Scalar Field Guidance in Swarms of Simple Robots

by

©Dalia Shouman El Shahat Ibrahim

Thesis submitted to the School of Graduate Studies in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Department of Computer Science
Memorial University

April 2024

St. John's, Newfoundland and Labrador, Canada

Abstract

Developing swarms of robots as simple agents without any central controller is a challenging task. We focus on simple robots with limited capability which do not have a localization system to show them where they are. Termite construction activity inspires our work; although they have poor vision, they can construct a massive termite mound, build a royal chamber and communicate with each other by observing the changes in the environment, sensing vibrations and detecting chemical substances (pheromones).

In our work, we used scalar fields as a global level of guidance for the robots. A scalar field associates a value with every point in the working region. We show how scalar fields can be used to guide a low-cost and limited-capability swarm of robots to execute a specific task. We present four examples of tasks using the scalar field, such as constructing shapes from ambient objects, finding the largest coverage-connected network among the robots, aggregating to a predefined area and foraging by finding and collecting ambient objects to the collection area.

This work is divided into three parts; first, we show how the scalar field with different resolutions can help divide labour among the robots and guide them in their movements. Second, we investigate combining a scalar field with reinforcement learning to find the largest coverage network. Finally, we design a coloured scalar field and use it as a road network for the swarm to reduce spatial interference among the robots. We practically build our robots based on the Zumo robot kit to perform the aggregation task.

Acknowledgments

After thanking Almighty "ALLAH" for his blessing and guidance to complete this work, I would like to express my deepest gratitude to my supervisor, Professor Andrew Vardy, for his support, valuable guidance, and insightful advice throughout my academic journey. I am truly grateful for his mentorship and the inspiration he has provided me with over the years. Also, I had the pleasure of collaborating with my colleague, Waleed Sherwani, for his help initiating the hardware experiments.

I am indebted to my loving husband, Hafez Seliem, whose belief in my capabilities has been a source of strength and motivation during the challenging phases of my studies. I am forever grateful for his love and support. To my dear daughter, Menna, your boundless joy and unconditional love have been my constant motivation and the driving force behind my endeavors. Your innocent laughter have brightened my days, making this work a more meaningful and fulfilling journey. Lastly, I would like to extend my heartfelt appreciation to my parents and family for their endless encouragement and sacrifices throughout my life. Their unyielding support, wise counsel, and belief in my abilities have been the cornerstone of my success.

These amazing people are my collective support and encouragement that have enabled me to overcome obstacles and reach the completion of this thesis. I am truly blessed to have such an incredible support system, and I cannot express enough gratitude for their unwavering presence in my life.

Table of Contents

List of Figures

Chapter 1

Introduction

Developing swarm robots is challenging, especially when these robots are simple agents with limited capability to adapt themselves to achieve a given mission. These robots have simple low-level actions and produce collective high-level results like aggregating in a specific goal, finding the largest coverage area in the working environment, or constructing shapes from ambient objects.

The robots in our swarm depend only on their local sensing without any central controller. The coordination among them was achieved indirectly by observing the shared environment (stigmergy). Relying on simple agents makes our system robust because there is no single-point failure, and is scalable (we can increase/decrease the number of robots without any modifications).

We are inspired by social insects like ants, termites, wasps, and bees because these insects use pheromones and other cues to guide them to perform different activities in their societies [1]. For example, in building the royal chamber for the queen termite, the queen discharges a particular type of pheromone as a template to guide the worker termites to deposit the mud at a specific distance around it [2]. Similar to pheromones in insects, we present the scalar field as a global signal to guide a robot swarm in their work. Assigning each point in the working area with a singular value, like measuring the

temperature at every point in the room, indicates where the heater is. These measurements are considered a static scalar field; incorporating scalar fields to guide the robots' movements would more closely resemble pheromone use in social insects. The difference is that the insects produce pheromones, but in robots, the scalar field will be provided to them.

1.1 Problem Statment

Carvalho et al. [3] analyze different global localization systems for a robot swarm, making it easier for the robots to locate themselves and finish their task. But the drawback of that system is its cost.

In our proposed work, we present simple and cheap robots without any ability to localize. Using scalar fields could be used as the labour division technique, indirectly guiding them to accomplish their mission with independently identical simple rules without any central control or solving spacial interference among robots.

1.2 Research Motivation

The advantage of using a swarm of simple robots is that they can solve the given problem without central control with simple and local behavior. Using a swarm of robots is robust and scalable and can quickly adapt to environmental changes. Another advantage of such a system is that it consists of many individuals; there is no single point of failure, and a problem could be solved with any number of these simple robots. Still, the number of working robots affects the overall time required to solve the given task but does not affect solving the problem itself.

Over the years, swarm robotics researchers presented different systems to solve some basic swarm behaviors like pattern formation, aggregation, self-assembly, collective ex-

ploration, etc [4–7]. The main contribution of solving any proposed behavior is how to divide the work and reduce the spatial interference among the robots in the swarm and handle the communication between the individuals if required.

We are also motivated to design simple behaviors to run on low-cost, simple agents, which allow us to increase the number of robots with a reasonable budget.

1.3 Research Objectives

- Develop a low-cost localization alternative for robot swarms: This objective involves presenting a system where simple and inexpensive robots perform tasks without the ability to localize themselves. Scalar fields are utilized to guide robots in a decentralized manner, allowing them to accomplish missions using simple, identical rules without central control.

- Optimize work division and reduce spatial interference: A significant focus is placed on developing methods to efficiently divide labour among the swarm and minimize spatial interference. This involves creating strategies for effective work distribution and studying various communication protocols among robots.

- Explore applications of scalar fields: The research aims to investigate how scalar fields can be utilized to organize work within the swarm and apply these fields in various tasks, such as shape construction, finding largest coverage area, aggregation, and foraging.

- Enhance swarm robustness and adaptability: The use of a swarm of simple robots aims to create a system that is robust, scalable, and capable of adapting quickly to environmental changes.

1.4 Research Questions

- How can scalar fields be used to divide the work among the swarm? (Chapter 3)
- Can scalar fields serve as a low-cost localization alternative and aid in movement guidance? (Chapter 4)
- How does the resolution of the scalar field affect the swarm's performance? (Chapter 4)
- Does the performance of the swarm improve if we apply machine learning techniques such as reinforcement learning in the presence of high and low-resolution scalar fields? (Chapter 4)
- How can we design a road network to act as the scalar field for the swarm, reducing spatial interference among the robots? (Chapter 5)

1.5 Thesis Structure

The thesis is structured as follows: Chapter 2 provides the related work and background for the thesis. We address the research questions in the subsequent chapters. Chapter 3 presents the use of the scalar field for labour division. Chapter 4 shows the movement guidance using different resolutions of the scalar field and the enhancements when we apply reinforcement learning. Chapter 5 presents how we handle spatial interference among the robots. Conclusions and future work are presented in Chapter 6.

Chapter 2

Related Work

In this chapter, we present how social insects inspired swarm robotics researchers in their work to build artificial systems using a swarm of robots. We show different guidance communication in nature and how researchers mimic it in their work. In addition, we introduce the scalar field, which we will use in the rest of our work.

2.1 Social Insect Inspiration

There are many research papers focusing on how termites' complex, massive structure has different functions as a nest, nursery, and food storage from the soil [8]. Fig. 2.1 shows one of these fascinating mounds with amazing thermoregulation and ventilation systems. Despite the termites have poor vision, they are using indirect communication (stigmergy) for colony communication. Stigmergy is a type of indirect communication that allows agents to communicate with each other by noticing the modifications to their environment [9].

Heyde et al. [11], want to understand how the labyrinthine architecture of the African termite Apicotermes Lamani is built. They studied the recurring designs of the physical structure of ant nests or colonies, called structural motifs. Based on the environmental

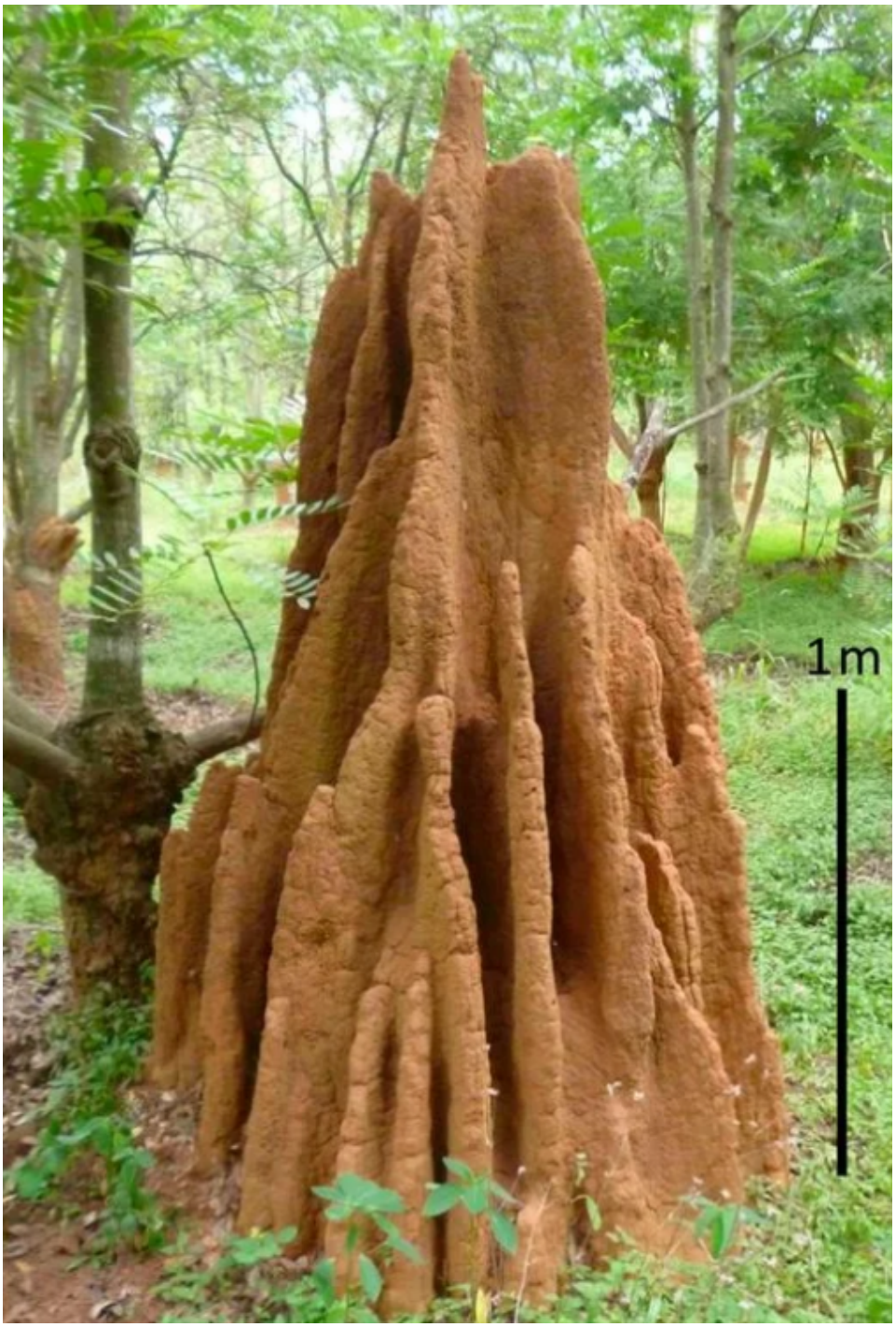


Fig. 2.1: Sculpted morphology of a termite colony of Odontotermes Obesus species [10].

observations of the structural motifs of MeMo13, MeMo14 and MeMo80 (Fig. 2.2), they formulated a minimal model of the spatiotemporal evolution of mud, termites, and pheromones. They found a mathematical model by analyzing the 3d model of the motif, which was assembled using a series of 359 and 177 vertical slices. They measured the floor thickness in every slice and the vertical spacing between floors. Fig. 2.3 shows one of these vertical slices for MeMo80 and MeMo14. After that, the artificial model has generated, as shown in Fig. 2.4, the parallel floors connected by linear and helical ramps that mimic observations of natural nests.

Because of the large number of termites in the colony, termites use active chemical substances called pheromones to regulate social behavior. There are different types of pheromones, such as trail-following, aggregation, and alarm. The trail-following pheromones are used when the workers find new food; they deposit these pheromones on their way back to the nest to guide others to the food source. The aggregation pheromone used to attract individuals to the pheromone source, which the foraging workers discovered. In case there is a presence of danger, the termites use alarm pheromones combined with vibrations to alarm their nestmates. Hence, the soldier termites face the disturbance source, and the workers run away and spread the alarm signal to the others [1].

2.2 Bio-Inspired Swarms of Robots

Social insects inspire researchers because their colonies are robust and flexible; they can adapt to changes in their environment despite their limitations.

In [12], the researchers dealt with the chemical pheromones in insects as a transmitted optical signal from each robot and identified them as "virtual pheromones". These signals propagated through the neighbors and were implemented using directional sensors on each robot sending simple beacons. These virtual pheromones act as messages between robots and have three main parts: the message type, the hop-count field, and

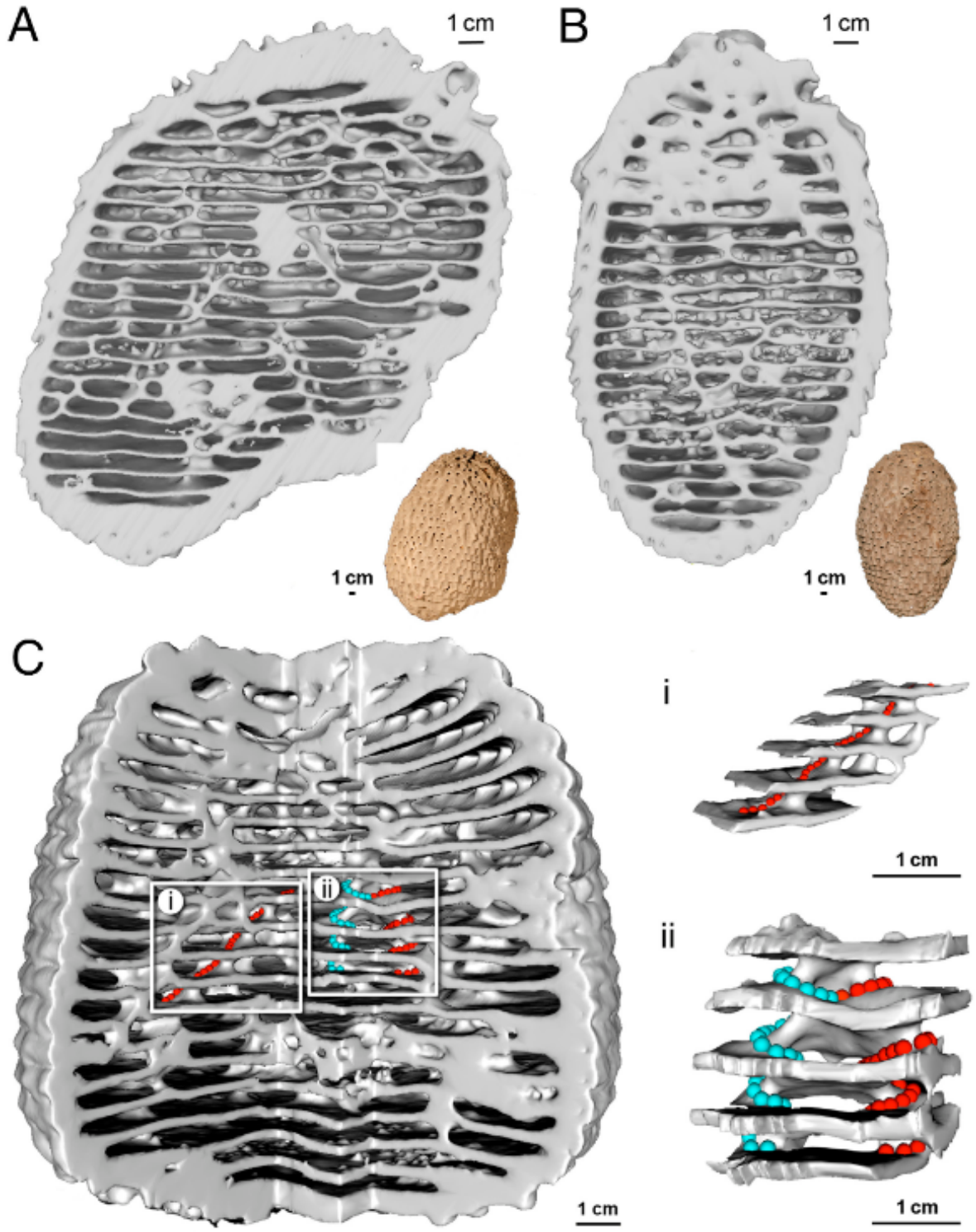


Fig. 2.2: 3D digital reconstruction of x-ray computer tomography of nests of Apicotermes Lamani species. A) Memo14 and b) MeMo13, C) MeMo80; These nests have linear ramps represented by red dots in (C, i) and helicoidal ramps shown as red and cyan dots in (C, ii) [11].

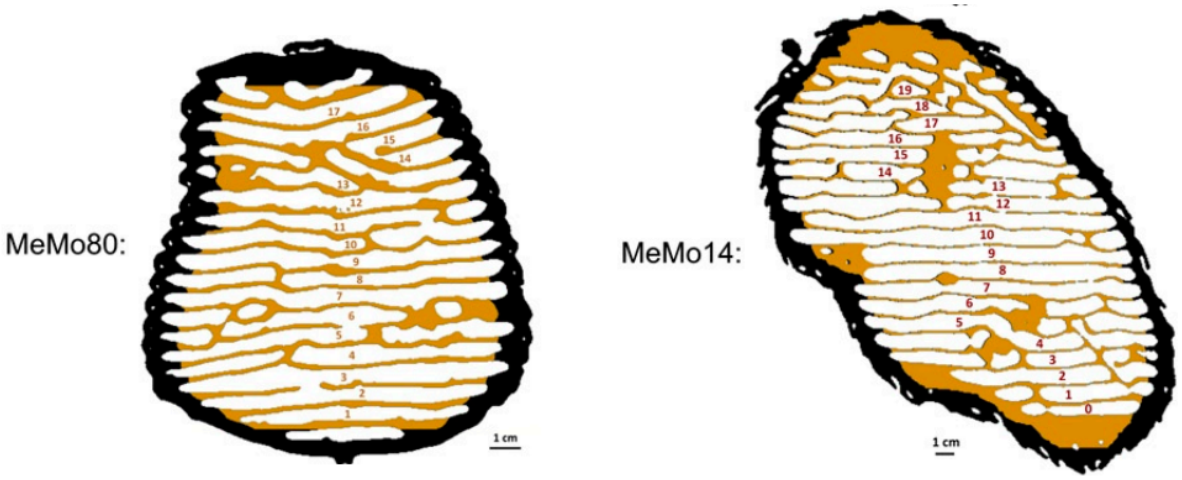


Fig. 2.3: Vertical slices of Memo80 and MeMo14 nests' structures with labelled floors [11].

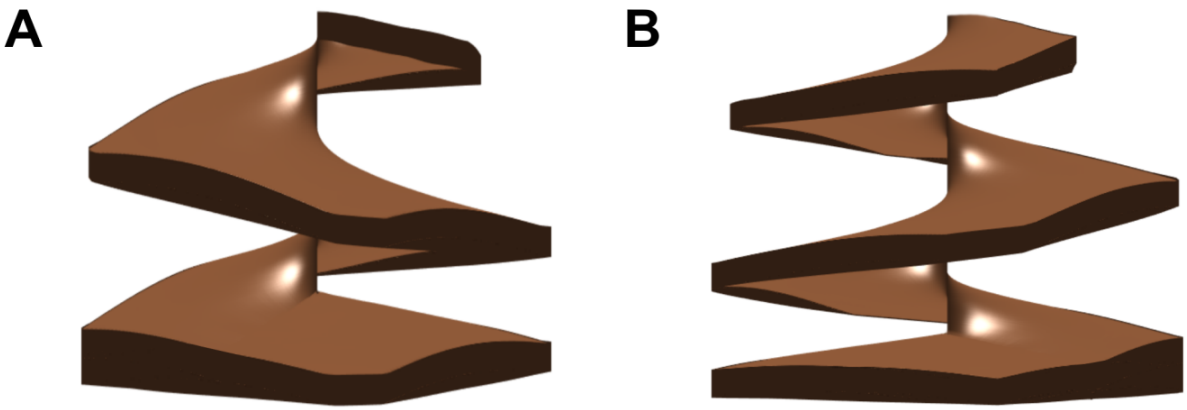


Fig. 2.4: Helicoidal ramps that have been smoothed and extracted from model simulation. To emphasize the regularity of the helicoid, two exemplary helicoidal ramps are presented (A) left-handed and (B) right-handed were taken from numerically generated nests. [11]

the data field. When the robot receives this signal, it modifies and retransmits it to its neighbor. In their proposed system, the swarm of robots is treated as a grid of distributed computation nodes. The authors use the grid to get the global information about the environments, like the shortest paths between different points. The difference between chemical and virtual pheromones is that the proposed virtual pheromones connect robots to themselves instead of tying them to the environment.

Arvin et al. [13] define another artificial pheromone system and call it COSΦ. They try to simulate the natural pheromones from the insect's colony and change the pheromones' parameters like diffusion and evaporation. To do their experiments, they established a low-cost setup that included a visual localization system that captured the robot's position. They released the artificial pheromones on the LCD screen based on the robots' positions. The robots were equipped with light sensors to detect these pheromones, as shown in Fig. 2.5 . The authors are interested in studying how changes in the parameters of the pheromones affects the robot's behavior, so they targeted four parameters. The first parameter is an injection which means how fast a given robot releases the artificial pheromone. The second parameter is the evaporation half-life which shows the time it needs for the pheromone to fade. Diffusion is defined as the spreading rate of the artificial pheromone. Finally, the fourth parameter is the influence, which measures how much the displayed image on the LCD is influenced by the pheromones. They implemented a swarm scenario; one leader robot moves forward with a constant speed and leaves a pheromone trail for the follower robots to follow. When the follower robots find the trail, they randomly choose to take a left or a right direction to follow this trail. The conclusion from this work is increasing the evaporation half-life leads to increases the stability of the pheromone trail, and the follower robots can easily find the trail and follow it.

Like the termite's mound, built by using millimeter-scale insects, Werfel et al. [14] are

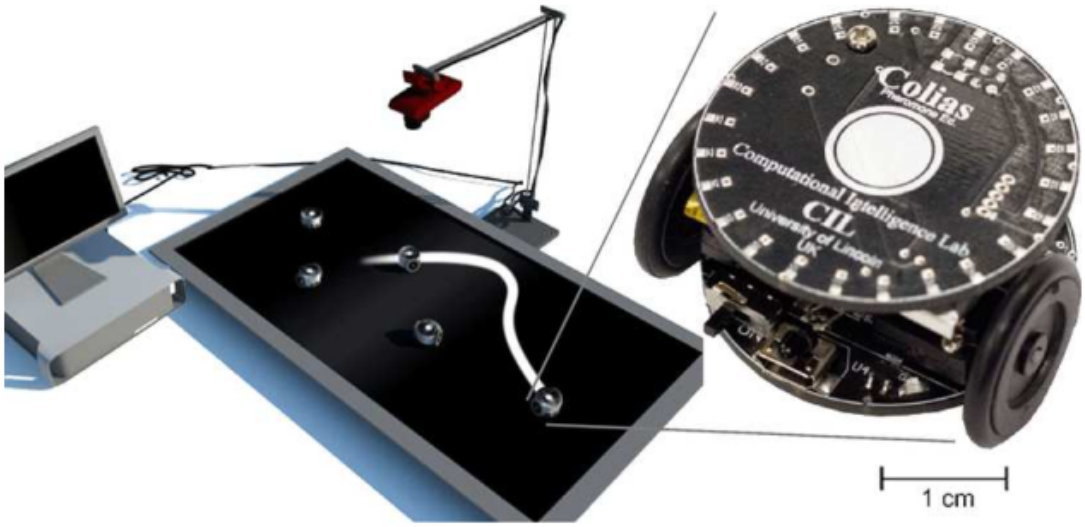


Fig. 2.5: COSΦ artificial pheromone system. [13]

interested in collectively constructing a 3d large-scale structures using climbing robots by placing passive building blocks in a specific design to form the final construction. They introduced a system called TERMES 3D collective construction system in which robots are decentralized and dependent only on local information and implicit coordination. Their robots can only climb or descend one step of a block, and they target handling large structures, so they placed the blocks as staircases to climb and reach the higher levels, as shown in Fig. 2.6. The user provides the target structure as an occupancy grid and marks which sites they want to be occupied by the blocks. This occupancy grid should not have overlapped blocks or curved surfaces, and other blocks or the ground must support all building blocks. The authors made a compiler that takes this occupancy grid as input and generates a data structure that encodes information about the structure's height and allowable travel directions at each point in the grid to avoid any deadlock. They called this data structure a "struct path" as shown in Fig. 2.7, which is distributed to robots and used in construction. Using this struct path for robots is like a guide similar to pheromones which the termites used in royal chamber construction. In this system, they can construct any 3D user-specified and find simple rules to discover

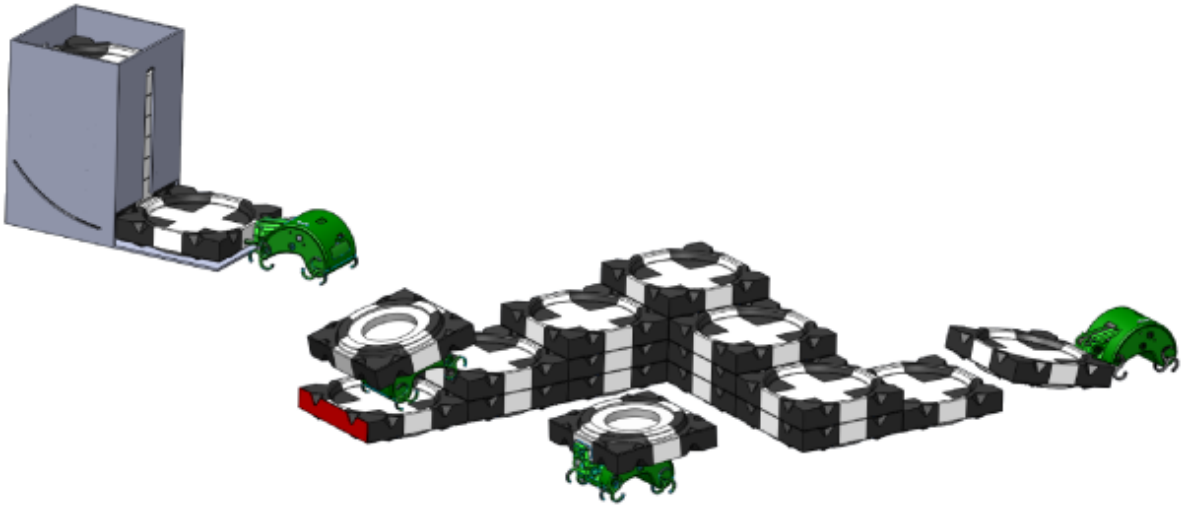


Fig. 2.6: Overview of the TERMES System, robots gather building bricks from a cache (at left) and start with a marker block (with a red face) to construct a desired structure [14].

