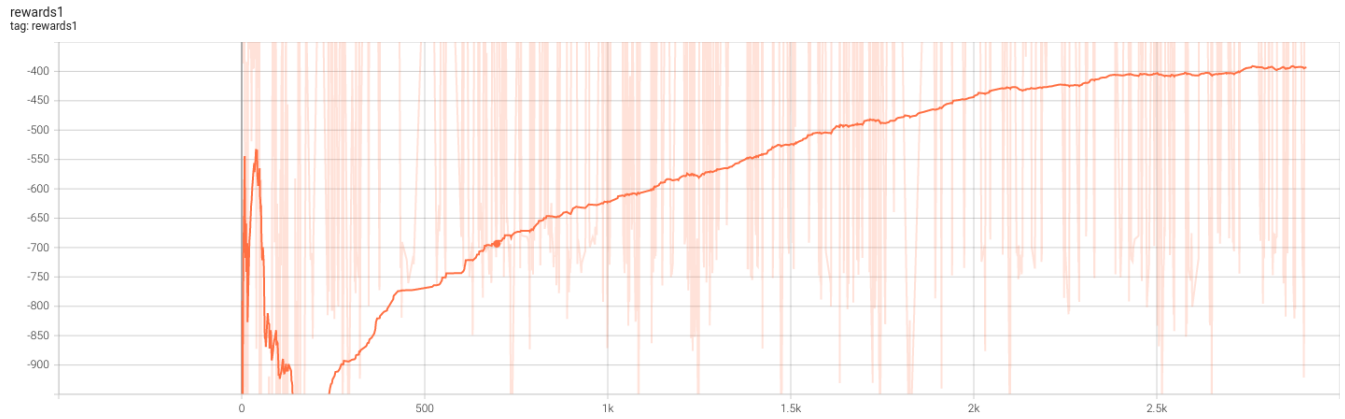

Figure 17: robot1

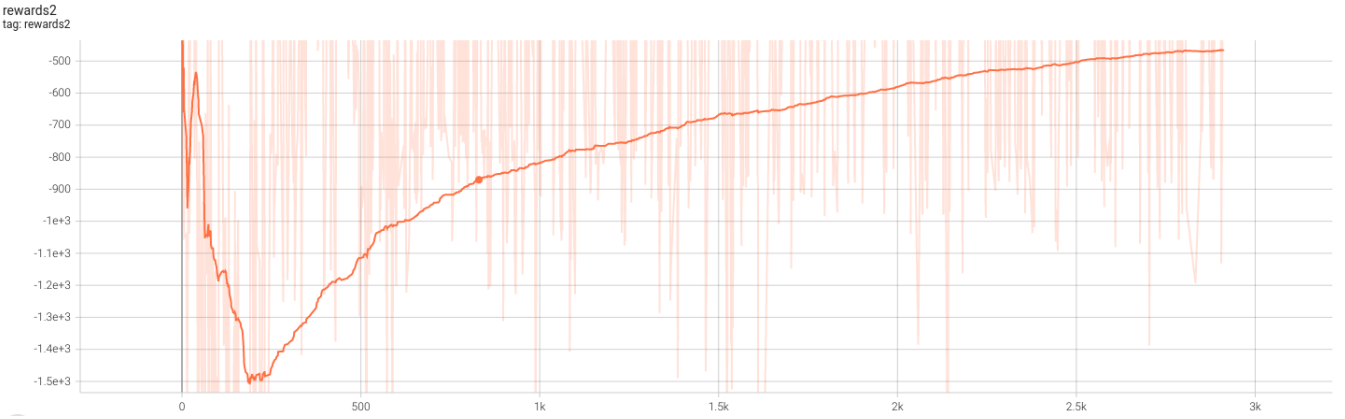

Figure 18: robot2

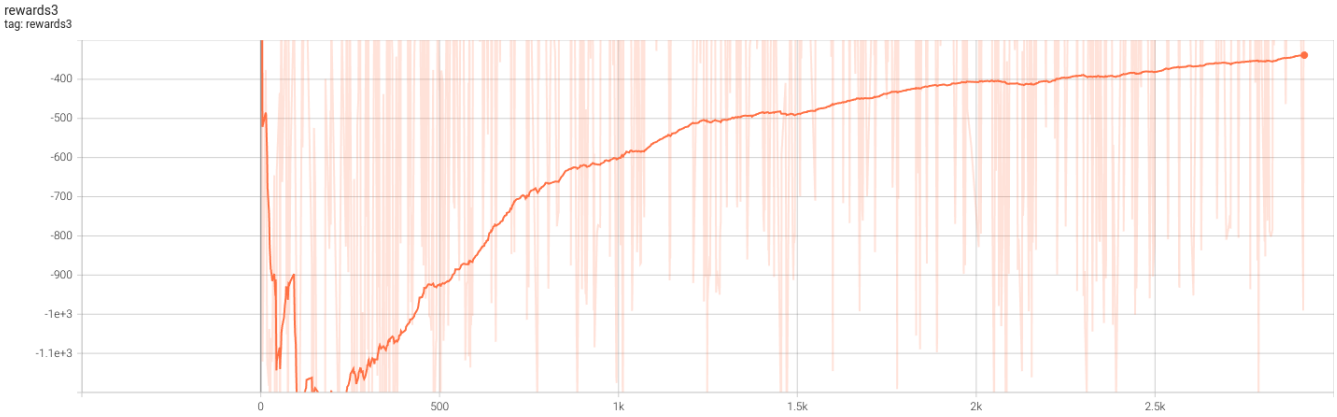

Figure 19: robot3

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.