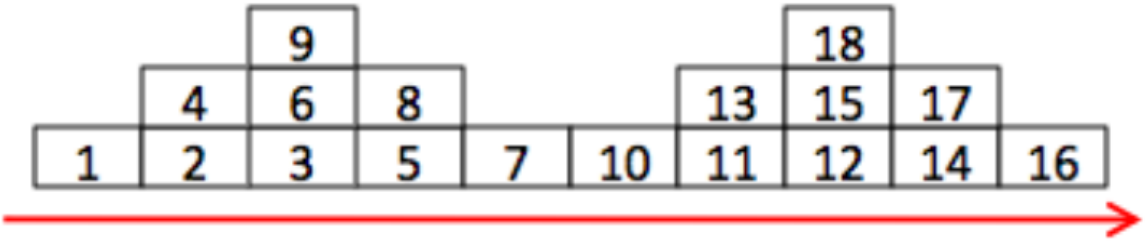
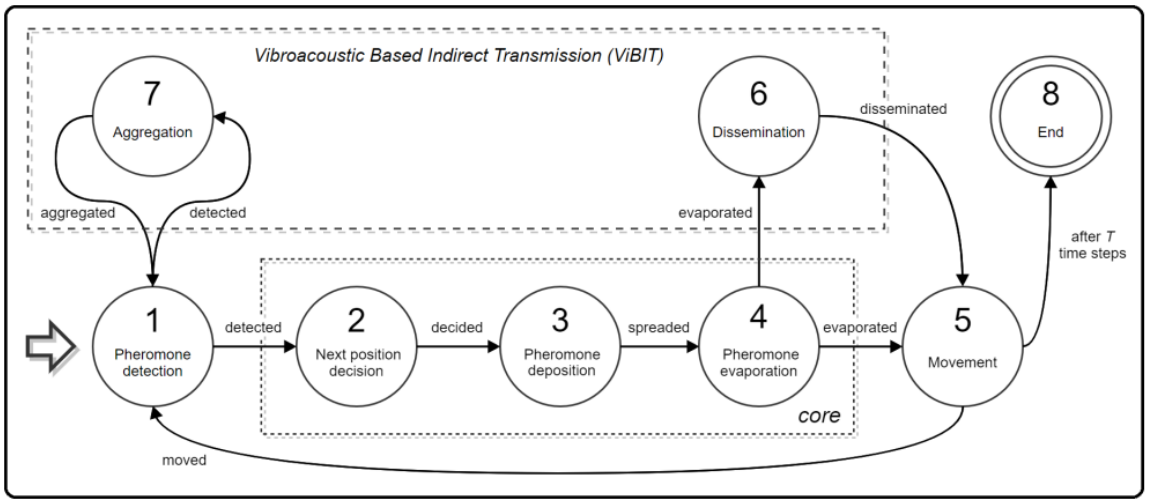


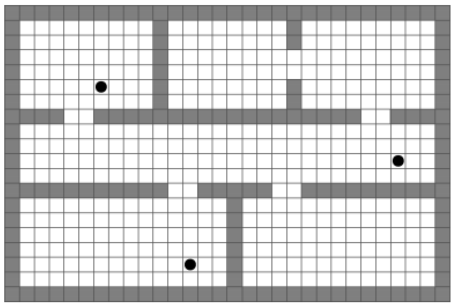
Fig. 2.7: Blocks will be added to the structure in the way specified; robots enter from the left and depart to the right, as seen in this side view of a linear structure [14].

the intermediate deadlock structures or detect the structures the robots can not traverse.

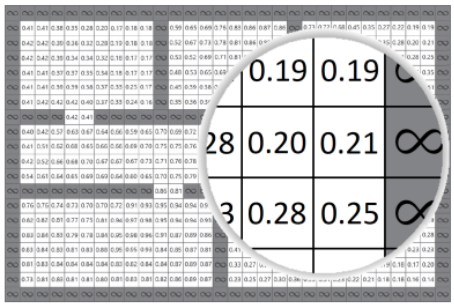
In PheroCom [15], the authors designed a pheromone-inspired model and used it as a decentralized navigation decision-making process for robots to handle the coordination among them. PheroCom controls pheromones' dynamics by distributing and storing local virtual maps on each robot. The robots propagate their local pheromone map with the others. They are inspired by the Vibroacoustic communication in ants in which ants communicate through the sounds reproduced from the vibrations. Ants produce that sound through the friction of body parts and the drumming of the abdomen against the environment. The researchers introduced their communication as a hybrid



(a) Robot control mechanism represented by a FSM.



(b) Physical grid.



(c) Pheromone grid.

Fig. 2.8: PheroCom system; a) A FSM for the decision-making process has eight states to explain each robot's behaviour. b) and c) Examples of virtual grids/maps used by the FSM. [15]

communication of Gossip protocal and local storage of phermone and physical map on each robot. They called their communication technique Vibroacoustic Based Indirect Transmission (ViBIT). Fig. 2.8, shows how the robots get the global pheromone grid by asynchronously involving local grids, as each robot may be at a different time step of its internal finite state machine.

There is another robotics task called the shape formation problem in which the relatively simple group of robots should construct a desired shape or pattern, such as a straight line, circle, or another geometric shape; the robots try to move and reposition themselves to achieve that shape. Researchers try to develop robust and scalable al-

gorithms to control the robots to form that specific shape while working together to accomplish complex tasks. In [16], Wang et al. tackled this shape formation problem by proposing a fully distributed and leaderless method requiring local communication among the swarm, and a grid showed the desired locations to form the shape. They divided the problem into two main subproblems: First, they assigned distinct locations for the robots in the shape. The second subproblem was finding collision-free and deadlock-free paths for the robots to reach their places. To find the robot's goal location, they provide a robot with a set of desired target points as a set of nodes on a grid, and all the robots have the same global reference frame. Their algorithm distributed the target points among the individuals and tried to find a collision-free path for the robots to form the desired shape using peer-to-peer communication. Each robot can communicate with other robots if they exist in its communication range, so they can decide whether that path is free. If they find there is a deadlock, the robots can swap their goals, so they can also reduce the total distance travelled by them. Fig. 2.9, shows how their goal selection algorithm and robot communication work to solve the same goal assigned to 2 robots. Each robot is coloured with the same colour as its destination goal, and it holds its id, hop count and next candidate goal. So in frame 1, robot #1 and robot #3 have the same blue goal, but in frame 3, robot #3 reached the position first, so robot #1 changed its destination to the green goal after they communicated and found that its id is smaller than robot #3.

2.3 Guidance Communication

As we explained in section 2.2, researchers use virtual pheromones, data structures like struct path and pheromone grid, or peer-to-peer communication to guide the robots in their tasks. All these methods help robots to make use of information relevant to the task that is not available locally. It is worth studying the guidance communication used

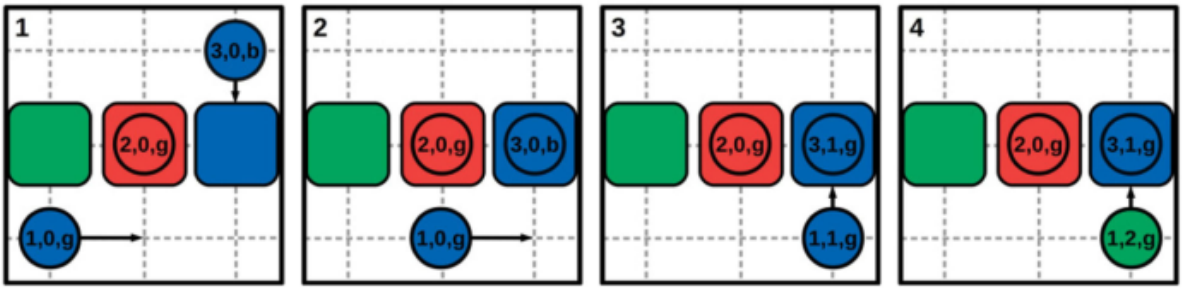


Fig. 2.9: Example of how robots re-assign their goal based on peer-to-peer communication. [16]

by social insects and finding the equivalent in the virtual systems. In Fig. 2.10, natural communication shows different types of communication inside various colonies, either animals or insects. However, virtual communication demonstrates how the existing models for the swarm of robots can mimic natural communications. We explain both in the following subsections.

2.3.1 Natural Communication

Intracolony communication is not limited to chemical pheromones; several communication techniques exist in the same colony. For example, as shown in Fig. 2.11, termites can send messages through antenna touch and produce vibrations in exocrine glands located at the end of ants' abdomen [17]. Wasps and bees can use vibroacoustic signals (vibrations and sounds) among their colony, including body movements like wing movements, high-frequency muscle contractions, or tapping body parts. Honey bees use the waggle dance to inform others about profitable food sources [18]. This dance is considered a multi-component signal because it can encode different information, for example, 1) attract other bees to receive the information, 2) show the distance and the direction of the good food source. Fig 2.12 shows how the waggle run performed.

On the social animal side,(e.g. fish,bats,and mammals), they use chemical signals like pheromones by producing them. Also, males can use it to signal their territory [19] [20].

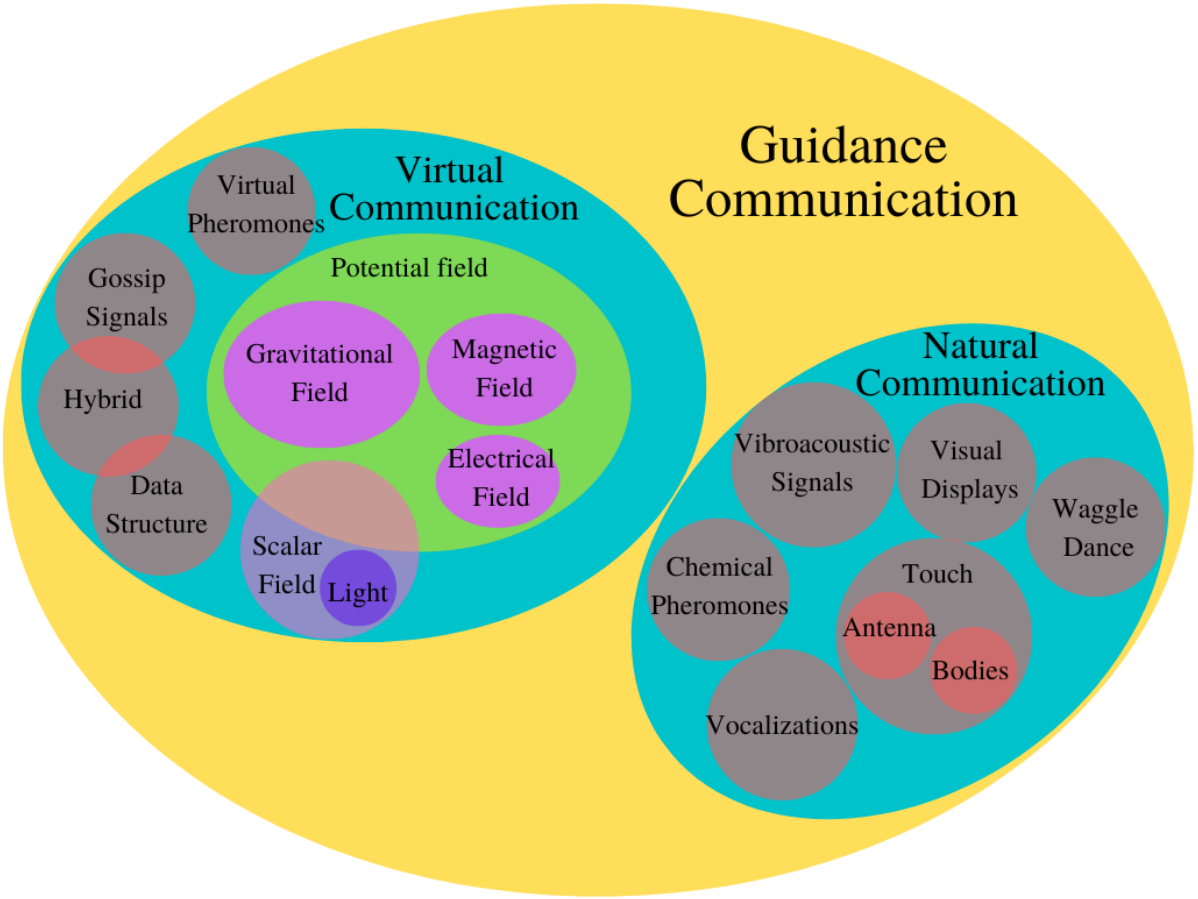


Fig. 2.10: Guidance Communication for Multi-agent System.

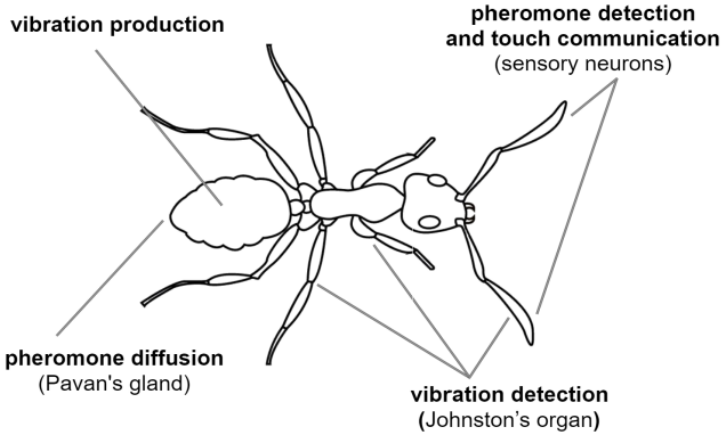


Fig. 2.11: Communication mechanism of ants; Antenna touch acts as vibration and pheromone detection. [17]

Fishes and birds use visual displays and perform different postures and movements to communicate. Besides the typical vocalizations, communication exists in mammals, frogs [19], and birds [21].

2.3.2 Virtual Communication

As mentioned, researchers mimic natural communications and present virtual communications to guide robots in pathfinding, avoiding obstacles, or finding the goal. These virtual pheromones are the artificial version of chemical pheromones and significantly guide swarm movements. Also, gossip signals as intercommunication among robots and using a data structure to store information locally on a swarm are used separately or in hybrid mode.

2.3.3 Reactive Robotic Paradigm

The robot's behaviour is defined as a direct mapping between the sensory data to appropriate motor action to accomplish the required task as shown in Fig. 2.13. In order to build a real-time, responsive and simple robotic system, this robotic system should

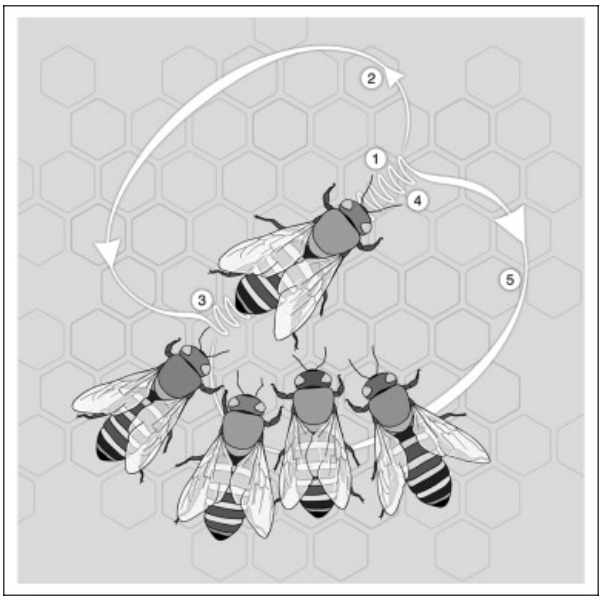


Fig. 2.12: A dancer honey bee performs a waggle run, then turn to one side and circles back to the first point of the waggle run. After that, it starts another waggle run on the other side. The more waggle runs done by the bee, the more profitable the food source it finds. While dancing, the bee releases chemicals and produces vibrations, so followers use their antennas to touch the dancer. [18]

follow a reactive robotic paradigm in which the robot senses the environment and then acts directly without the need for planning [22]. The reactive system has some important characteristics [22] such as:

- Every robot has its goals, and the environment is changed based on its actions. The robot adaptively senses the change and adjusts its goals to meet the task requirements.
- Behaviours are considered building blocks for the robot's actions. The system should define how these behaviours got triggered.
- Only local sensing is permitted.

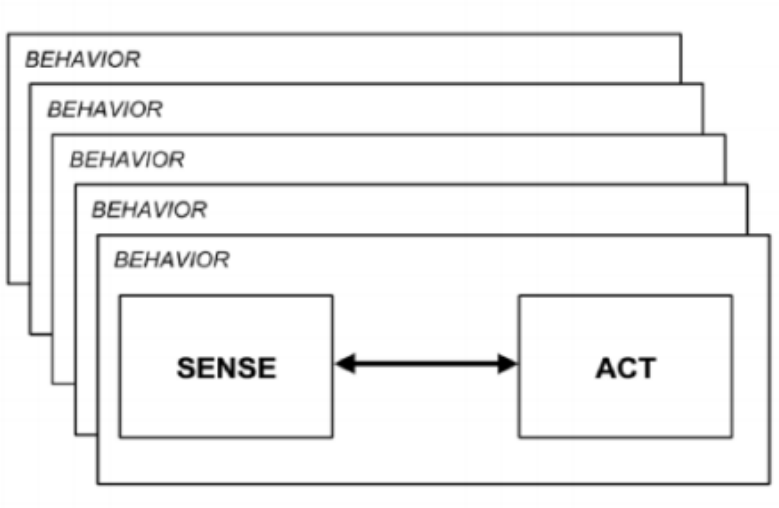


Fig. 2.13: Sense - Act design of behaviours in reactive paradigm [22].

2.3.3.1 Potential Field

In physics, a potential field is used to represent the potential energy connected to a specific force. The electric potential, for instance, in electrostatics, describes the potential energy per unit charge at a particular location in space. Similarly, gravitational potential quantifies the potential energy per unit mass at a specific location. Defining forces in different locations helps distinguish between these locations and helps agents find their way through the environment. A potential field assign a value to all points in the space. For example, if we are taking inspiration from an electromagnetic field, we can specify that the goal has a positive charge, the agents have a negative one, and any obstacles have a negative charge. So, the agents can easily be attracted to the goal using that attractive potential field and avoid the obstacles using the repulsive potential fields, as shown in Fig. 2.14 . Therefore, the potential fields can regulate the robot movements in the environment without local storage or extra communication among the swarm [23].

A potential field method is a well-known method that fits in the reactive paradigm due to its reactive nature and simplicity in generating robot behaviours. Potential fields allow robots to react immediately to local sensory information. This aligns with

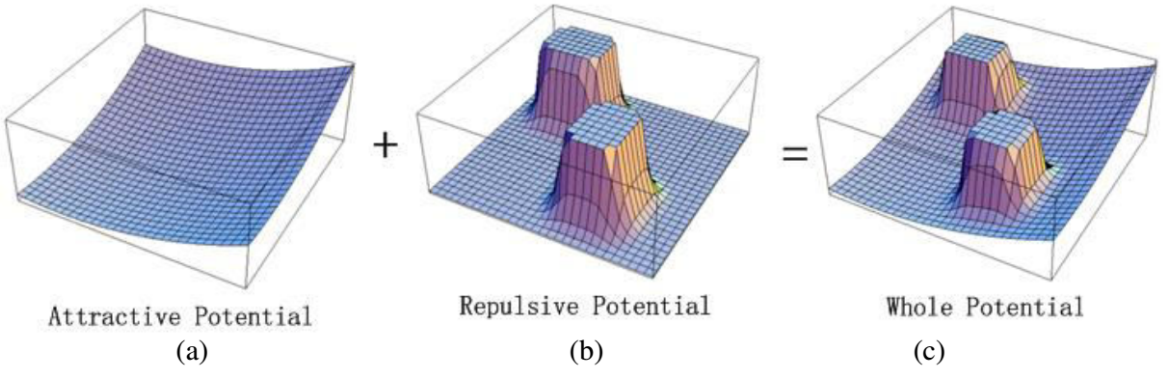


Fig. 2.14: (a) Demonstrates the attractive potential field to the goal (b) The repulsive potential field shows the highest value in the presence of the obstacles (c) The final potential field is a combination of the attractive and repulsive fields that the robot uses to navigate and avoid obstacles. [23].

the reactive paradigm's focus on using local sensory data to guide immediate actions rather than relying on a global understanding of the environment. The behaviour of the potential field is described as a vector representation, and it can combine multiple behaviours using vector summation. Fig. 2.15 shows examples of vector fields of the primitive potential fields: (a) uniform field, the robot has the same force everywhere. (b) perpendicular field: The robot is directed to move towards or away from the wall. (c) attractive field: robots get attracted and move toward light or food. (d) In a repulsive field, the robots avoid and move away from the obstacles. (e) Tangential field: The robot rotates clockwise around an object. We can combine multiple potential fields, as shown in Fig. 2.16, so the robots are attracted to the goal and avoid obstacles [22].

2.3.3.2 Scalar Field

A scalar field is defined as a function that associates a scalar value with each point in a given space. In other words, it is a function that assigns a single numerical value (or scalar) to every point in a region of space [24]. Scalar fields could represent different quantities like temperature or light intensity. As we mentioned earlier, the primary

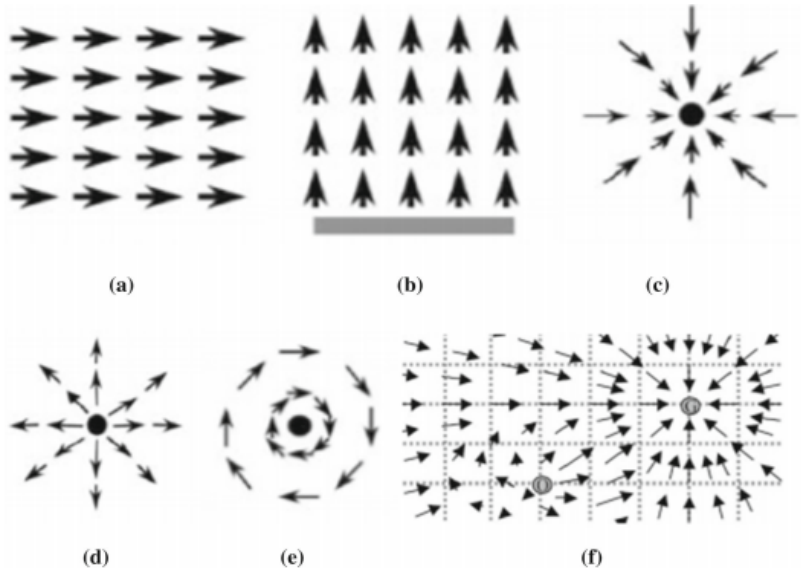


Fig. 2.15: Vector fields that result from computing the gradient of primitive potential fields: (a) uniform, (b) perpendicular, (c) attraction, (d) repulsion, and (e) tangential. (f) a potential field combined two primitive potential fields: attraction (goal) and repulsion (obstacle) [22].

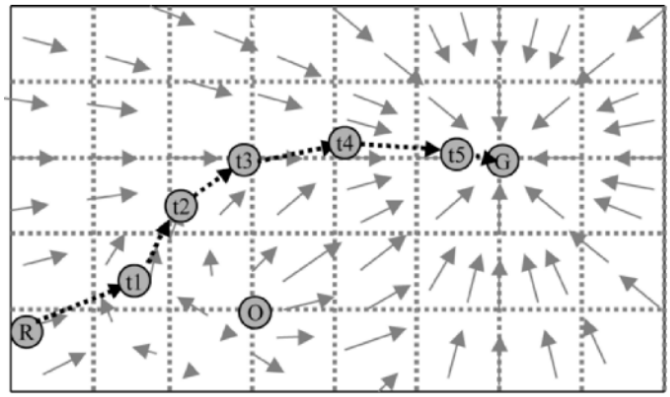
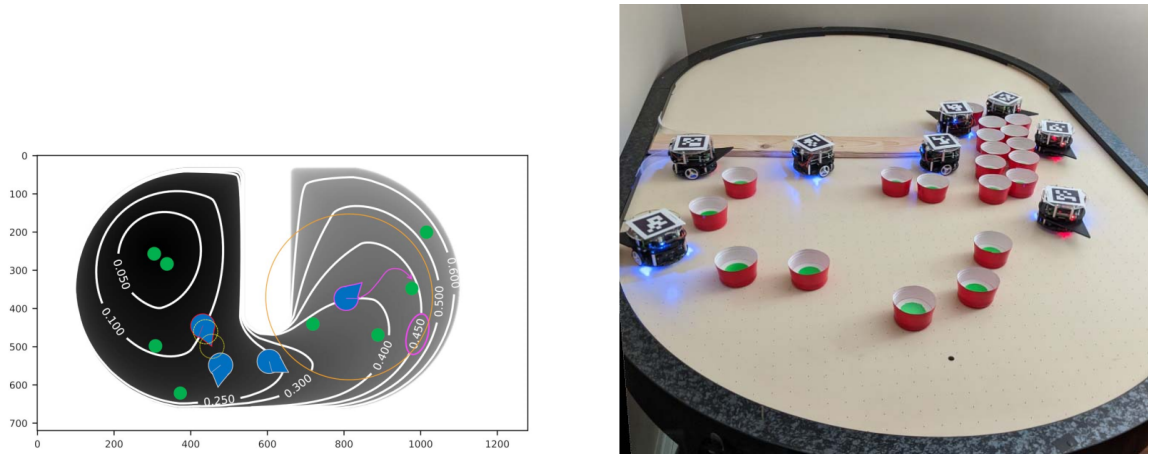


Fig. 2.16: The robot takes this path to avoid the obstacle and be attracted to the goal [22].



(a) (b)

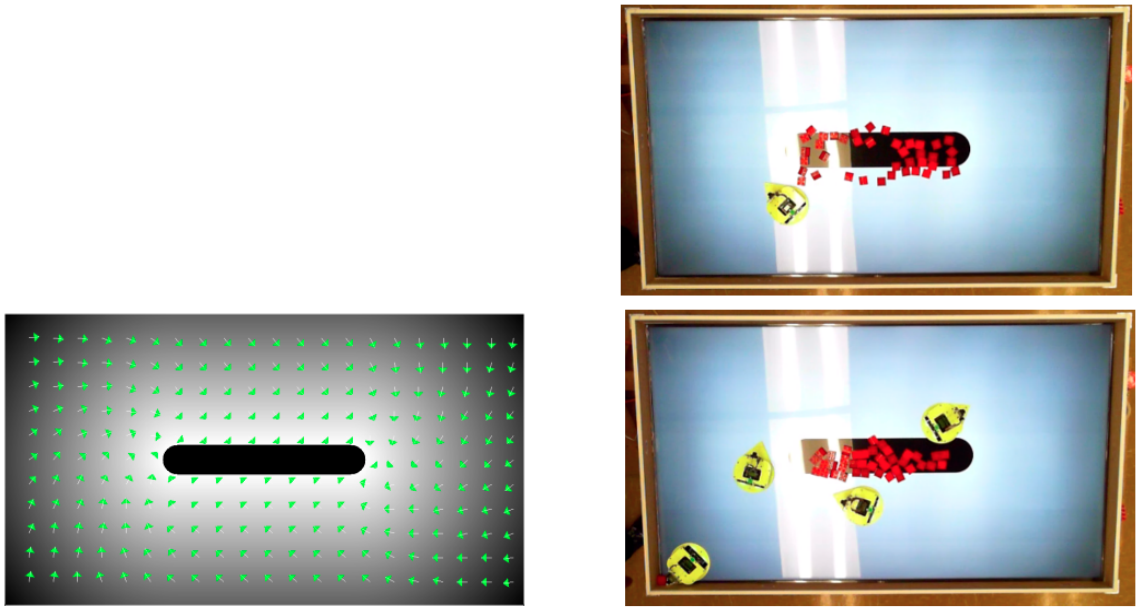
Fig. 2.17: The right figure shows the scalar field for the lasso method for the foraging task. The left figure shows the robots looking up for the scalar field based on their position [25].

communication in the termite colony is depositing chemical substances (pheromones), which are volatile substances that will evaporate over time. So if the termite detects a greater amount of pheromones, it indicates this site has been visited by another termite recently. However, if the pheromones concentration is lower, this place is less important than the other places [15].

2.3.4 Providing Potential field to Robots:

Researchers provide potential fields to robots in various ways, depending on the specific application and the robots' capabilities. Here are some ways used to provide a potential field to robots.

Vardy [25] calculated the scalar field based on the goal position and the boundary's geometry for the foraging task. In this work, the author assumed robots can localize, and they use their position to get the associated scalar field values. Fig. 2.17(a) shows the generated scalar field and Fig. 2.17(b) shows the robots in action.



(a) (b)

Fig. 2.18: The right figure shows the vector field for constructing a planar Shape. The left figure shows the robots sensing the scalar field using their local sensors [26].

However, in this work [26], the robots rely on local sensing, So they provide the scalar field as a static image to robots, which are equipped with phototransistors to measure the image and use it as guidance in their movement. Fig. 2.18(a) and 2.18(b) shows the vector field and the real robots operated on the projected scalar field image on TV.

The potential field could be calculated based on the robot's surroundings. Martínez et al. [27] implemented a decision-making algorithm for autonomous vehicles operated in unknown environments. Their proposed path planning was based on the potential field, which was calculated given lanes, the other vehicles' position, and the existence of surrounding obstacles.

Chapter 3

Using Scalar Fields as a Division of Labour Technique for Robot Swarms

In this chapter, we demonstrate the use of the scalar field to regulate work among a swarm of robots. We present a planar construction task as an example of collective behavior. Additionally, we explore different communication techniques among the robots. This chapter presents parts of the work published in my papers:

- Dalia S. Ibrahim, and Andrew Vardy. "Adaptive task allocation for planar construction using response threshold model." Theory and Practice of Natural Computing: 8th International Conference, TPNC 2019, Kingston, ON, Canada, December 9–11, 2019, Proceedings 8. Springer International Publishing, 2019.

3.1 Planar Construction using Response Threshold Model

The objective for the planar construction task is to form a desired shape from ambient objects existing in the robot's plane of operation in a distributed behavior approach using a swarm of robots. We assume robots can do simple arithmetic operations and sense the projected scalar field, which guides a swarm of robots in their construction task. We are using fixed thresholds in this scalar field to specify the contour of the desired shape. The specific shape targeted is an annulus (ring shape); we need to allocate resources and distribute tasks among the robots. We present three communication techniques besides the scalar field to show how coordination is done in the swarm. Robots switch between tasks autonomously based on a calculated response function.

3.1.1 Introduction

Using swarms in collective construction tasks have exciting applications. They may be used to build a wall around a chemical or radiation leak [28]. Also, they can be used in many automated applications, like sorting recycling or floor cleaning.

The author in [29] uses the pheromone idea in swarm robots to construct enclosed shapes like circles, oblongs, and crosses. They used the pheromone as a static template of the desired shape and implemented it by projecting a scalar field with a combination of fixed thresholds to guide the robots to the desired shape's contour. The formation of two-dimensional shapes is called **planar construction**. The author introduces an algorithm called orbital construction **(OC)** in which the swarm of simple robots orbit in a clockwise direction and push pucks inwards and outwards to form a specific shape. Robots consist of two different types: **innies** and **outies**. Firstly, innies work inside the desired shape and pushing pucks outwards. Secondly, outies work outside the shape and

pushing the pucks inwards.

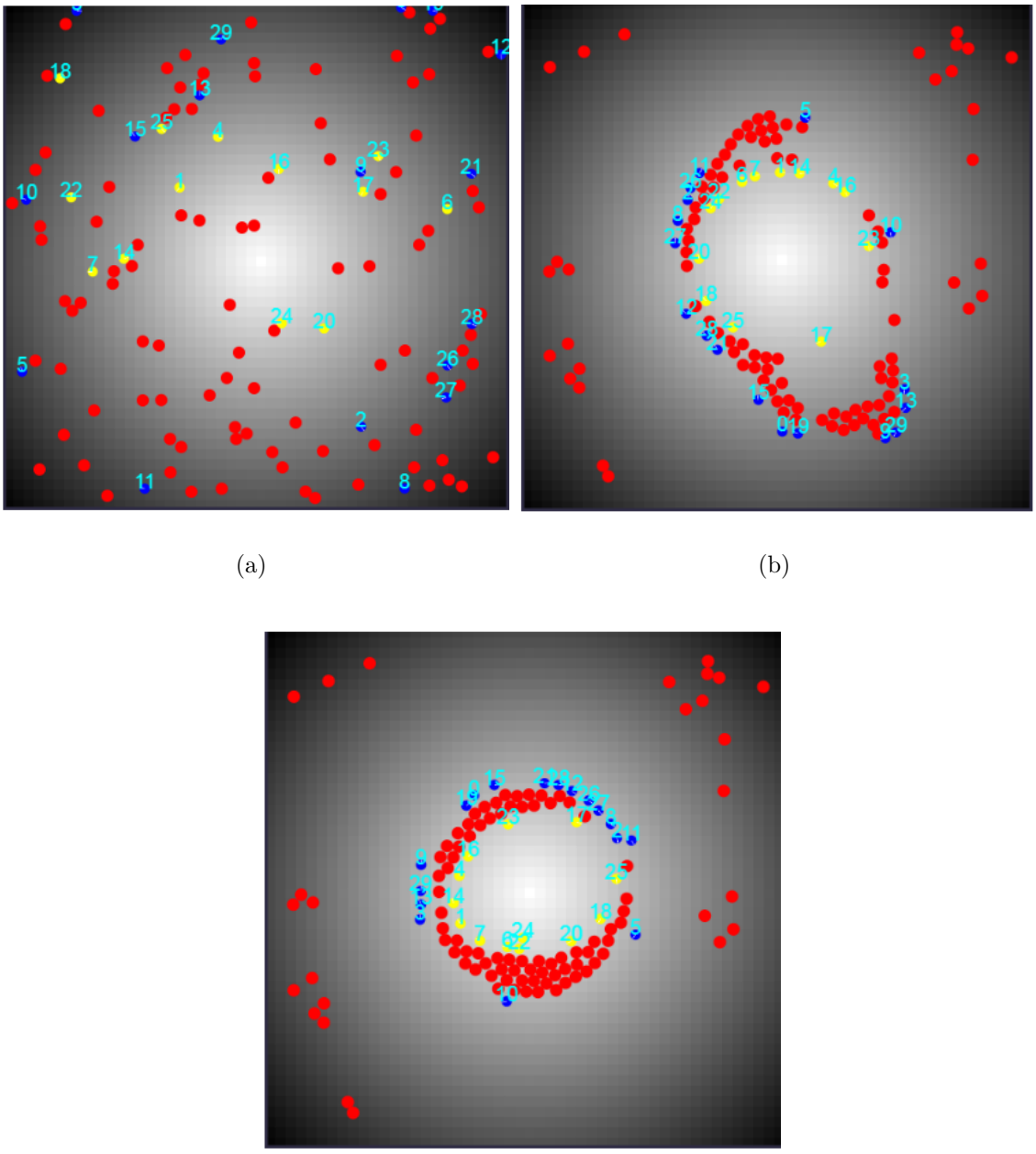
These robots have three basic behavior characteristics [29]: Firstly, periphery seeking, in which robots move to the outside of the region occupied by pucks. Secondly, they push these pucks by nudging them inwards. Thirdly, the robots orbit around the environment in a clockwise direction, so their paths are called orbitals. These **orbitals** are defined by the light intensity of the projected scalar field. In the proposed work, we divide the environment into 10 orbitals with different light intensities $\tau \in [0.1, 1]$. Explorers work in outer orbitals searching for detached pucks, operating where $\tau \leq 0.6$. Outties maintain the shape contour and exist on $0.6 < \tau \leq 0.7$. Innies work inside the annulus where $\tau \geq 0.75$.

Fig. 3.1 shows our implementation of the **OC** algorithm to form an annulus with a projected scalar field. This algorithm has promising results. Most of the pucks are pushed and placed in the annulus, but there were still some pucks detached from the constructed shape, especially when these pucks were near to environment borders.

In this work, we introduce the role of **explorers**. The task of an explorer is to scan the environment and push pucks towards the annulus. Outties still take care of the annulus' boundary, and innies work inside the shape to keep it empty by pushing the pucks outwards to the annulus' contour. The main challenges here are knowing which orbital has pucks and the regulation of the division of labor between exploring robots.

3.1.2 Using Response Threshold in Orbital Construction

The general idea of the response threshold to stimuli in social insect society is these insects should be highly responsive to their surroundings and communicate efficiently. That requires a high level of sensitivity to the stimuli. For example, ants' response threshold to stimuli is low, allowing them to detect the pheromones quickly. Another example is a household analogy of a person who resists doing a task until the stimuli



(a) (b)

(c)

Fig. 3.1: Sequence of snapshots show our implementation of the OC algorithm at 0, 1000, and 5000 time steps. The simulation is done with 30 robots consisting of 13 innies (yellow) and 17 outies (blue) with 100 red pucks.

associated with the task exceed a threshold; for example, he resists throwing out the garbage until its smell exceeds his ability to handle it. His ability to tolerate the smell represents his threshold, and smell represents the stimuli.

In this work, we use the exponential response threshold function to help the explorers know if their orbital is empty or occupied. Theraulaz et al. [30] proposed a model based on the concept of response thresholds to divide labor in insect societies. Their definition of response threshold is a model of how the insects react to task-associated stimuli. Every worker has its response threshold for every task. This worker will be engaged to perform the task if the task stimuli, s, exceeds its response threshold, θ. Therefore, they are calculating the response function $T_\theta(s)$, which determines if they can do this task or not [31]. After they have engaged in this task, they will have a low task stimulus, so they will switch to another task.

In our work, the threshold is θ, the time a robot takes to circumnavigate the orbital. This threshold represents the maximum time required for the robots to circumnavigate any empty orbital, so it will be used later to calculate the response threshold function. The following Equation 3.1 shows how threshold θ is calculated where C is the orbital circumference, ω is the robot's angular velocity, and i^{th} represent orbital number.

$$\theta_i = \frac{C_i}{\omega} \tag{3.1}$$

In this equation, ΔT is the difference in time between two consecutive actual pushes in the same orbital, as shown in Fig. 3.2.

Algorithm 1 shows how the simple moving average for time ($Avg\Delta T$) is calculated with a subset size of 2. In general, if $Avg\Delta T$ is very small compared with θ, this orbital is occupied with pucks, and the robot should stay on this orbital and keep pushing the pucks to the next orbital. In contrast, if $Avg\Delta T$ is equals θ, that means this orbital is empty. To summarize the relationship between the task stimulus and response threshold,

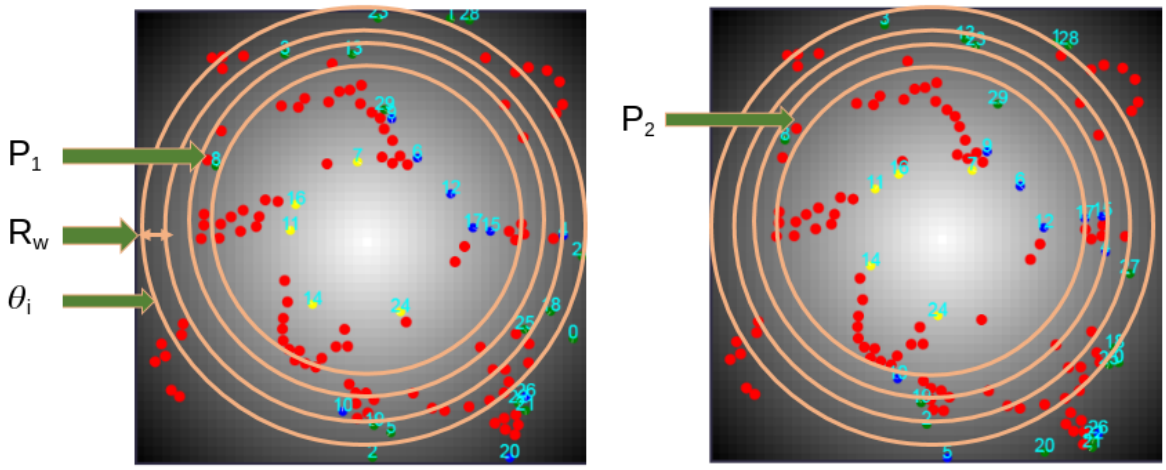


Fig. 3.2: The left image shows the initial environment configurations; the orbital width R_w, θ_i is the elapsed time to circumnavigate the orbital i, and P_1 is the point where robot number 8 pushes the red puck. On the right, this image shows the next point P_2 where robot number 8 pushes another puck in the same orbital. So, the ΔT will be the difference between the moments where these points were visited. Also, this figure shows the three types of robots innies (yellow), outies (blue) and explorers (green).

we calculate F_θ, which represents the local Response function for each robot at the given time. If F_θ equals zero, that means this orbital is empty. If F_θ equals 1, this orbital is highly occupied with pucks. We use equation 3.2 and multiply the exponent by -40 to ensure the $F_\theta(Avg\Delta T) \in [0, 1]$.

$$F_\theta(Avg\Delta T) = e^{\frac{-40 \cdot Avg\Delta T}{\theta}} \tag{3.2}$$

3.1.3 Proposed Methods

In this section, we present the common configuration for the three proposed communication methods, and after that, the differences among them are discussed.

The robots N are randomly assigned to one of three categories: innies N_i, outies N_o, and explorers N_e.

$$N = N_i + N_o + N_e \tag{3.3}$$

Algorithm 1 Measure simple moving average time for two consecutive pushes

```
1: function MeasureConsecutivePushes(LastΔT, θ, PrevTime )
2:     CurrTime = System.getTime()
3:     if CurrTime − PrevTime >= θ then          ▷ Empty Orbital
4:         AvgΔT = θ
5:     else if Robot.Push then                   ▷ Robot pushes a puck now
6:         AvgΔT = (LastΔT+(CurrTime−PrevTime)) / 2
7:     PrevTime = CurrTime
8:     return AvgΔT, PrevTime
```

The robots have sets of sensors. An obstacle sensor (obs_l), which indicate the presence of a wall or another robot in the robot's left region. Three floor sensors ($floor_l$, $floor_c$ and $floor_r$) can measure the light intensity of the projected scalar field. Robots are also equipped with two puck sensors located on the robot's left and right regions ($Pucks_l$ and $Pucks_r$). In addition, there is a lamp that can be switched on or off to indicate if this robot needs help or not and a light sensor ($Light_l$) that detects the existence of lights for the other closest robots on its left. The innies' and outies' movement follows the **OC** algorithm in [29] which takes these sensor readings as inputs and produces angular speed. The explorers' movement follows Algorithm 2.

Explorers start their work by choosing a random orbital, because if they start from the environment boundary, after some time, it will be difficult for the robot to push multiple pucks at the same time. During their work, the explorers communicate with each other as described in the following subsections. Finally, when they have entirely scanned all of the environment, the explorers will start to scan the environment again from the beginning in order to not miss any detached pucks and keep searching for any new pucks added to the environment.

Innies will be trapped after some time; that is why their group is very small compared

Algorithm 2 Orbital Construction for Explorers

```
1: function ExplorerOrbitalConstruction(obs_l, floor_l, floor_c, floor_r, pucks_l, pucks_r)
2:     if obs_l then
3:         return ω_max                          ▷ Robots avoid obstacle by veering to the right
4:     if floor_r >= floor_c ∧ floor_c >= floor_l then      ▷ Robots alignment for clockwise movements
5:         if Explorer ∧ pucks_l then
6:             return −ω_max                     ▷ Explorers steer outwards to collect remaining pucks
7:         if floor_c < τ then                   ▷ Steering inwards
8:             return 0.3ω_max
9:         else                                  ▷ Steering outwards
10:            return −0.3ω_max
11:    else if floor_c >= floor_r ∧ floor_c >= floor_l then      ▷ Robots is aligned uphill
12:        return −ω_max                         ▷ Turn left
13:    else                                      ▷ Robots is aligned downhill
14:        return ω_max                          ▷ Turn Right
```

with the other two groups, and their rule is to keep the area inside the desired shape empty. The outies operate outside the shape and keep maintaining the shape's boundary. In case any explorers push pucks towards an annulus, the outies will catch them and push the pucks towards the boundary.

In the following subsections, we present three alternatives for the communication of explorers.

3.1.3.1 Global Communication

As stated in Section 3.1.3, the explorer chooses its orbital randomly, calculates its response threshold, and works in this orbital until it is cleaned. While it works, it broadcasts its calculated value and others, who are finished working, listen, calculate global response threshold for all orbitals, and choose the orbital with the highest puck capacity and start their work in that orbital.

Every explorer robot has an array initialized with ones, and its size is equal to six, which is the number of orbitals that the explorer could work on. When any robot receives the response threshold values for a specific orbital X, it updates the X orbital's response threshold value in its array based on equation 3.4. The explorer trusts in its calculation more than the received values. Hence, we weight these two thresholds at α and $(1-\alpha)$ respectively, where $\alpha = 0.7$, as shown in Equation 3.4.

$$ResponseArr[X] = 0.7ResponseArr[X] + 0.3RecievedResponseThreshold \quad (3.4)$$

Random distribution of the robots allows the measuring of the response threshold over the whole environment at the same time. By exchanging this information, the robots determine which orbital is the most occupied.

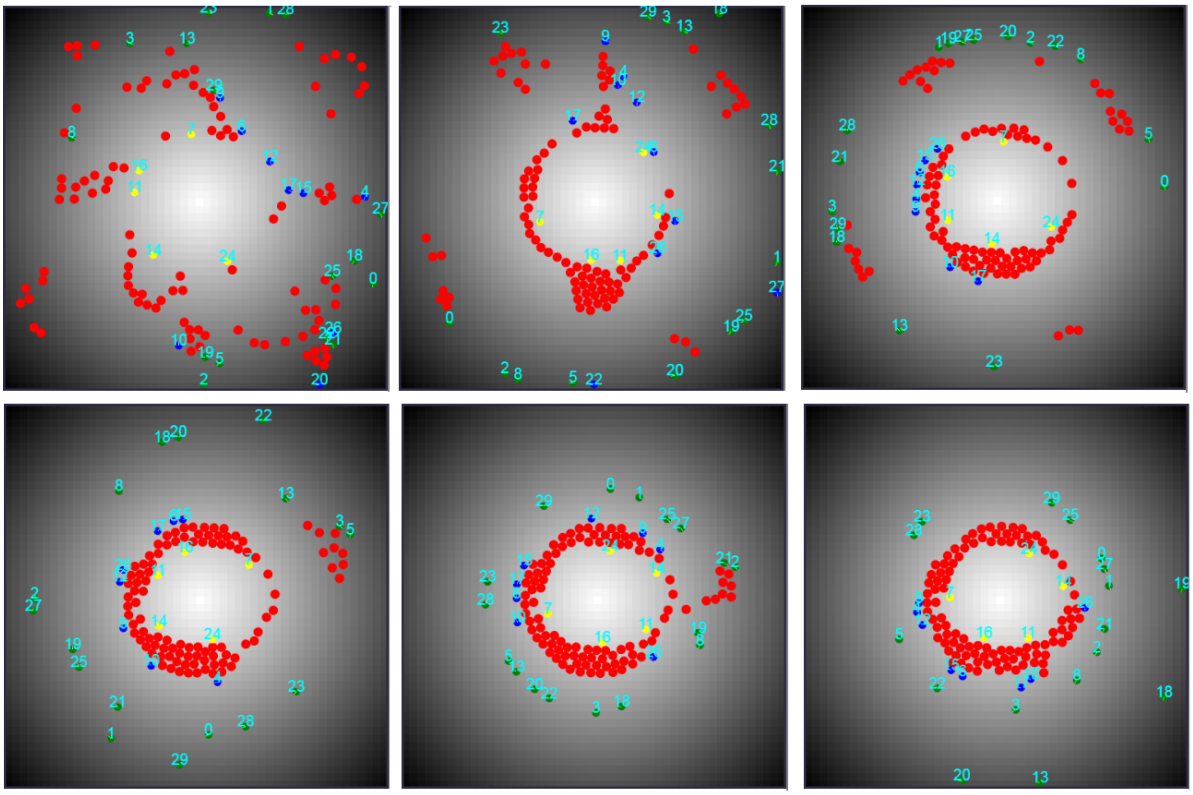


Fig. 3.3: Screenshots of simulation environment using 100 pucks and thirty robots consisting of five innies (yellow), eight outies (blue) and seventeen explorers (green). The explorers use global communication approach to form the annulus.

3.1.3.2 No-direct Communication

In this approach, the exploring robots calculate their response function based on Equation 3.2. They do not directly communicate. Once an explorer senses its orbital is empty, it switches to the next orbital towards the annulus. When an explorer reaches the annulus' boundary, it goes back to the environment boundary and starts scanning again.

The benefit of this approach is that there is no requirement for direct communication among the swarm.

3.1.3.3 Local Communication

In a case where the orbital is fully occupied with pucks, the explorer multicasts this orbital's status to its neighbors by setting its light on to attract other robots to come and help it in this orbital. If any of the others finish working on their orbital, they look for any light on their left side using their left light sensor, (see Fig. 3.4). If there are no detected lights on its left, It increases its orbital towards the shape contour, searching for detached pucks. However if they find the light on, they go towards this orbital by decreasing their τ and turn on their lights. After they push these pucks from this orbital, they will switch off the light and go to the next orbital, and so on. The displayed light could be designed as an LED light strip placed around the robot's chassis. The left light sensor could be an RGB omnidirectional camera mounted on the top of the robot, and we analyze the left half of images searching for the respecified lights. The pseudocode is presented as Algorithm 2.

The advantage of this approach is that the robot searches for help from the very closest neighbors, and they will come directly and help it.

Algorithm 3 Local Communication

OrbitalNum = RandomInteger(1 to 6) ▷ Select random Orbital number
$PrevTime = System.getTime()$
while $OrbitalNum < 7$ **do**
    $\tau = 0.1$ OrbitalNum ▷ Convert to the corresponding scalar field
    $\omega_{avg} = ExplorerOrbitalConstruction(obs_l, floor_l, floor_c, floor_r, pucks_l, pucks_r)$
    $Radius = \frac{H}{2} - (R_w(\tau_i R_n - 1))$ ▷ H environment height, R_w orbital width and R_n Total number of orbitals
    $c_i = 2\pi Radius$ ▷ The circumference for each region
    $\theta_i = \frac{C_i}{|\omega_{avg}|}$ ▷ The intensity threshold for each orbital
    $Avg\Delta T, PrevTime$= MeasureConsecutivePushes($\Delta T, \theta_i$, $PrevTime$)
    $F_\theta(Avg\Delta T) = e^{\frac{-40 Avg\Delta T}{\theta}}$ ▷ Local Response function for each robot
    if $F_\theta(Avg\Delta T) < 0.1$ **then** ▷ Orbital is empty
        $Robot.Light = "Off"$
        $PrevTime = System.getTime()$
        if $Light_l$ **then** ▷ Detecting Lighting on robot's left side
            $OrbitalNum = OrbitalNum - 1$ ▷ Moving to its left orbital to help the robot there.
        else
            $OrbitalNum = OrbitalNum + 1$ ▷ Increasing its orbital to search for available pucks.
    else if $F_\theta(Avg\Delta T) > 0.7$ **then** ▷ Highly occupied orbital
        $Robot.Light = "On"$
OrbitalNum =1 ▷ Robot starts scanning from beginning

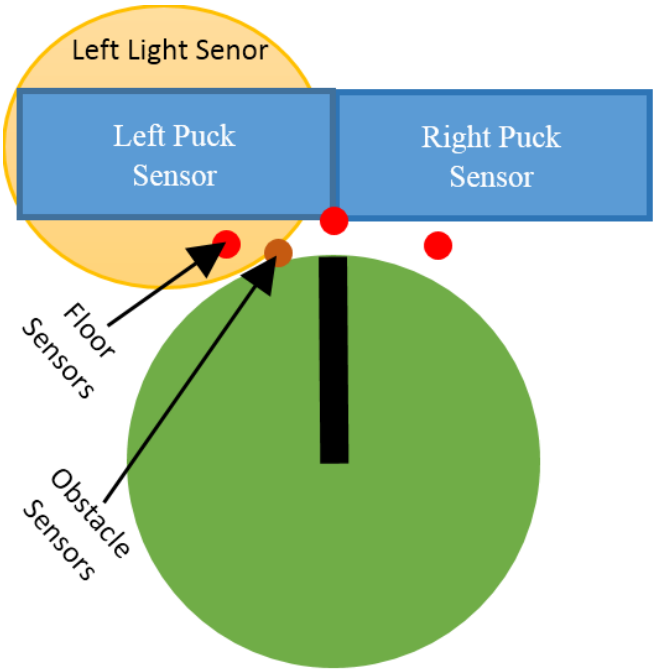


Fig. 3.4: Explorer's Sensors.

3.1.4 Experiments and Results

We addresses the challenge in [29] when trying to construct an annulus with 100 pucks. The author found some pucks near to the border environment were unreached by outies, as shown in Fig. 3.1. Our proposed methods were performed over ten trials with thirty robots and 100 pucks. The results were generated from a javascript simulator called Waggle [32].

Following the original algorithm [29], the scalar field has a single point source at the center and is used to split the environment into ten equal orbitals which are uniquely identified by their light intensities $\tau \in [0.1,1]$. Innies and outies operate on $\tau \geq 0.75$ and $0.6 < \tau \leq 0.7$, respectively, while explorers operate in $\tau \in \{0.1K | K \in \{1,2,3,4,5,6\}\}$. We use the average Euclidean distance between pucks and desired annulus' contour as a performance metric to show how this distance decreases over time steps across these approaches and OC algorithm. The heavy traces in Fig. 3.5 show the average distance

over ten trials for each approach separately. Fig. 3.5(a) shows the Euclidean distance decreased, but after some time steps it becomes stable, meaning that the outies can not reach the remaining pucks. However, Fig.3.5(b), Fig.3.5(c) and Fig.3.5(d) show Euclidean distances of the proposed methods decreased until they reach zero, which means all pucks are pushed and form the annulus.

Fig. 3.6 shows a comparison between OC, global, no-direct, and local communication. The OC algorithm has the worst performance; the average distance is stable after some time steps, and if the pucks are placed far away from the shape, the robots will not be able to push them. In the case of the global communication approach, the highest average distance is at the beginning of the simulation; then it significantly decreased at the end. This is because, in the beginning, all robots distributed randomly to discover the environment and exchange their response threshold values. Once they found the most occupied orbital, their movement may cause two problems: firstly, it may nudge the pucks outwards, especially if some of the explorers are working near to the annulus and the others notify them to go and help in outer orbital. Secondly, robots waste their time in switching between their orbitals and other far away orbitals. On the other hand, near the end of the simulation, most of the pucks have already been pushed toward the desired orbital contour, so the environment is quite clean. In this case, when robots go towards the most occupied orbital, it helps to push these pucks quickly. As a result, the average Euclidean distance is significantly decreased. For instance, the global-communication approach has the lowest value for the average Euclidean distance after 7000 steps.

In the case of the no-direct communication approach, it consumes more time to push all pucks because every robot works alone, and they only depend on their local calculations.

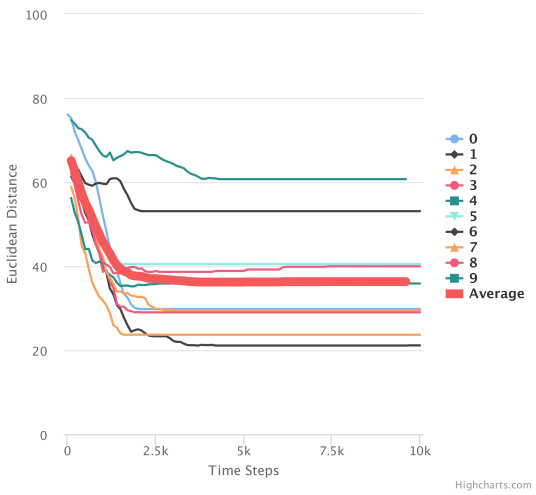
On the other hand, local communication takes advantage of sharing environment

information between nearby robots and solves the global communications problems by only switching to the very closest orbital. Therefore, local communication outperforms the two communication strategies in terms of execution time, but all can form the annulus correctly.

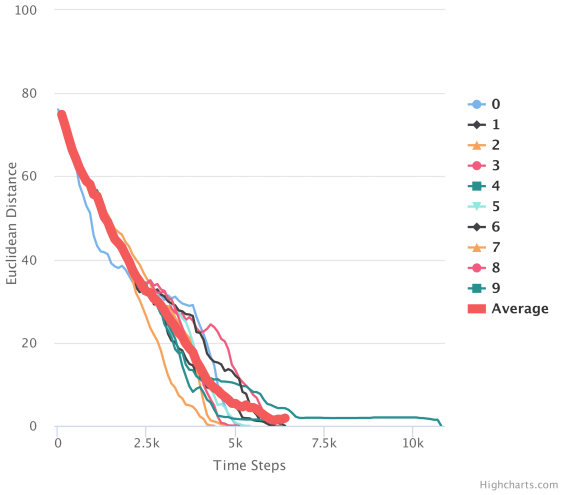
3.2 Conclusion

In this chapter, we present the use of the scalar field as a division of labour to distribute the pucks among the robots. Additionally, the scalar field defines the shape contour, and the robots push the pucks toward that contour.

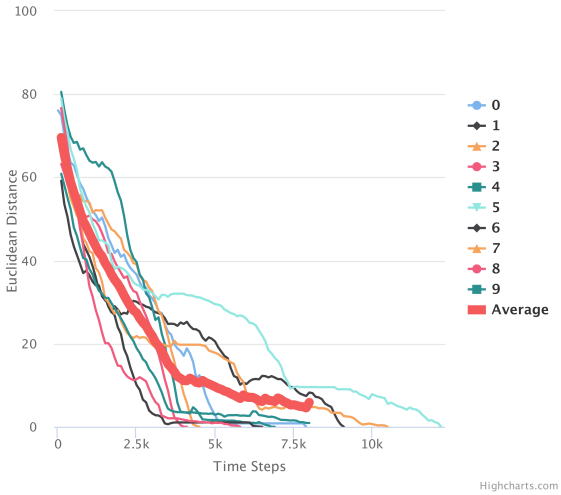
We show how these robots coordinate to construct an annulus shape. Three different communication techniques are discussed; local communication can outperform global and no direct communication. The working environment is split into six orbitals with different τ, and the robots use the response threshold function to guide them on when they should switch their orbital to the next. Compared to the OC algorithm, these techniques successfully construct the annulus, and all pucks are attached.



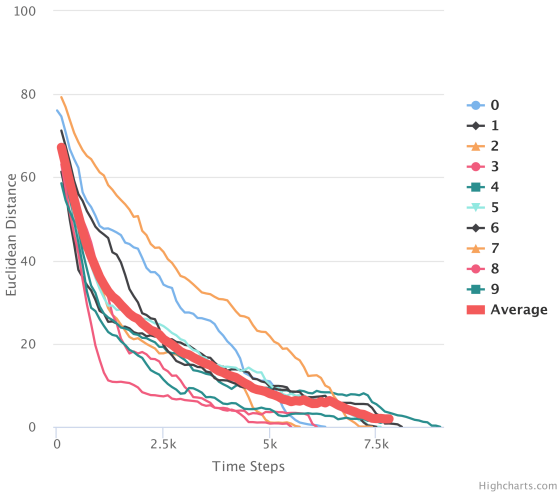
(a) OC Algorithm



(b) Global Communication



(c) No-Direct Communication



(d) Local Communication

Fig. 3.5: The average distance between pucks and the annulus over ten individual trials as well as the average across trials in the thicker trace with 100 puck using the OC algorithm and different communicating approaches: Global Communication, no-direct communication and local communication.

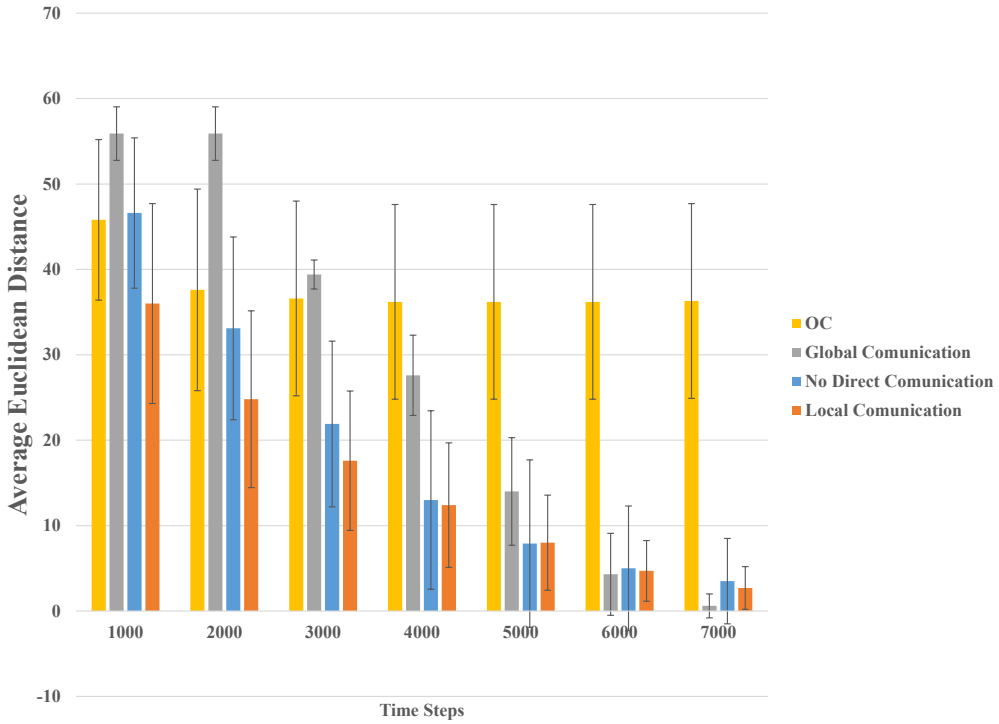


Fig. 3.6: Comparison between three types of communication among explorers and OC algorithm in terms of time steps and average Euclidean distance calculated over ten trials. The error bars indicate 95% confidence intervals.

Chapter 4

Guiding Robot Movement with Scalar Fields and Enhancing Performance Through Reinforcement Learning

In this chapter, we demonstrate how scalar fields are used as movement guidance for robots. We take the largest coverage network (LCN) as an example of collective behavior. Additionally, we explore the application of reinforcement learning (RL) to enhance the performance of the Largest Coverage Network in the presence of high and low-resolution scalar fields. We then compare the RL results with those of hard-coded implementation algorithms. This chapter presents parts of the work published in my paper:

- Dalia S. Ibrahim, and Andrew Vardy. "Largest coverage network in a robot swarm using reinforcement learning." Artificial Life and Robotics 27.4 (2022): 652-662.

4.1 Finding the Largest Coverage Network among Robots

The objective of establishing the largest coverage network among the robots is to allow them to gather more data and exchange it. This could be done by controlling the physical proximity between a swarm of robots. We present how the scalar field helps the robots to find their largest coverage network even with a low-resolution scalar field.

4.1.1 Introduction

Finding the largest coverage network can be helpful in many ways like each robot could be responsible for a specific territory, and if it faces a threat or needs help, it will send that information to its connected robots. Also, if there are some obstacles in front of some robots, they can propagate that information to others to avoid them. In the Largest Covering Network (LCN) task, the robots should be connected to one cluster with the least possible overlap to cover the largest possible working area. Each robot can detect another robot if the other robot is in its range. These swarm robots work independently in a distributed manner with very simple computational power.

We address the spatial coverage of mobile robots to achieve global connectivity with minimum overlapping sensing range. The robots are placed randomly without any central control. They collectively explore the environment and use the scalar field as guidance to build one connected network of robots. The scalar field resolution to be provided to the robots is an interesting quantity to explore. If a high-resolution scalar field (HRSF) is required, then this places certain constraints on the robots ability to sense this field. On the other hand, if a low-resolution scalar field (LRSF) suffices, it may be possible to implement our technique on simpler robots with fewer and less accurate sensors.

4.1.2 Related Work

In mobile robots swarm, Panerati et al. [33] select a master robot, and the swarm performs a rendezvous around that master robot as an essential step to establish the connectivity of the swarm inside the coverage area. In [34], Francesca at al. presented an automatic control design for a robot swarm. They tested their controller on five different robotics tasks with LCN as one of these tasks. Mitaka et al. use static templates to help the robots to perform the required behavior [1].

Solving LCN is also useful in the initial deployment of fixed sensors, such as a wireless sensor network. Based on the specific target wanted to be covered, the users calculate the sensors' locations to be sure that target is fully covered. The sensors can be deployed in an unstructured or structured way, depending on the application. In the structured deployment, the place of the sensors are predetermined with the advantage of less cost of maintenance and good coverage. However, the sensors are randomly distributed in the environment in the unstructured deployment, so more sensors may cover the same area [35]. Also, in [36], they show how the overlapped sensors can form a disjoint cover set, and the only disjoint sensor should be active at a time to save battery life.

4.2 Proposed Methods

The operated robots are equipped with five sensors, as shown in Fig. 4.1. The C, L and R sensors measure the intensity of light projected on the floor, which we named center, left, and right sensors, respectively. These sensors return the scalar value in a range from zero to one. The *Obs* sensor is the obstacle sensor to detect the presence of the walls and other robots. The *Com* sensor is the communication sensor that covers a circular region of the space around the robot. The *Pos* sensor is the position sensor that returns the robot's x and y coordinates and is used only in the low resolution scalar

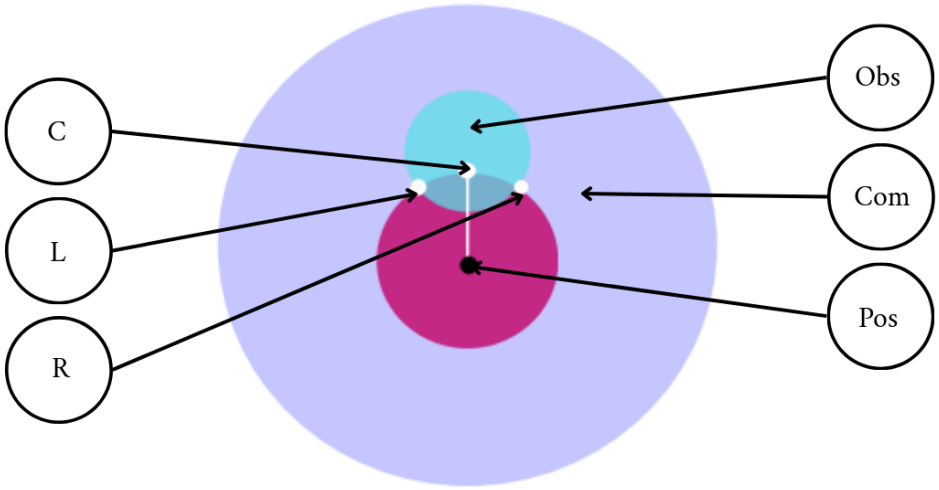


Fig. 4.1: Robot's sensors. C, L, and R are floor sensors. Obs is the obstacle sensor. Com is the communication range sensor. Pos is the position sensor.

field environment.

4.2.1 Low-Resolution Scalar Field (LRSF)

The robots depend on the readings of their communication range sensor and obstacle sensors. Using the scalar field guides the robots and gives them an idea of where they are.

In our proposed solution LRSF, we divided the working environment into four quadrants (Q1, Q2, Q3 and Q4). With this division, the robots could explore the whole environment, switch among different quadrants, and take random positions in these quadrants, giving them more options to find their best positions. At the beginning of the simulation, each robot is placed randomly in a random quadrant.

The robot moves to a random position in the lower integer number. For example, if a robot is placed in the third quadrant, it will move to a random position in the second quadrant. Any robot located in the first quadrant keeps selecting positions randomly and waiting for any other robot to join its cluster. Once two robots or more form a small cluster in the first quadrant, the robots in this cluster start to move backward to

reduce the intersection area among them using the backward speed as shown in Equation 4.1. We added a random floating number]0,1[to the backward speed to ensure the two intersected robots take different values to reduce their intersection areas.

$$BackwardSpeed(t) = -1\left(e^{\frac{MaxIntersectedArea(t)}{Robot'sRange}} + RandFloat(0,1)\right) \tag{4.1}$$

If they are connected with less than or equal to 2% of their connected range (maximum allowed intersection), they stop moving and wait for other robots to reach them, whether the coming robots reach the first quadrant or are on their way. We use the first quadrant as an anchor, so the robots' goal is to reach the first quadrant and start to form the cluster from it.

The control algorithm is presented as Algorithm 4 with the function getNextMove provided separately. This function calculates the forward speed and angular velocity required for the robot to move from its current position to the selected random position in the low integer quadrant.

Fig. 4.2 provides an example which demonstrates the robot's movements to establish the one connected network. The colors of robots show which quadrants it belongs to, the yellow color means this robot stop moving, and the black arrows indicate the robot's actions on the next move. In Fig. 4.2 (A), robots in Q2 to Q4 choose to decrease their quadrant number, hoping it might find any other robots. Robot #1 in Q1 waits for other robots to come to its quadrant, so it selects a random position; perhaps it finds other robots. In (B), robot #2 reaches the Q1 and finds robot #1, but their intersection area is greater than 2% of the robot's range size. Therefore, they choose to move back to reduce their intersection area with backward speed, as shown in the Equation4.1. Robot #3 and #4 are still reducing the number of their quadrants searching for other robots. In (C), robots #2, #3, and #4 are connected with big intersection areas, so they choose

Algorithm 4 Hard-Coded for LRSF

Input: Robot's Position, $\overrightarrow{Pos}$
Obstacle sensor's reading , obs
Number of neighbors, $Neighbors$
Max intersection with neighbors, A_{Max}
Robot's steer angle, $currAngle$
Output: Forward velocity, v
Angular velocity, ω
State variables: Robot's quadrant, Qid

$\rho \leftarrow 0.02 \times Robot'sRange$ ▷ Minimum Intersection
if obs **then** ▷ Avoid Obstacles
  return ω_{max}
if t is 1 **then** ▷ Beginning of the simulation
  $Qid \leftarrow getRandomQuadrant()$
else if $A_{Max} > \rho$ **then** ▷ robot constructs sub cluster
  if $(\ (Qid == 1 \wedge Neighbors \geq 2\ ) \vee Neighbors \geq \frac{Robots}{2}\ )$ **then**
    $v \leftarrow BackwardSpeed$ ▷ move backwards
    return v, ω
else if $A_{Max} \leq \rho \wedge Neighbors \geq \frac{Robots}{2}$ **then**
  ▷ robot connected with min intersection
  $v, \omega \leftarrow 0$ ▷ Stop movement
  return v, ω
else if $Qid \neq 1$ **then**
  $Qid \leftarrow Qid - 1$ ▷ Assign Qid to the lower integer
$\overrightarrow{NewPos} \leftarrow getRandPos(Qid, \overrightarrow{Pos})$
$v, \omega \leftarrow getNextMove(\overrightarrow{Pos}, \overrightarrow{NewPos}, currAngle)$
return v, ω

Function getNextMove

Input: Robot's Position, $\overrightarrow{Pos}$
Destination's Position, $\overrightarrow{NewPos}$
The robot's steer angle, $currAngle$
Output: Forward velocity, v
Angular velocity, ω

$\Delta X \leftarrow \overrightarrow{NewPos}.x - \overrightarrow{Pos}.x$
$\Delta Y \leftarrow \overrightarrow{NewPos}.y - \overrightarrow{Pos}.y$
$Kv \leftarrow 0.01$
$v \leftarrow Kv \times \sqrt{\Delta X^2 + \Delta Y^2}$
$Angle \leftarrow Atan2(\Delta Y, \Delta X)$
$Kh \leftarrow 0.5$
$\Delta Angle \leftarrow Angle - CurrAngle$
$\omega \leftarrow -Kh \times \Delta Angle$
return v, ω

to move backwards. However, robot #1 is connected with a good intersection area, so it stops moving, and its color changes to yellow. Based on the actions in (C), robots #2 and #3 are still connected with a good intersection area as shown in (D), but the total number of robots in their range is less than the total operated robots. Also, robot #1 and #4 are disconnected. So, robots #2, #3 , and #4 decrease their quadrant numbers, and robot #1 gets a new random position in Q1. In (E), robots #3 and #4 are connected with a good intersection area, but robots #1 and #2 need to move back to adjust their intersection areas. In (F), all robots stop moving and establish one connected cluster with minimum intersections.

4.2.2 High-Resolution Scalar Field (HRSF)

In the previous section, we discuss the LRSF, where the environment is divided into four quadrants. However, in this section, we introduce the HRSF, where we divide the working environment into ten different parts (orbitals) where each orbital has different light intensity. We projected the scalar field on the working environment with a central light source. The grid value is between one and zero based on the intensity of the

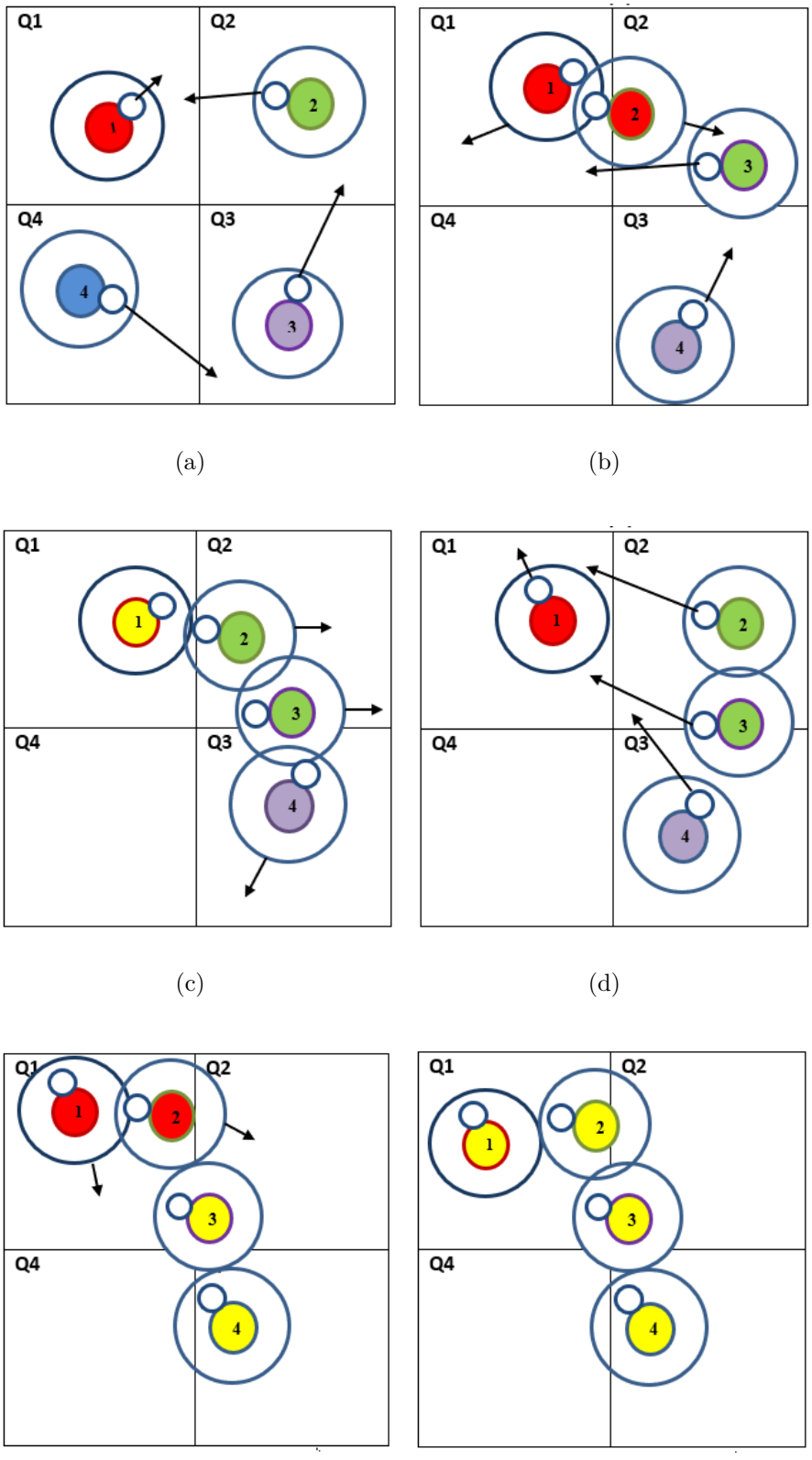


Fig. 4.2: The sequence of snapshots shows the hard-coded implementation in the presence of a low-resolution scalar field; the colored robots indicate which quadrants they are; the black arrows show the robots' next decision based on these situations.

projected light on the floor. The robots use the three floor sensors readings, which measure the scalar field value projected on the environment. With the scalar field's guidance, the robots move towards the lower scalar values. Once the robots reach the lowest scalar value, they save this value τ as the desired contour line to use in their circular movement. The robots follow the orbiting algorithm illustrated in [37], so all robots are aligned and orbit in a clockwise direction in the same contour line. The initial value of the forwarding speed is calculated based on the number of connected robots in its range Equation (4.2). If any robot detects another robot in its range using the communication range sensor, the robot's forward speed is reduced based on the number of robots in its range as shown in Equation (4.2). The more robots in a cluster, the less forward speed for the robots, which helps the other robots catch them. If the number of connected robots in its cluster equals the number of operated robots, the forward speed equals zero, and the robot stops moving.

$$ForwardSpeed(t) = MaxForwardSpeed\left(1 - \frac{RobotsInRange(t)}{OperatedRobots}\right) \tag{4.2}$$

Fig. 4.3 shows how the hard-coded algorithm works. The red, green, and yellow robots indicate the robots move clockwise, anticlockwise, or stop, respectively. The black arrows show the actions that the robots decide to take based on the current situation. In (A), all robots are distributed randomly in the environment and look to decrease the sensing scalar field projected on the ground. So that helps them to go to the orange outer ring (just for demonstration). In (B) and (C), the robots orbit clockwise and try to align with the lowest scalar value. The robots' speeds are based on the number of robots in their range, as shown in Eq (4.2). In (D), robot #3 connects to all other robots, and its intersection area is smaller than or equal to 2% of its communication range, so it stops moving. However, robots #1, #2, and #3 connect with a big intersection area, so

they try to reduce this intersection by orbiting anticlockwise with speed, as shown in the Equation 4.1. In (E), All robots should move anticlockwise to reduce their intersection area. In (F), all robots are connected and stop moving.

4.2.3 Experiments and Results

We used the C++ open-source simulator *CWaggle* [1]. This simulation is inspired by the Javascript robot simulator *Waggle*[2]. *CWaggle* was used before in [38] and [26].

The circular robots move in a rectangle space 900×900 and we use two performance metrics to evaluate the proposed methods. Firstly, the number of clusters constructed by the robots. Secondly, calculating the axis-aligned bounding box around the coverage area. We simulated all experiments over ten trials; the thick lines are the average performance for each proposed method, and the shaded areas are the confidence interval for the 90% confidence level.

Low-Resolution Scalar Field: The sequence of screenshots from the hard-coded algorithm with the presence of LRSF can be seen in Fig. 4.4. At $t = 1$, robots are distributed randomly over the four quadrants. At $t = 19$, Robots reaches their positions in their quadrants; robots #1, #2, #3, and #4 are connected with some intersection areas between them, so they move back to reduce these intersections. At $t = 26$, robots #1 and #4 have a good intersection area, so they stop moving, and their colors turn to yellow; however, robots #2 and #3 are still moving back, aiming to reduce their intersection and keep connecting to that cluster. As a consequence of robot #3 's movements, it has a big intersection with robot #4, so both starts moving back to reduce this intersection. After a while at $t = 6000$, robots #2 and #3 fix there intersections. Robots #5 and #6 move from their initial fourth quadrant to the third one and then reach the second quadrant. After that, they connect with the cluster, but they still

[1] https://github.com/davechurchill/cwaggle
[2] https://github.com/BOTSlab/waggle

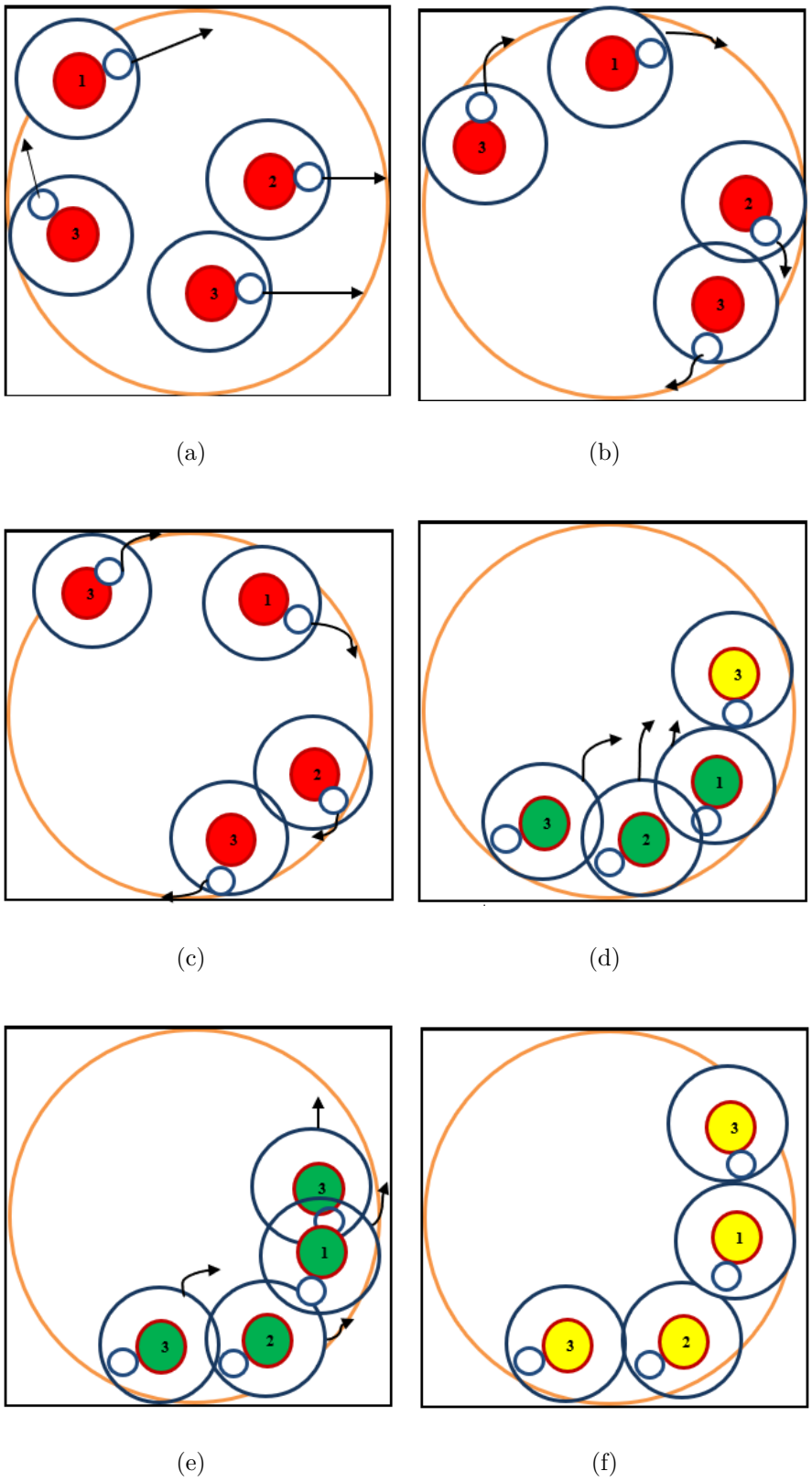
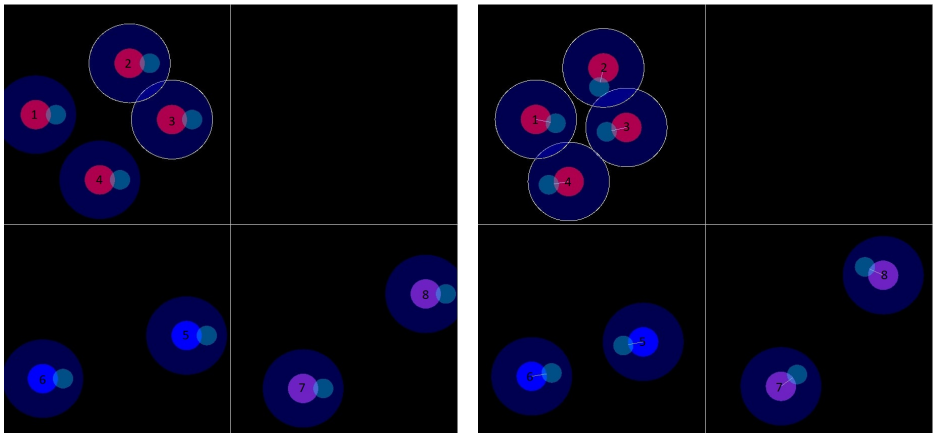


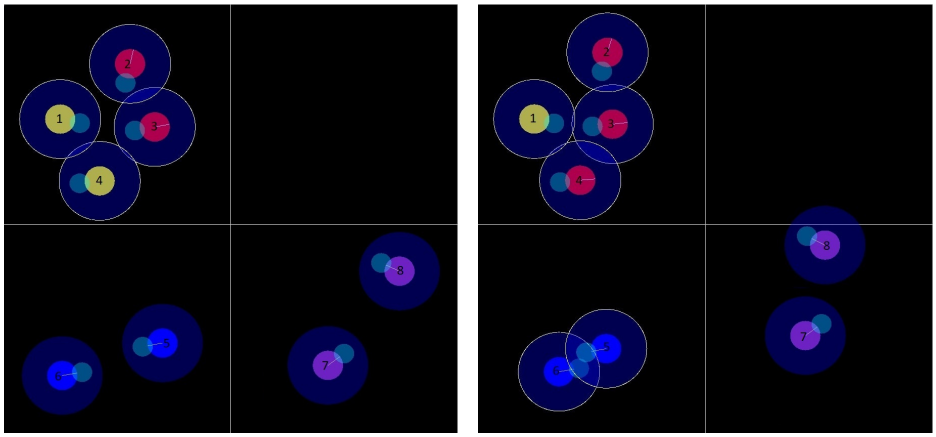
Fig. 4.3: The sequence of snapshots illustrates the hard-coded implementation in the presence of a high-resolution scalar field. The orange ring shows the lowest scalar field in the environment. The colored robots indicate the motion of the robots, either forward, backward, or stop; the black arrows show the robots' next decision based on the intersection between robots' communication range.

have an intersection that needs to be reduced. Robot #8 changes its quadrant to the second quadrant and connects with the cluster with the acceptable intersection, so it stops moving. Robot #7 reaches the second quadrant but with a big intersection with robot #8, so it moves back and disconnects from the cluster, so it moves to the first quadrant and intersects with robot #4. Finally, at $t = 10000$, all robots are connected with the smallest intersection except robots #4 and #7; they still move backward and forward to reduce the intersection and keep connecting with the cluster.

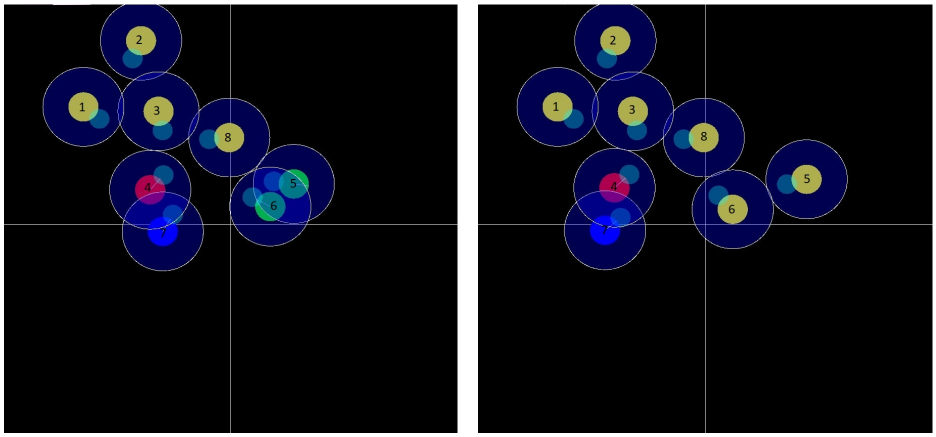
High-Resolution Scalar Field: Fig. 4.5 presents the screenshots of hard-coded implementation with the HRSF environment. At $t = 1$, the robots detect the scalar values and move with the guidance of the scalar field to reduce the sensing values. According to this motion, robots are aligned to the outer orbital near the border as shown in $t = 290$. The robots orbit in a clockwise motion, and some of them construct small clusters. So, robots in that cluster decrease their speed with the same ratio based on the number of robots in their cluster. At $t = 290$, the first group are consisting of robots #6, #7, and #8; and the second group has the robots #1, #3, and #4. Robots #5 and #2 are orbiting alone, so their speed is faster than the other robots so that they can catch them. After a while, robot #2 connects with robot #8, so their cluster speed is decreased, which allows robots #3, #4, and #1 to catch them as shown at $t = 1862$. At $t = 1868$, all robots in green color start to move anticlockwise to reduce their intersections, and robot #1 successfully gets a minimum intersection, so it stops moving. On the other hand, robot #5 in its moves clockwise to reach that cluster. At $t = 2498$, all robots construct one cluster, but they still fix the intersections among them by orbiting anticlockwise. Finally, at $t = 10000$, robots in yellow color stop motion while the green robots keep moving anticlockwise, aiming to fix their intersection area.



(a) $t = 1$ (b) $t = 19$

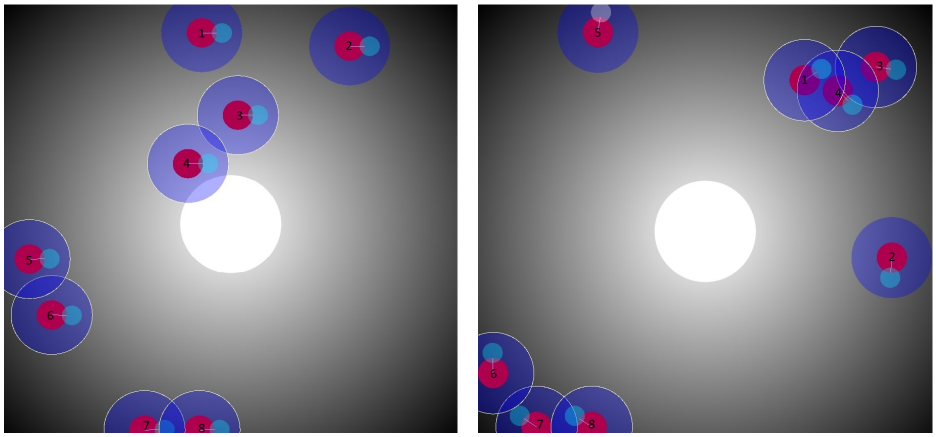


(c) $t = 26$ (d) $t = 50$

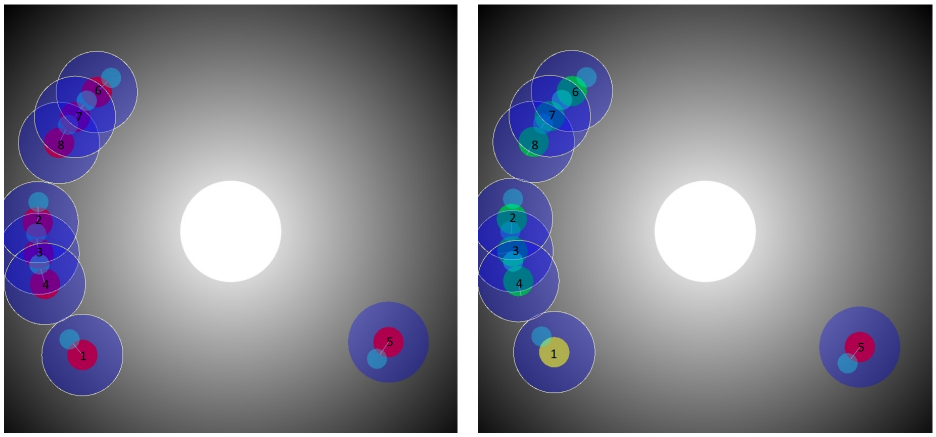


(e) $t = 6000$ (f) $t = 10000$

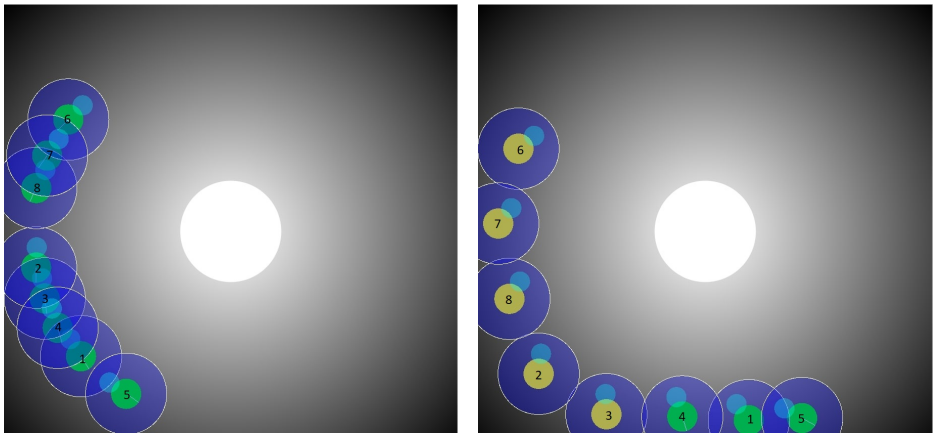
Fig. 4.4: Screenshots from the CWaggle simulation in different time steps. The working environment is divided into four quadrants. The colored robots indicate the destination quadrant the robots choose to go there. The colors are red, green, purple, and blue for the four quadrants respectively starting from the upper left corner. The yellow color indicates the robot chooses to stop.



(a) $t = 1$ (b) $t = 290$



(c) $t = 1862$ (d) $t = 1868$



(e) $t = 2498$ (f) $t = 10000$

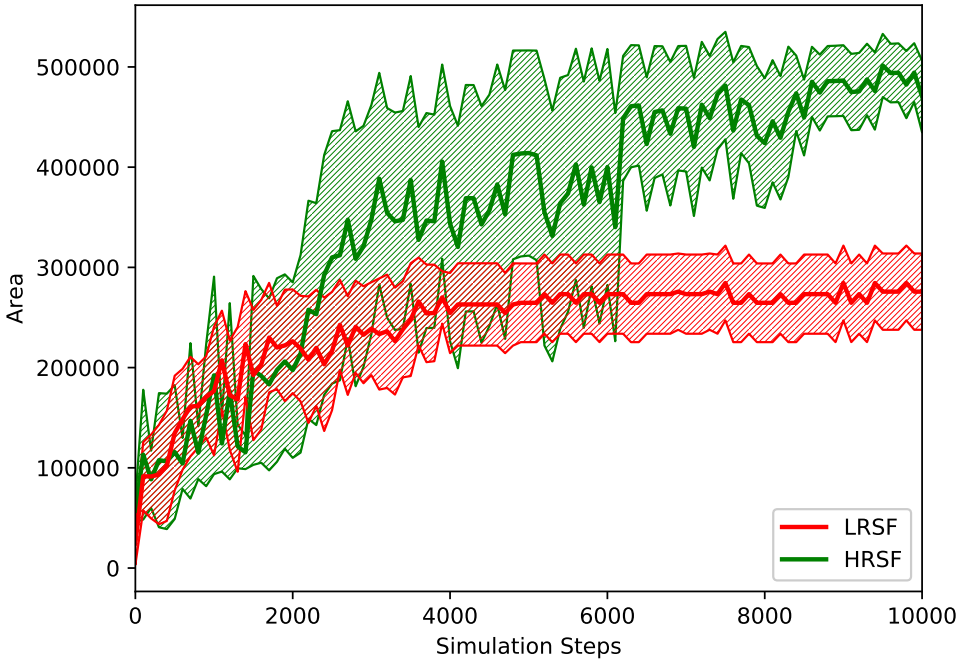
Fig. 4.5: Screenshots from a CWaggle simulation in different time steps. Eight robots are randomly distributed at $t = 1$, sharing the same orbital with different, forward speeds. The robots' colors are red, green, or yellow, indicating that robots orbit with forwarding, backward speed, or stop, respectively.

4.2.4 Discussion

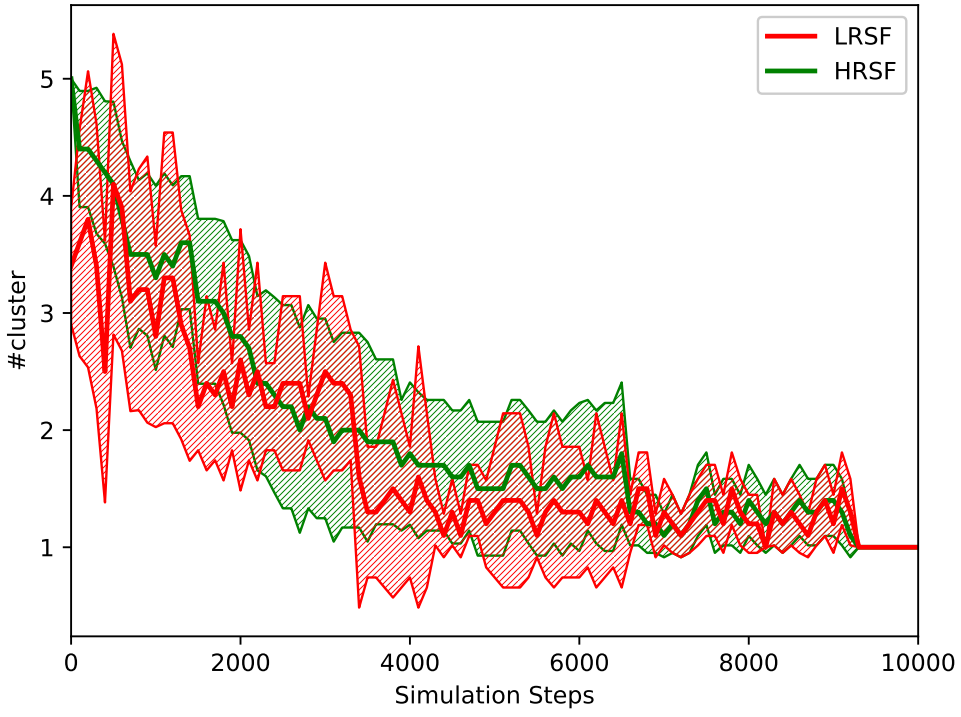
Fig 4.6 shows the performance of Hard-coded implementation in HRSF and LRSF. Fig. 4.6(a) presents the obtained coverage area in both implementations over ten trials. The area in HRSF reaches nearly 500000 by the end of the simulation; however, in LRSF, the coverage area is nearly 300000. The HRSF helps robots find the furthest orbital, and they keep orbiting in that orbital and adjusting their speed based on the number of the robots in their cluster so they construct a larger area faster than the LRSF. On the other hand, in LRSF, the robots have only four quadrants and try to enlarge their cluster size by moving back to reducing the intersections. The advantage of LRSF is it gives us different coverage shapes in every run, so it could be used if we want robots to cover other territories. Fig. 4.6(b) shows the number of clusters versus the simulation time. In HRSF, the robots create one or two groups earlier than the LRSF and focus on decreasing the area of their intersections, shown at t = 3800.

4.3 Using Reinforcement Learning to Enhance LCN Performance

As mentioned earlier in Section 4.2, LCN aims to maximize the coverage area by a swarm of robots while maintaining the connection among these robots. Robots try to form one connected cluster with minimum intersection areas among them. In this section, we propose to use Q-Learning as an RL algorithm for solving LCN in two different environments; low-resolution scalar field (LRSF) and high-resolution scalar field (HRSF). We prefer to study tabular Q-learning over any other reinforcement learning algorithms because we can discretize the state and action space into small and discrete spaces. Tabular Q-learning will be simpler and converge to the optimal policy with fewer training samples because it stores Q-values for each state-action pair in a table, making



(a)



(b)

Fig. 4.6: Hard Coded implementation in the presence of low and high resolution scalar field.

the update quicker. Also, tabular Q-learning is easy to interpret how the agent makes decisions, which helps in understanding the agent's behaviour [39].

In RL, where some agents interact and modify the environment, tabular Q-Learning shows promising results when applied in a dynamic environment [38], [40]. Its goal is to find the best sequence of actions to get the maximum outcome as much as possible. the agent can explore and get some observations from the environment. It will interact with the environment by using a specific action and learn by getting a reward and a new observation from the environment based on that action. Based on that new observation, the agent decides to repeat or avoid that action. It keeps doing this learning process until it has a good action sequence to achieve its goal. This policy can be iteratively enhanced through multiple simulation runs.

4.3.1 Low-Resolution Scalar Field (LRSF)

As mentioned in Section 4.2.1, which presents the hardcoded implementation for LCN, we use the same configurations in which we divide the working environment into four quadrants. In this section, we use the RL to handle the switching between these quadrants. The states for the Q-Learning algorithm are determined based on a robot's point of view and how many neighbors are in its range. Besides, the robots have limited memory and computation capabilities; we tried to find the minimum possible representation for state and action space. Therefore, we discretize the many states into four states S ,

where n is number of neighbors and TR is the number total robots.

$$S_i = \begin{cases} n = 0 \\ n \neq 0 \wedge n < \frac{2}{3}TR \\ n \geq \frac{2}{3}TR \wedge n < TR \\ n = TR \end{cases}$$

The robot chooses between eight actions:

- Normal movement in which it goes to a random spot in the lower integer quadrant (1 action).
- Selecting any other quadrant from the environment (Q1, Q2, Q3, Q4) (4 actions).
- Increasing or decreasing its forward speed (2 actions).
- Stop (1 actions).

In the reward function, we focus on maximizing the number of connected robots and increasing the coverage area as much as possible by minimizing the intersection area between neighbors.

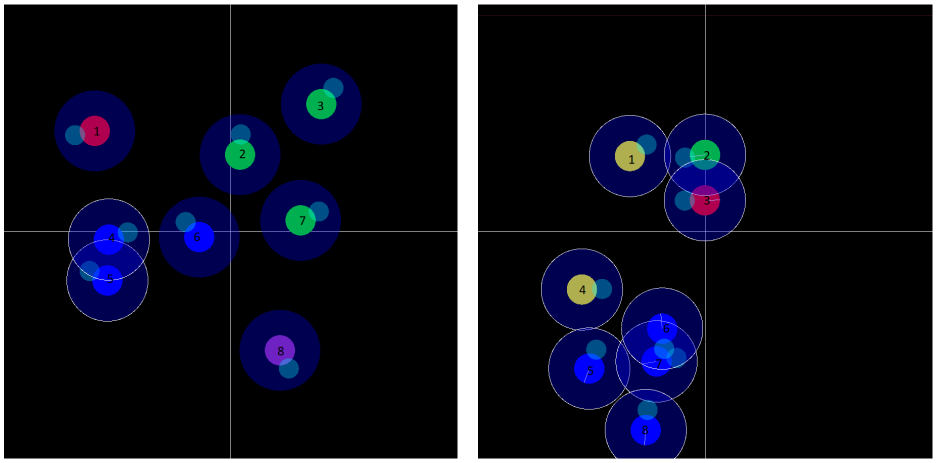
$$R_{\mathrm{i}}(t) = \frac{ConnectedNeighbors(t)}{OperatedRobots} + \frac{CoverageArea(t)}{MaxCoverageArea} - 1 \tag{4.3}$$

As shown on Equation (4.3), in the worst scenario, when a robot moves alone and does not belong to any cluster, the reward function approximately equals -1. When all robots are connected, the reward value is directly proportional to the coverage area.

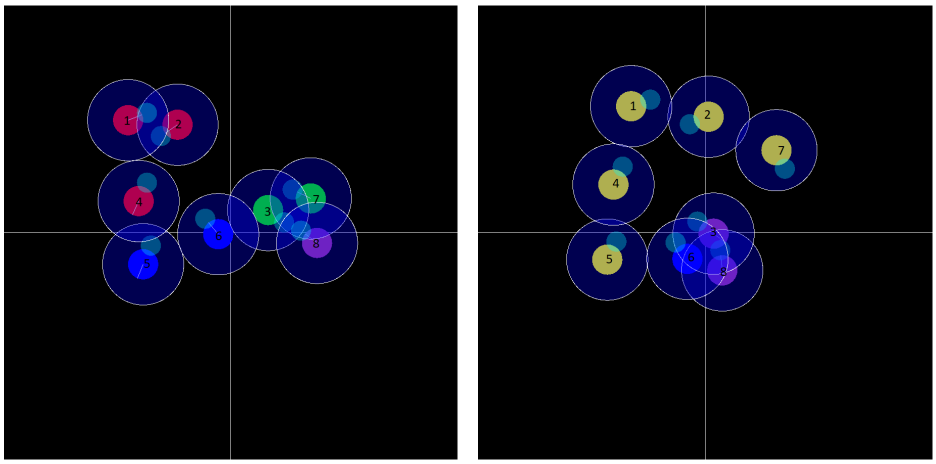
Fig. 4.7, shows screenshots from the RL implementation in the presence of LRSF in operation. At $t = 1$, the robots are on their way to their quadrants. At $t = 859$, some of the robots (yellow color) achieve their goal by connecting to others with minimum intersection areas. All other robots suffer from significant intersection areas, so they decide to continue searching for the best position for them. When these robots move away from that cluster, it reflects in the others' reward function, so all the robots continue working again, as shown at $t = 2507$. After some tries, they can form a cluster with a different shape at $t = 3805$. But there are still three robots that have a big intersection area. At $t = 4795$, based on their learning policy, they can take actions that increase their reward; in this case, they decided to move to Q4. Finally, all robots fix their intersections and connect to one cluster at $t = 6097$.

4.3.2 High-Resolution Scalar Field (HRSF)

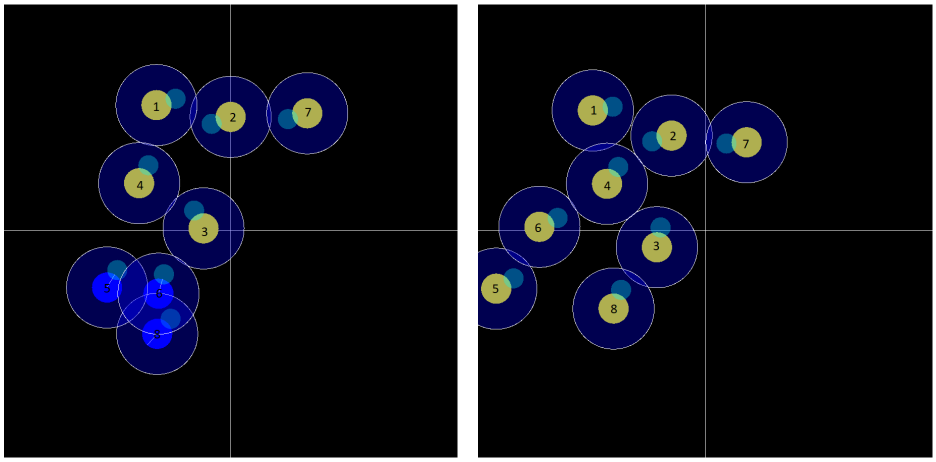
We use the same configuration presented in Section 4.2.2 for hardcoded implementation in which we projected the scalar field with to central light source on the working environment. Then, we discretize the scalar field to ten values $\in [0.1, 1]$ representing different light intensities; the area has the same light intensity called orbitals. The robots go towards the furthest orbital $\tau = 0.1$ to achieve the largest coverage network. But the robots share the same orbital so that the robots can be connected at any time because they move with different forward speeds based on the number of their connected neighbors. Therefore, the states for Q-learning are defined based on the maximum intersection areas between the robots and their connected neighbors. The states are discretized into four states: (1) there is no intersection between this robot and the others; (2) the maximum intersection area between this robot and its neighbors is greater than two-thirds of its coverage range; (3) the maximum intersection is greater than one-third; or (4) the maximum intersection is less than one-third of its coverage range.



(a) $t = 1$ (b) $t = 859$



(c) $t = 2507$ (d) $t = 3805$



(e) $t = 4795$ (f) $t = 6097$

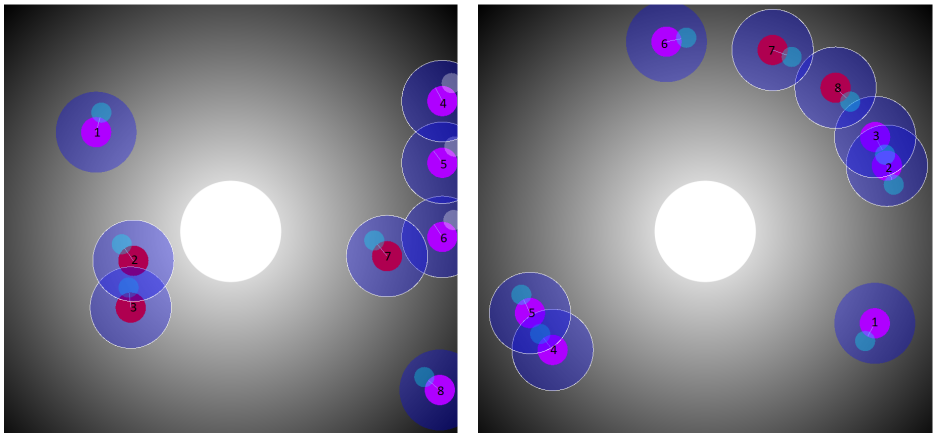
Fig. 4.7: Screenshots in different time steps show how the robots perform in the Cwaggle simulation using RL.

The robots choose between three actions: (1) Moving with the calculated forward speed as shown in Equation (4.2); (2) increasing their forward speed by 0.3 to catch the other robots; (3) stopping their movement. We used the same reward function defined in Equation (4.3) in the LRSF environment.

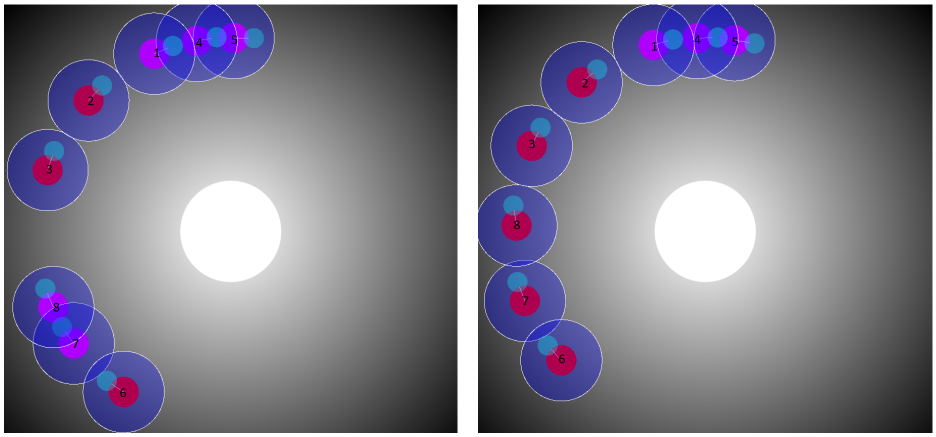
Fig. 4.8 shows screenshots of the RL algorithm with HRSF from the CWaggle simulator in different simulation time steps. At the beginning of the simulation $t = 1$, all robots are distributed randomly in the environment. After that, they reach the lowest scalar value. These robots choose between actions: some increase their forward speeds, moving with the calculated forward speed Equation 4.2 or stop movement, and the robots are colored purple, red, or yellow, respectively. According to their learned policy, at $t = 1000$, the purple robots that move alone try to increase their speed to catch other robots. Also, the robots with the maximum intersection with its neighbor try to increase their speed to reduce the intersection area. At $t = 4100$, and $t = 4200$ the red robots have a minimum intersection, but the number of connected robots in their cluster is less than the total number of connected robots. So, they move with the calculated forward speed, while the purple robots increase their speed to reduce their intersection areas. At $t = 4500$, the yellow robots have a minimum intersection area, and the number of robots in their cluster equals operated robots, so they get a high reward and choose to stop their movement. Simultaneously, some robots are still moving to reduce their intersection areas, as shown in red. Finally, all robots stop and form one cluster with a minimum intersection area at $t = 4560$.

4.3.3 Experiments and Results

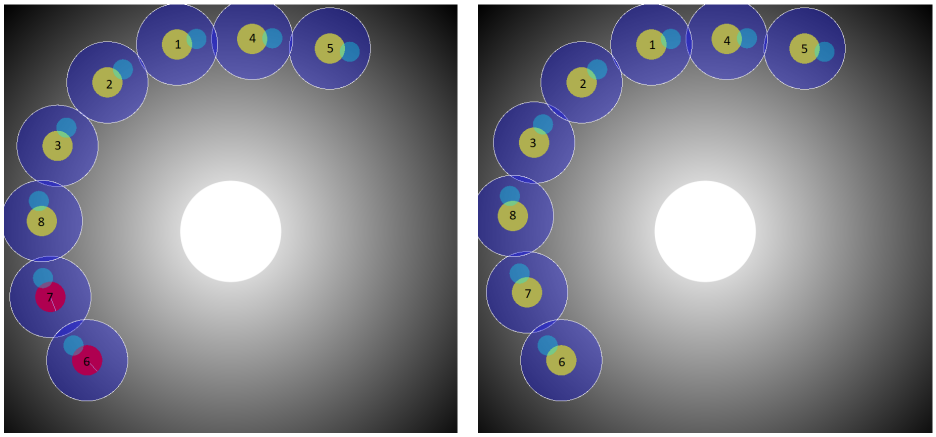
We follow the same simulation environment described in the previous Section 4.2.3. For all RL experiments in LRSF and HRSF, we use a discount factor equal to 0.9 with learning rate varying between experiments.



(a) $t = 1$ (b) $t = 1000$



(c) $t = 4100$ (d) $t = 4200$



(e) $t = 4500$ (f) $t = 4560$

Fig. 4.8: Screenshots show a CWaggle simulation in different time steps. The eight robots are using RL to form LCN and working on a central source point scalar field. The colors of the robots indicate the action they choose; the red color means the robots move with the calculated forward speed; the purple color indicates that the robots increase their speed and the yellow color means the robot chooses to stop.

Fig 4.9 shows the RL results in both environments, the LRSF and HRSF. The top figure presents the number of formed clusters versus the simulation time steps; the results depict that using RL in both LRSF and HRSF constructs one cluster after $t = 7500$. However, in the coverage area plot, we can note that the RL–HRSF can achieve a larger coverage area than RL–LRSF at most time steps. This is because, in HRSF, all robots move in a narrow orbital, which helps the robots decrease the intersections among them; reducing the intersections leads to achieving more coverage area.

Fig. 4.10 shows the comparison between RL and hard-coded implementation, which is presented in the Section 4.2.1, with LRSF regarding the number of connected clusters and the maximum area achieved. The RL implementation can construct one cluster faster than the hard-coded implementation, and that cluster is more stable from t =5700. Unlike the hard-coded implementation, it can build one connected cluster, but due to its several backward movements to fix the intersections, this cluster is not stable like RL.

Regarding coverage area, the RL can outperform the hard-coded implementation, and its average coverage area equals 290000. However, in hard-coded implementation, the average coverage area equals 270000 by the end of the simulation time.

Fig. 4.11 presents the differences in the performance between both implementations. The top figure shows that most of the trials in the RL implementation can construct one cluster around $t = 3000$ simulation step, which is faster than the hard-code. However, In some trials, robots destroy this cluster while moving to minimize the intersection areas, and others keep it because it has minimum intersection areas among the robots. Regarding the average coverage area, the RL implementation got a larger coverage area than the hard-coded implementation.

The comparison between all the proposed solutions is shown in Fig. 4.12. Regarding number of clusters, all proposed algorithms start with high number of clusters and by the end of the simulation they can establish one cluster. However, the RL implementation

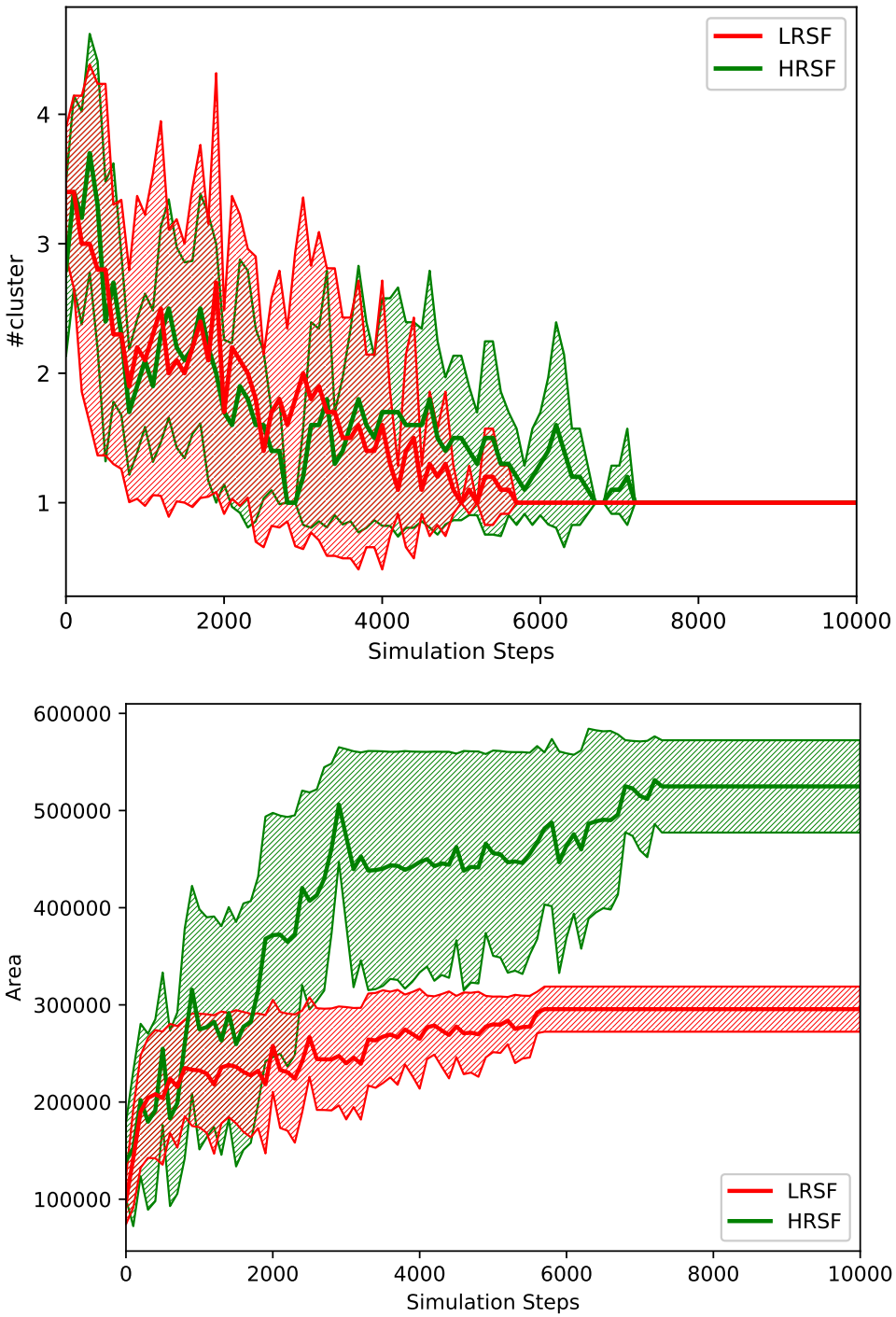


Fig. 4.9: Comparison between Reinforcement learning implementation using Low and high Resolution Scalar Field.

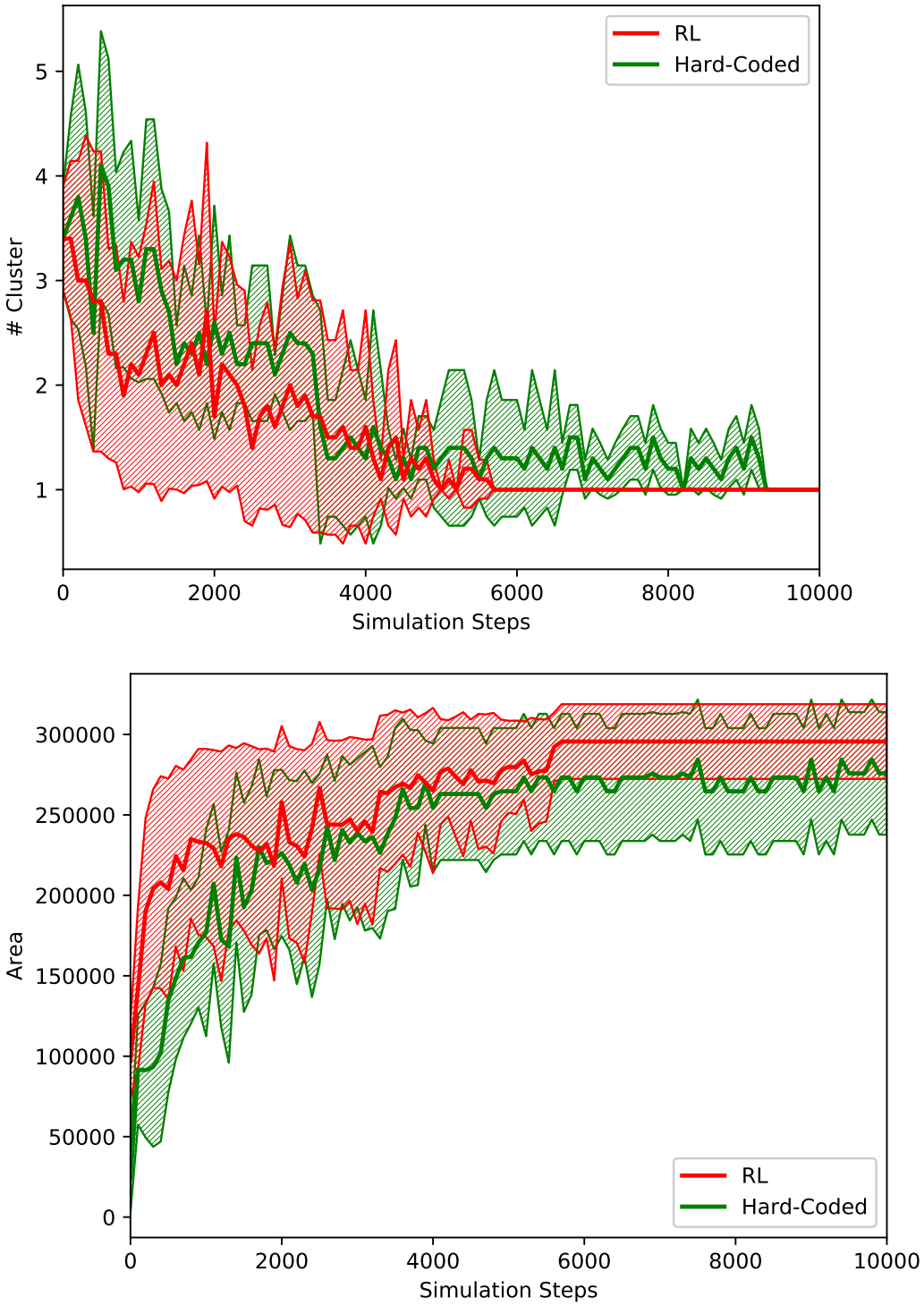


Fig. 4.10: Comparison between Hard-Coded and Reinforcement learning implementation using Low-Resolution Scalar Field.

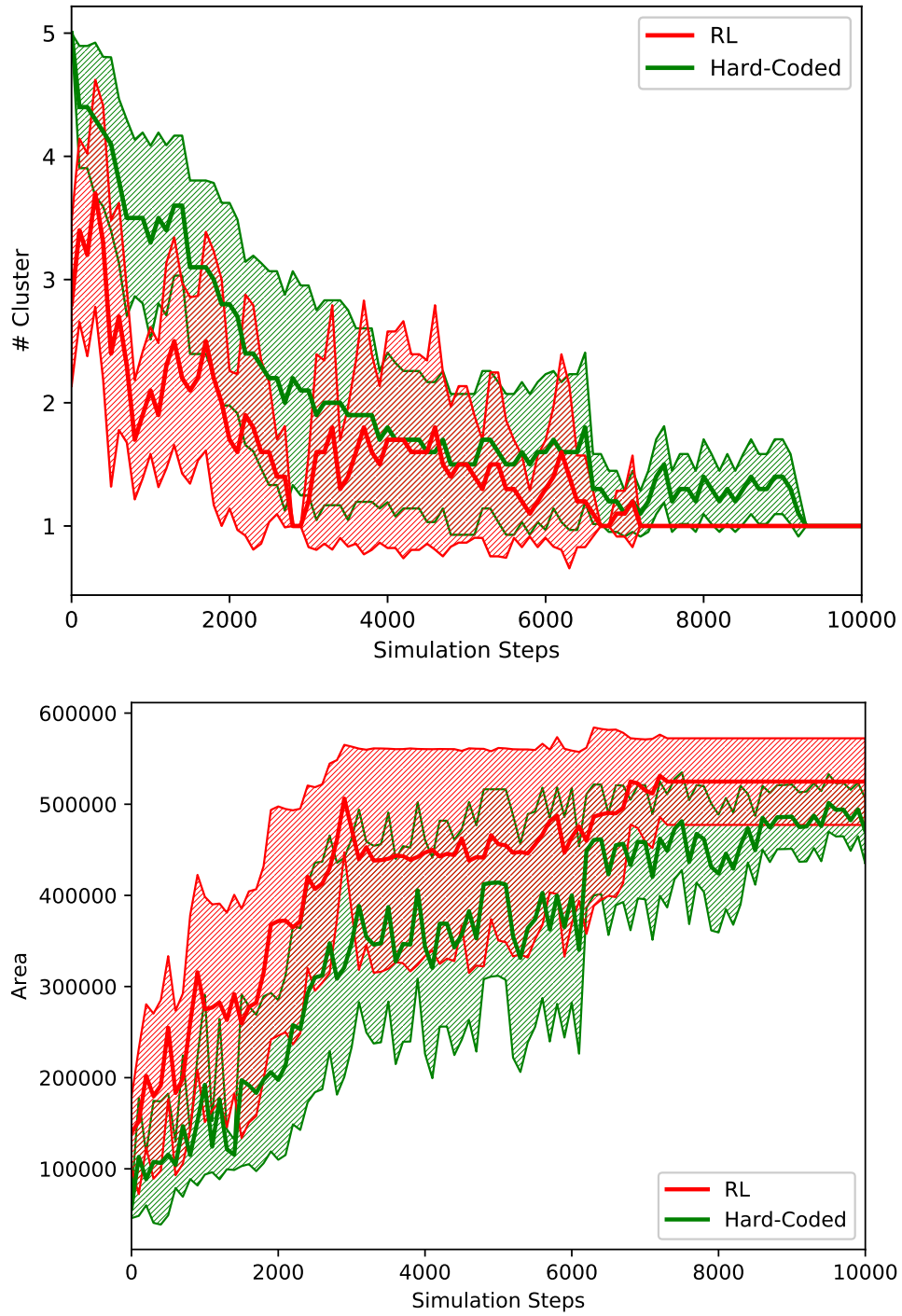


Fig. 4.11: Comparison between Hard-Coded and RL using High-Resolution Scalar Field.

with LRSF takes the lowest simulation steps to form one stable cluster. Also, in LRSF, every trial produces different shapes that depend on the initial position of the robots. Therefore, this solution will be preferable if the application wants to divide the working environment into arbitrary territories. In respect of the average coverage area, RL with HRSF achieved a higher coverage area compared to the other implementations.

4.4 Conclusion

In this chapter, we show the scalar field in two resolutions to guide the robots in finding the largest coverage area. The high-resolution scalar field (HRSF) helps the robot to find the furthest orbital and align into the same orbital, so they focus on reducing the intersection among them. Also, we introduce the low-resolution scalar field (LRSF), which could be used for robots with limited sensing capability; the robots can successfully construct one cluster in a reasonable time compared with the HRSF. Based on the results, the more resolution the scalar field is provided to the robots, the more coverage they can achieve.

We also used RL to improve the LCN performance in the presence of scalar fields. We compared four approaches, either hard-coded or reinforcement learning using Low and High-Resolution Scalar Fields. The obtained simulation shows that using the benefits of reinforcement learning to find the best policy to maximize the coverage area is preferable to the hard-coded approach in both low and high-resolution scalar fields. Moreover, using RL combined with HRSF gives the largest coverage area, as HRSF guides the robots into alignment in the same orbital, making it much easier to construct one large, connected network.

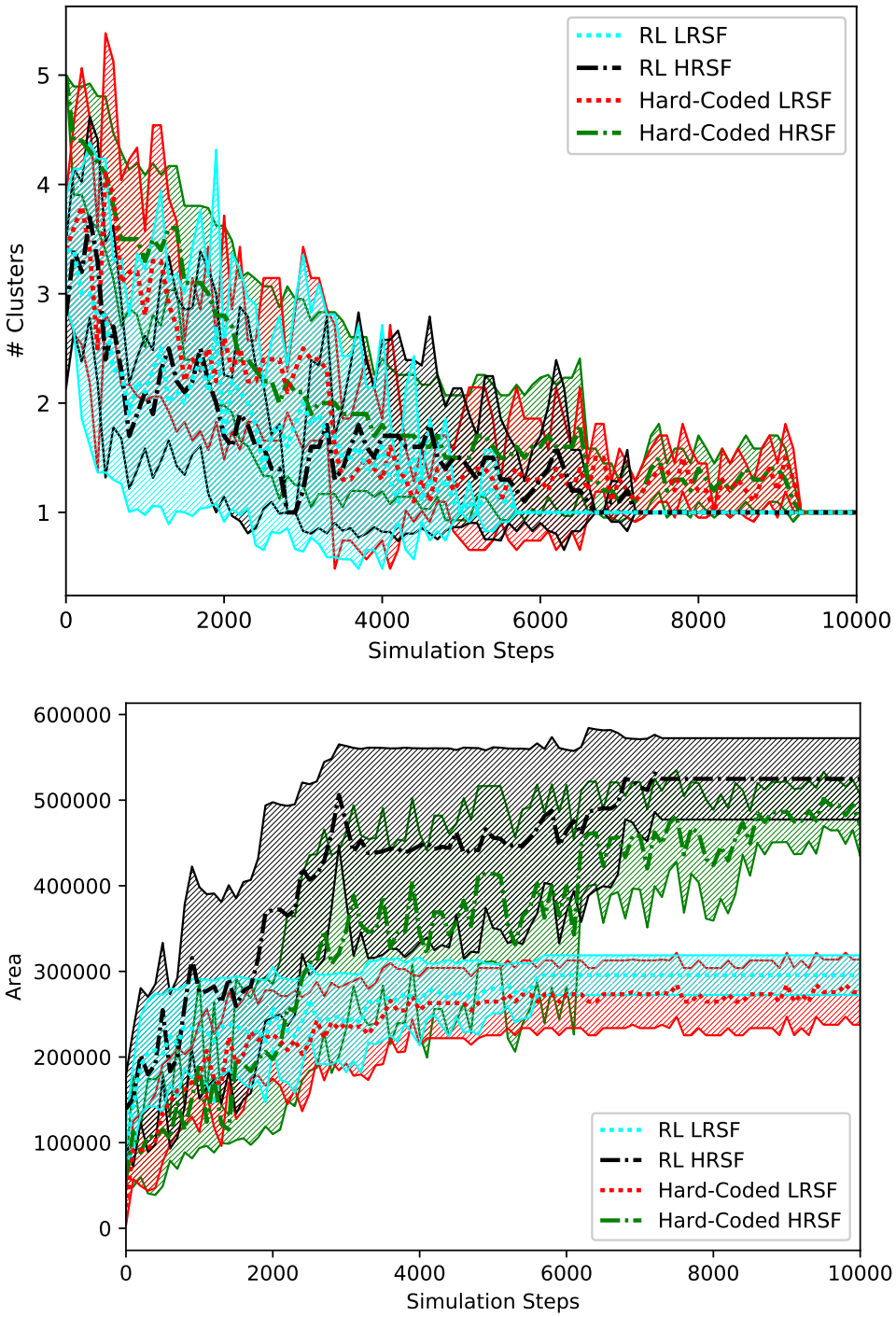


Fig. 4.12: RL Vs Hard-Coded implementations.

Chapter 5

Addressing Spatial Interference using Scalar Fields

In this chapter, we address the following question: Can access to a scalar field guide a set of robots in executing tasks that require them to visit specific positions in the environment, all while avoiding collisions with each other? We explore a scalar field inspired by the city road network and examine our scalar field design using aggregation and foraging tasks. This chapter presents parts of the work published in my submitted paper:

- Dalia S. Ibrahim, and Andrew Vardy. "Swarm in the City: Inspirations from Urban Street Networks for Swarm Robotic Guidance"

5.1 Introduction

When robots work in the same environment, there may be destructive interactions among the robots. Spatial interference between robots means that a robot's movements affect the performance or behaviour of another robot in the same space. When we add more robots to accelerate the required performance time to finish the given task, interference

between the robots could badly affect the performance because robots waste time to avoid collisions. Spatial interference can also be defined as a competition for resources; for example, collecting the same object in space, accessing a doorway at the same time or from opposite directions [41]. The robots could also block and bump each other, which might cause harm to the robots' bodies. Some applications using robots in the real world that might encounter spatial interference, like mail [42] and warehouse delivery [43] or assisted operator wheelchairs [44, 45]. There are several ways to help the robots reduce their spatial interference. For example using extra sensors like cameras and LIDAR (Light Detection and Ranging) to detect obstacles, either robots or objects, so they can adjust their movements to avoid a collision [46, 47]. Also, robots could have force feedback, called aggressive behaviour, so they are equipped with force sensors to detect and respond to physical contact with their surroundings [44]. In addition, machine learning algorithms could be used to analyze sensory data and adjust the robot's behaviour based on the patterns in these data [48, 49].

Another way is using path planning algorithms [50], which means finding the optimal feasible paths and allocating paths to each robot, considering the robots' tasks and overall mission. Some aspects of path planning should be handled, such as collision avoidance [51], maximizing the overall system efficiency, and cooperation among the robots so they can work together to accomplish the tasks, and scalability to accommodate an increasing number of robots.

As shown in Fig. 5.1, artificial potential fields (APF) and sampling-based are considered classical approaches in path planning algorithms. In APF, the robots are attracted to the goal and avoid obstacles by following the designed path from the attractive and repulsive fields. In sampling-based methods, instead of searching for the optimal path in the entire configuration space, these methods randomly sample points and construct paths through the sampled points [52]. An example of the sampling-based method is

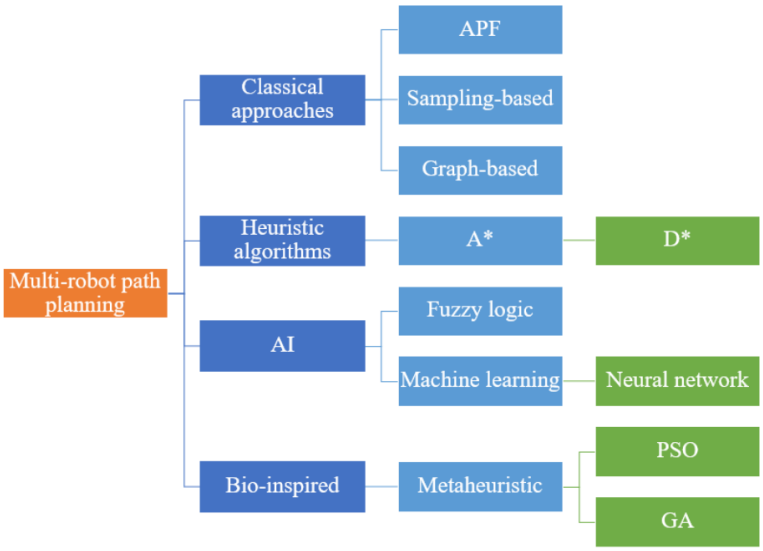


Fig. 5.1: Classification of path planning algorithms for mobile robots [52].

Rapidly-exploring Random Trees (RRTs) [53]. RRT builds a tree from the start position and incrementally grows towards the goal point.

Our work combines a sampling-based algorithm (RRT*) with the artificial potential field. We design the scalar field from the selected optimal paths by RRT*. We provide the scalar field to the robots as a projected coloured scalar field that fits the robot's capabilities in that it can sense the colour of the working environment.

In real life, designing roads in the city follows some safety evaluation to be sure the cars can be safely driven through the town. Road evaluation is divided into two parts: the first one is related to engineering design, and the second evaluation is the behavior of the drivers [54], as shown in Fig 5.2. To mimic this strategy for a swarm of robots, we design roads for robots and control the robots' behaviour by providing them with a Road-Following controller to traverse the environment.

In this chapter, we study two collective behaviours: aggregation and foraging. We chose these particular tasks because the principles learned from these behaviours could be applied to various multi-agent system applications, including exploration [55–58], surveillance [59–61], transport objects [62–65], information sharing [66–69], and search

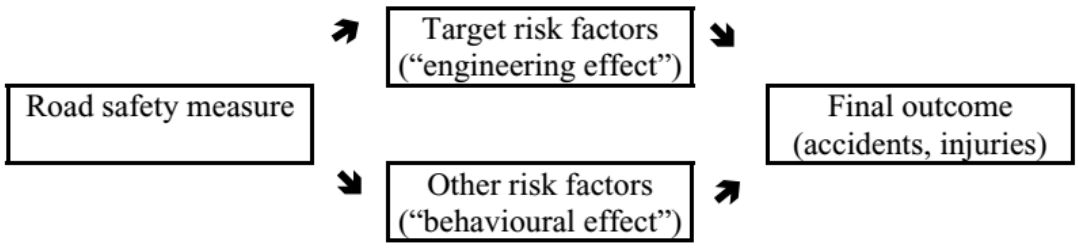


Fig. 5.2: Generalized causal chain model for how road safety is measured [54].

and rescue missions [70–74]. In addition, the foraging task shows how the multi-agent system can collectively retrieve shared accessed objects within the environment and enhance the overall system performance [75–77].

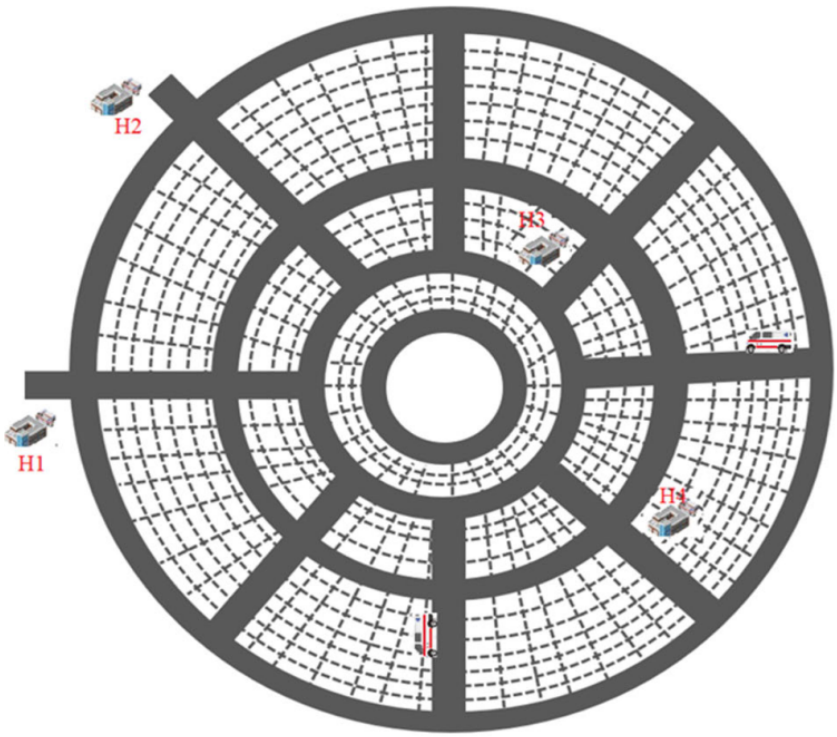
5.2 Designing Streets

5.2.1 Urban Street Design for Cities

Urban street design is the process of organizing and planning the layout of streets within the city and shaping the overall functionality of accessing different places in cities [78]. Effective urban street design enhances connectivity and makes it easier for people to move around the city. Street design can foster social interactions, encourage active lifestyles, and contribute to a sense of community [79]. There are several street layouts, such as circular, grid, crescent, and star-shaped designs [80].

The circular street layouts follow the curve pattern, do not have sharp angles, and allow for smoother turns and navigation [81]. The cities that follow the circular design have a central focal point, such as a central park or a great building [82]. For example, in Saudi Arabia, there is an "Al Masjid an Nabawi" in the city of Madinah; this mosque is the focal point in the city, as shown in Fig.5.3 in that all streets in a circular shape around it.

The grid street layout is characterized by a network of streets that create a rectan-



(a) Virtual map of Madinah city [82]

(b) A map of Madinah city [83].

Fig. 5.3: Circular street layout.

Fig. 5.4: Street networks for Manhattan and Boston [84].

gular or square pattern in urban areas, and all streets intersect at right angles. The advantage of this design is the ability to divide the city into easily organized blocks and predictable direct routes for vehicles. Examples of grid street designs include the Manhattan grid in New York city, the Boston grid system as shown in Fig. 5.4 and the Chicago grid system shown in Fig 5.5.

5.2.2 Street Design for Swarm of Robots

In this chapter, we design streets for robots to help them move around the environment and accomplish their task. We designed our swarm city layout inspired by the grid layout of Chicago because it is well-organized and has predictable routes. Fig. 5.6 is a sketch to describe our layout design. We assume the blue spot is the desired place the robots will visit frequently; it could be an aggregation area, charging, communication, or collection station. The diagonal line (direct street) connects the environment's furthest

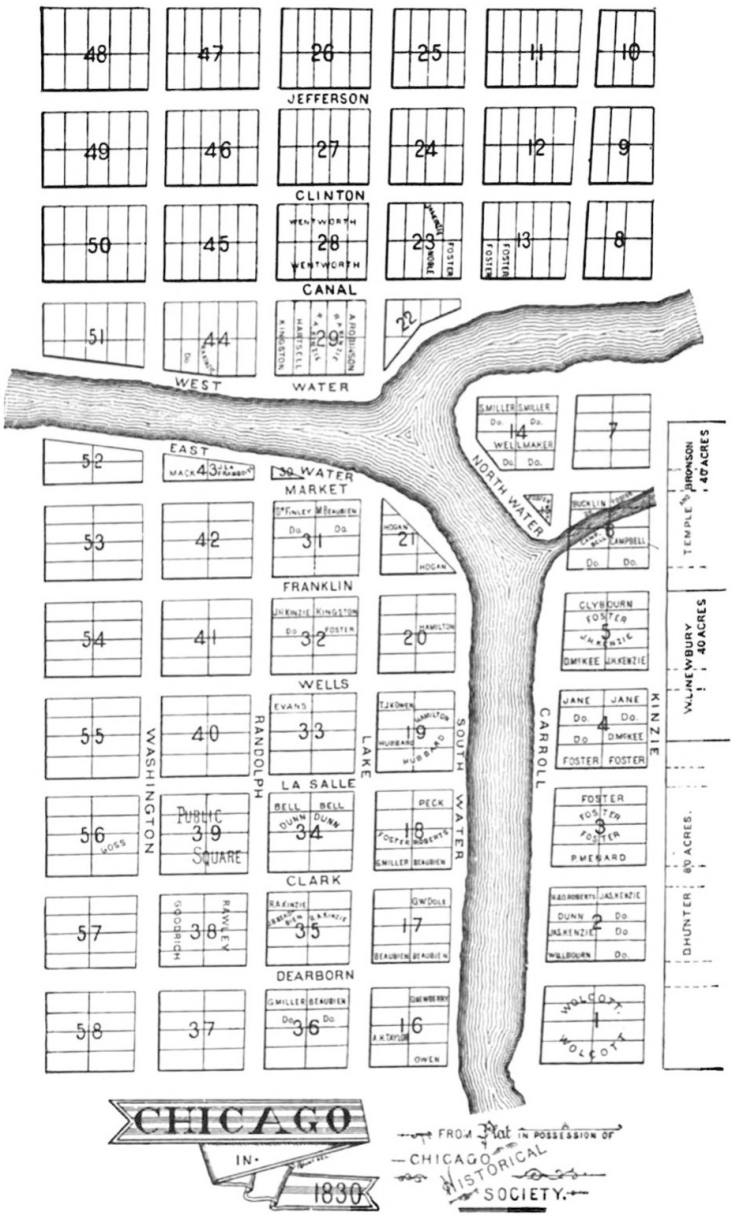


Fig. 5.5: A map of Chicago in 1830 [85].

point, intersects with all streets, and ends at the desired area. Then, we divided the environment into blocks of approximately equal size using horizontal streets (streets A - H). These blocks represent the position where the robot could be placed, as sketched in block 6. By using this street layout, the robots can reach the desired station if it is placed on any street or the direct one. Also, if the robot is placed in any position inside the block, it can use simple movement to find and follow the streets. For example, we represent the robot with the red x. If it moves left, right or straight, it finds the desired area, street A or direct street, respectively.

We will refer to this diagonal line with a highway as a metaphor. It is the fastest way for robots to reach the desired area. We call horizontal streets main roads as a metaphor; they divide the environment into main parts and are connected to the highway. Our road design should be task-oriented, meaning the roads are designed based on the robots' required mission. For example, in the foraging task, the robots should have a return path to continue searching for objects after their delivery to the collection area. However, in the sorting task, the highway should have a roundabout so the robot delivers the carried item to the desired location.

5.2.3 Designing Roads

In our design of roads for robots, all roads will be one-way to improve the overall efficiency of traffic flow and reduce the conflict points compared it two-way streets in which the vehicles may cross paths [86]. Also, one-way can have high road capacity, work well in heavily congested areas [87], and a streamlined flow of vehicles in one direction is more efficient use of road space, has safer turning movement and reduces collisions [88–90]. We assume our robots do not know their orientation, so our one-way roads will be defined by two colours: white and green. Robots will keep the green colour on their right to help them figure out the direction of the goal.

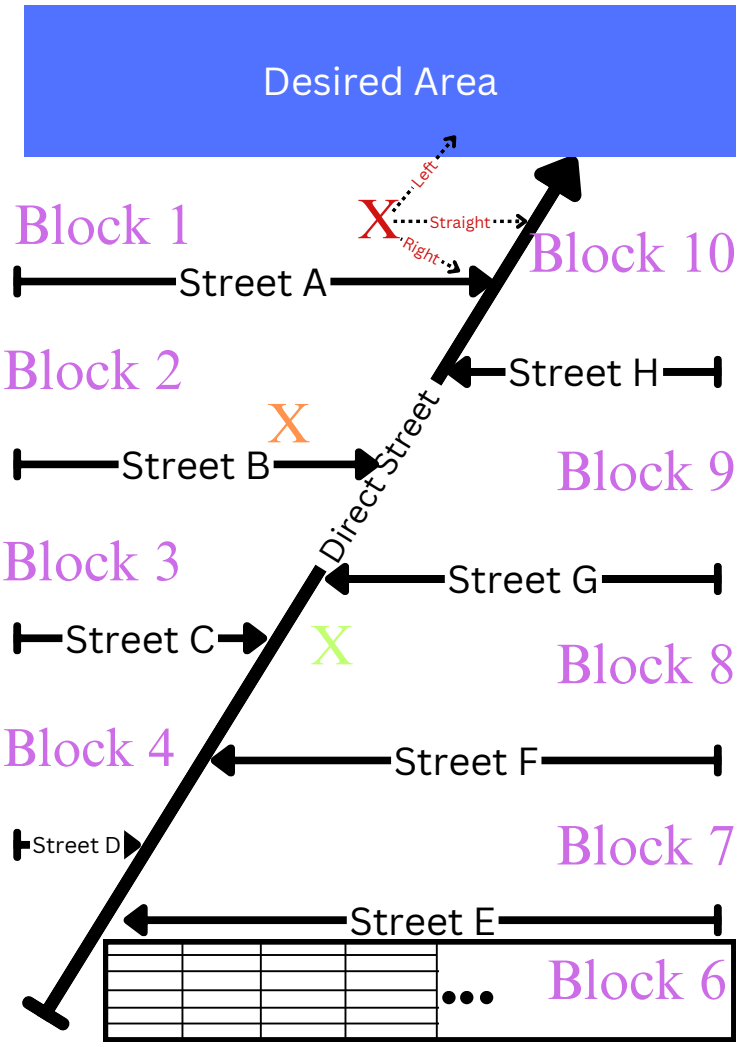


Fig. 5.6: Swarm city using grid layout.

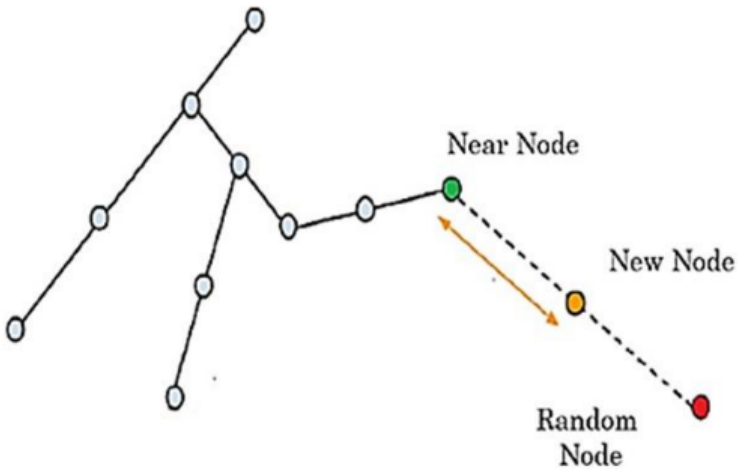


Fig. 5.7: Tree expansion process in RRT* algorithm [92].

In our work, we design the roads using a sampling-based path planning algorithm, Rapidly-exploring Random Trees (RRT*) because it has a low computational cost and is probabilistic complete, which means if there is a path, it will find it [91]. RRT* randomly samples the free configuration space by getting the optimal collision-free path between the initial and goal points. This path is organized in a tree structure; the tree's head is the starting point, and the tree gradually expands and improves with iterations. In each iteration, we select a random point. If it belongs to an obstacle-free region, we search in the tree for the nearest point to that random point. Based on the step size, the local planner connects the random point to the nearest point, or the local planner returns a new node to the nearest point, as shown in Fig. 5.7. Also, RRT* finds the least cost parent and then rewires the tree to find the optimal path [91–94], as shown in Fig.5.8.

5.2.3.1 Our Road Design using RRT*:

We design the highway by connecting the goal to the furthest point of the environment using RRT*, as shown in Fig. 5.9(a). The road connects that furthest point directly to the blue goal area so the robots can reach the goal faster if it is placed on this highway.

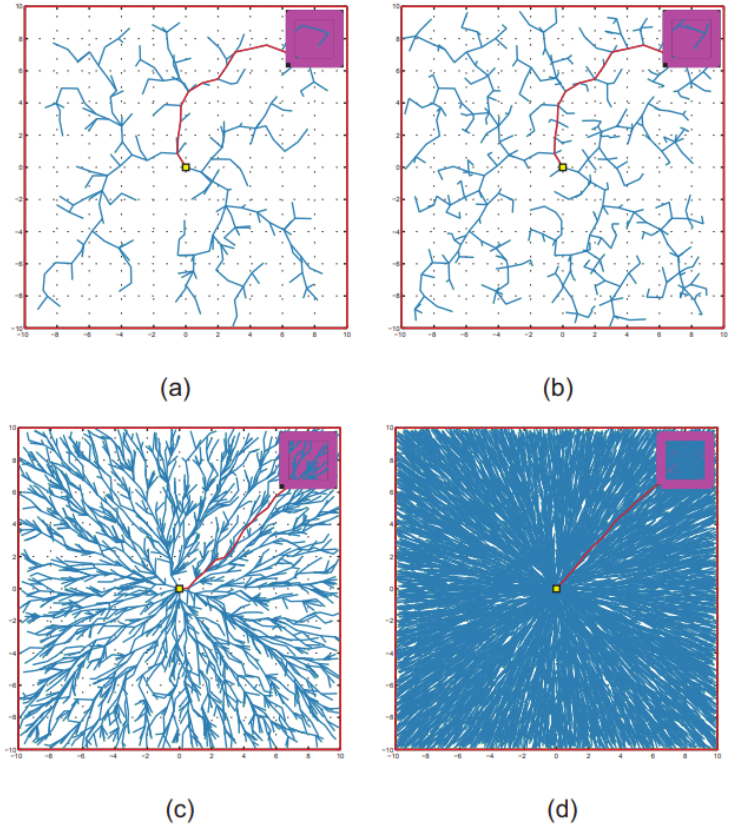


Fig. 5.8: The blue paths show the edges constructed by RRT*, starting from the yellow square to reach the goal region in the upper left (magenta colour). The tree snapshots are presented in different iterations: (a)-(d) have 250, 500, 2500, and 10000 vertices, respectively. The optimal path is highlighted with a thick red colour. [94]

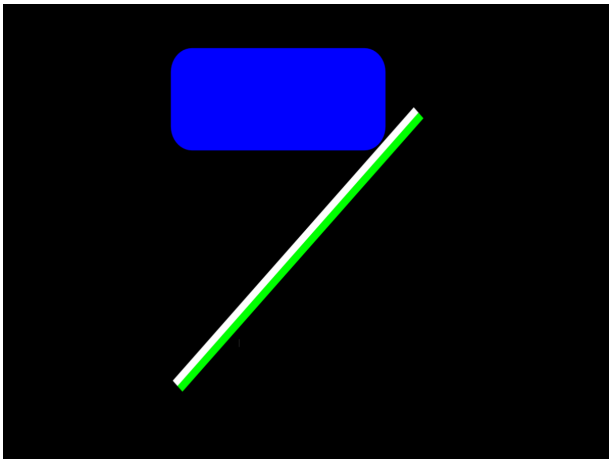
Also, we design main roads using RRT*, starting from both sides of the highway to the working environment side borders. The main roads are plotted using RRT* and interpolated to avoid sharp curves. To connect main roads with the highway, we use Bezier curves to act as ramps to enter the highway as shown in Fig. 5.9(c). A task like foraging requires the robots to return to the environment after reaching the goal to continue working; we added a return path as showed in Fig. 5.9(d), and the robots drive through this path to enter the environment again. Also, this return path will be used in the aggregation task if the robots get nudged out by other robots. Using this design, the robots follow any road, and eventually, these roads will lead them to the highway, and they can reach their destination quickly.

5.3 Software implementation using city road network

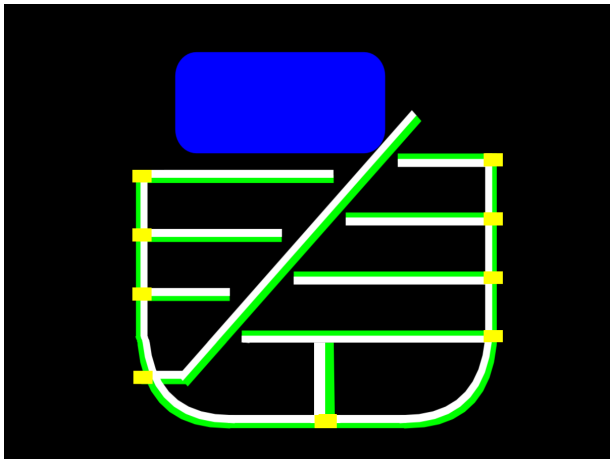
This section presents solutions for two collective tasks: aggregation and foraging in the presence of a city road network. We compare our results vs random walks, which act as a simple baseline benchmark for evaluating the performance of our controller [95]. The robots' positions and orientations are drawn from a uniform distribution, and we present the performance with a differing numbers of robots. In the foraging task, we show the results for two placements of the pucks: in the center and the bottom-right corner of the environment with a differing numbers of robots.

5.3.1 Aggregation task

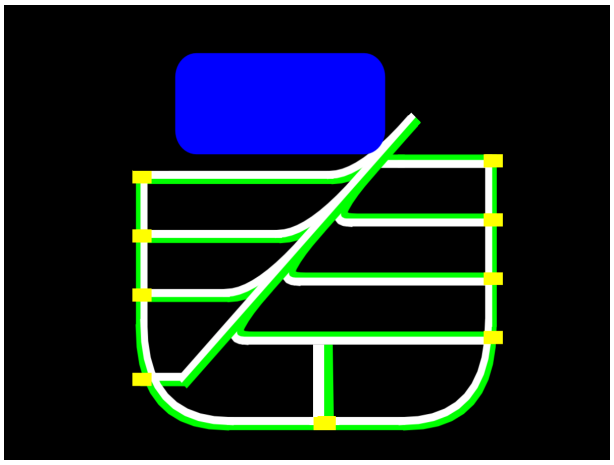
The aggregation of robots can be achieved through different strategies, including self-organizing approaches [96–99] and goal-prespecified (cue-based) methods [100–103]. In this work, we provided the prespecified area (coloured blue) for the robots to aggregate



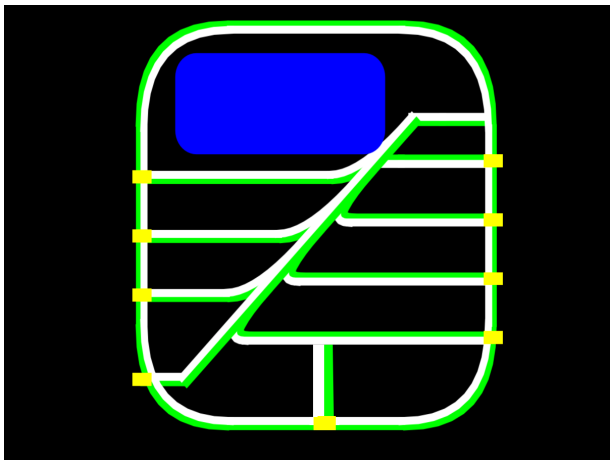
(a) Highway road and the blue destination.



(b) Adding the main roads on both sides of the highway and yellow turning points.



(c) Using Bezier curves to connect the main roads with the highway.



(d) The return path is added.

Fig. 5.9: The city road network is designed to guide the collective execution of tasks for a swarm of robots.

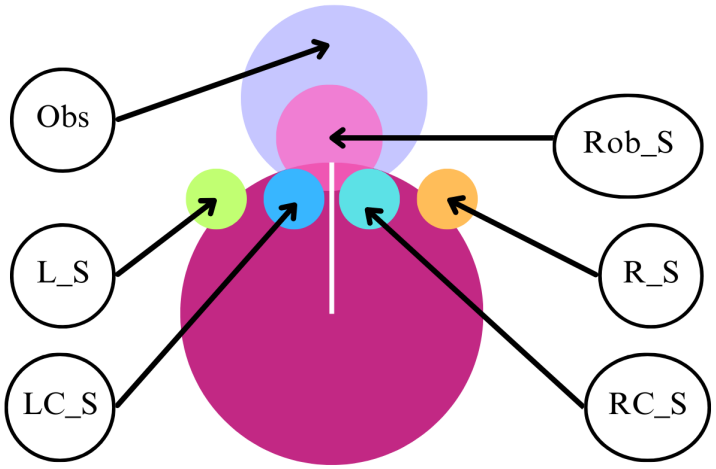


Fig. 5.10: The sensors used in the aggregation task. Left sensor (L_S), left center sensor (LC_S), right center sensor (RC_S) and right sensor (R_S) are colour sensors that detect ground colours. Obs is an obstacle sensor to detect walls. Rob_S indicates that there is a front robot ahead or not.

there and designed the roads to lead to this area. The robots are equipped with four colour sensors to detect the blue aggregation area, as shown in Fig. 5.10, and the robots can detect the front obstacles, either walls or other robots. We present our controller to a swarm of robots using our design for the swarm city road network and call it the Road-Following Controller. Then we use the Random-Walk controller as a benchmark for performance comparison.

5.3.1.1 Aggregation using city road network design and Road-Following controller:

Using the road network makes it easier for the robots to reach the aggregation area by following main roads which lead them to the highway, and at its end is the desired location to aggregate. As we mentioned, all roads are coloured with two colours: green and white. The robot can know the direction of this road by keeping the green line all the time on its right side. If it does not sense the green at any time, it rotates until the right center sensor (RC_S) detects the green, and the left center sensor (LC_S) sees

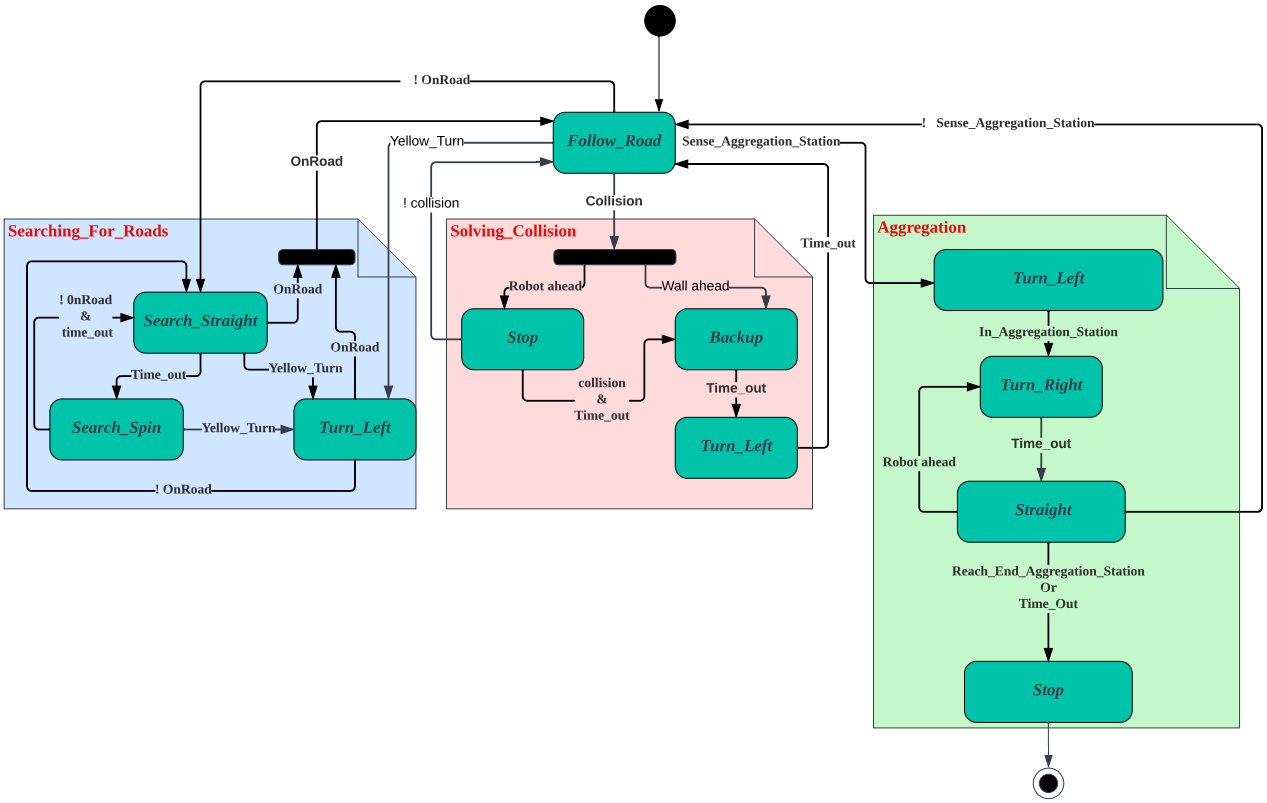


Fig. 5.11: State diagram of robot aggregation using the Road-Following controller. The filled black circle points to the initial state. The final state is denoted by a circle with a dot inside. The short heavy bar represents the fork if the bar has one or more arrows from the bar to states. Also, It is used as a join node if one or more states point to the bar.

the white colour.

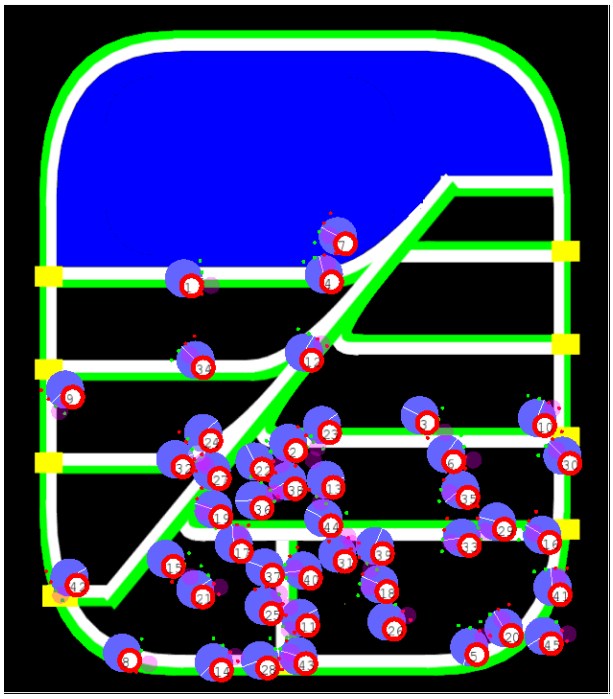
We implement our control structure using a finite state machine because it is commonly used [104–106] due to its readabilty and modularity [107]. Also, we can decompose any complex behaviour into simple behavoiurs represented by different states a robot can be in and the transitions between these states based on its sensory inputs and internal logic.

Fig. 5.11 presents the state machine for aggregation task using the Road-Following controller. The controller starts with the *Follow_Road* state in which the robots detect the direction of the road and move over it. If the robot detects a yellow turn, it turns

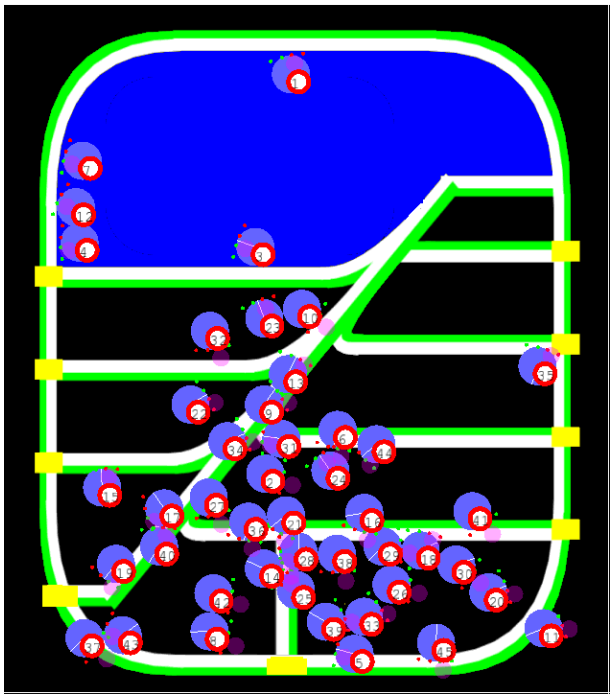
left to enter the main roads. If the robot is off the road, we assume it is between roads; the robot starts to search for the road by driving straight for some random time and then spinning and repeating that behaviour until it finds any route and follows it. At any time, if the robot encounters any obstacles, either walls or other robots, it attempts to avoid them. If there is a robot ahead, the robot stops for a random time to give that robot a chance to move. After that, if the collision is resolved, it starts to follow its road again, but if the collision is still there, it starts to move backwards and turn left to avoid that robot. In the case of the facing a wall, the robot moves backward and then turns left. Once the robot approaches the aggregation area, it turns left to enter it. The robot turns right with a random angle so that the robots drive straight through different directions in the aggregation area until they reach the edge of the aggregation area and stop there to leave room for the following robots. If there is any collision in the aggregation area, the robot solves by turning right and moving straight. If the collision is not solved, or it reaches the edge of the aggregation area, it stops. At any moment, if the robot is nudged out, it follows the return path and enters the aggregation area again.

The simulation of the following experiments was implemented in C++ using Cwaggle *CWaggle*. It is an open-source software simulator for robotics for circular-based physics agents. CWaggle has a very high update rate and allows both static and dynamic circle-circle and circle-line collision resolution.

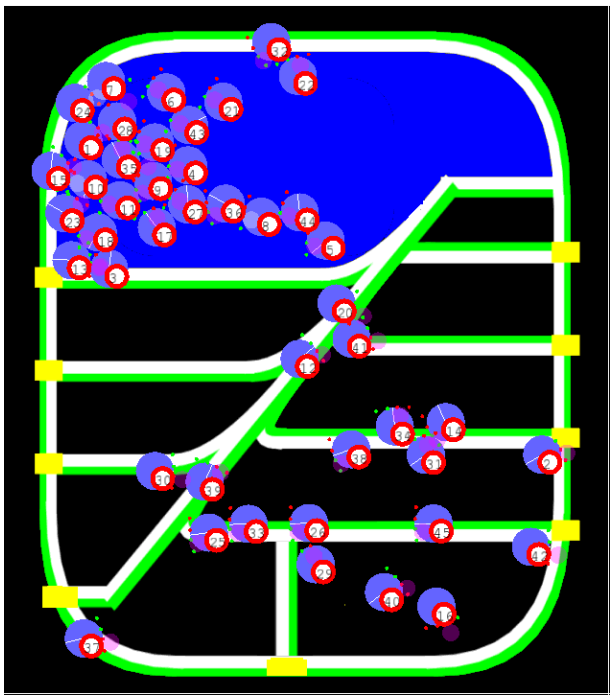
Fig. 5.12, shows screenshots of Cwaggle simulator for the aggregation task using our city road network design and the Road-Following controller. The robots' positions and orientations are drawn from a uniform distribution. At the beginning of the simulation, 45 robots are deployed and searching for the nearest road to follow. At $t = 1800$, robot #7 is the first to reach the aggregation area, then robot #4. Both of them drove with different angles and moved straight to the edge of the aggregation area, so they were



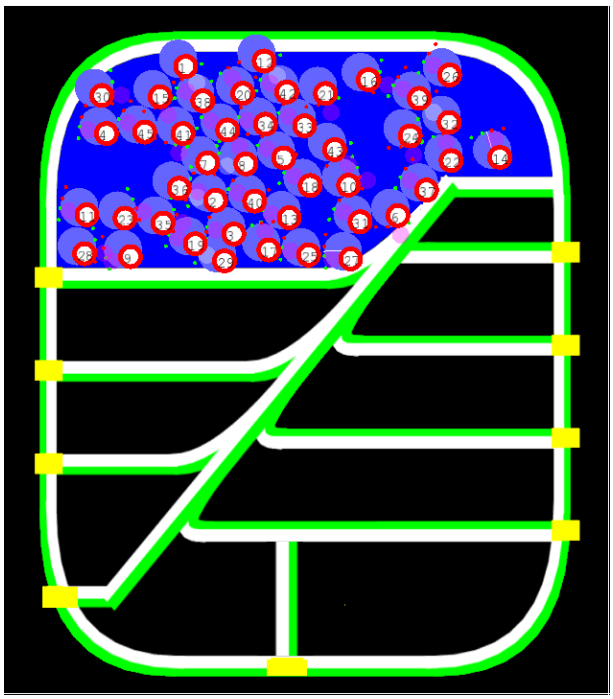
(a) Screenshots of CWaggle simulator for 45 robots at $t = 1800$.



(b) Screenshots for robots at $t = 3000$, some robots stop moving when they reach the end of the aggregation area (blue area) such as robots #7, #12, #7 and #1.



(c) $t = 5400$, some robots reach the aggregation area.



(d) t= 34500 all robots reach the aggregation area.

Fig. 5.12: Screenshots of 45 robots aggregating using city road network and Road-Following controller.

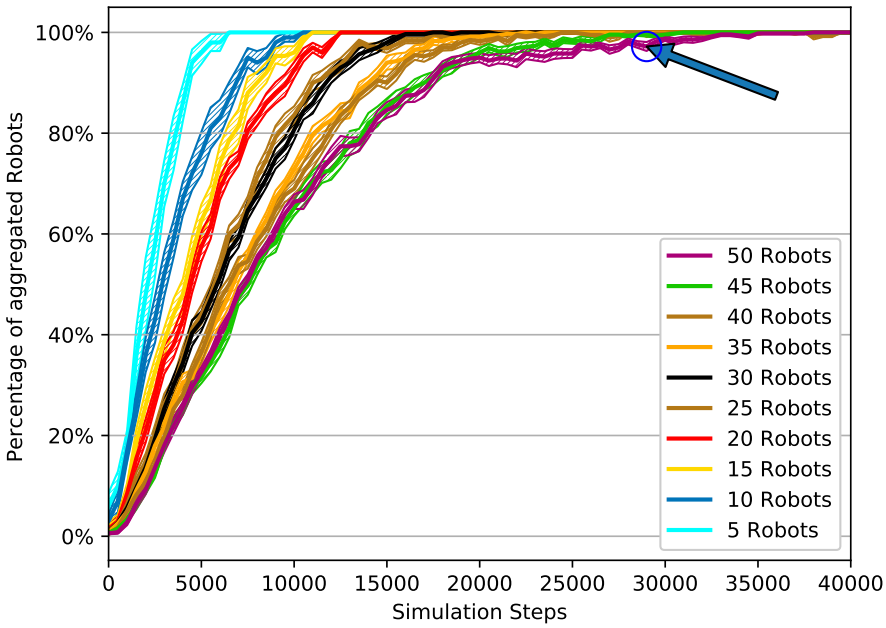


Fig. 5.13: Aggregation using city road network and Road-Following controller.

placed in different spots as shown at $t = 3000$ and the same happened for robot #1 and robot #12. At $t = 5400$, 26 robots reached the aggregation area, and the others were on their way. Robots #3 and #13 are about to be nudged out, so they will take the return path and follow the highway to enter the goal area again. Finally, all robots aggregated successfully at $t = 34500$.

The experiments are performed over ten trials for different numbers of robots; the results are generated from *CWaggle* simulator.

We studied the performance of the Road-Following controller for aggregation task for different numbers of robots, n, chosen from the set $\{5, 10, 15, 20, 25, 30, 40, 45, 50\}$. In Fig. 5.13, the thick line shows the average performance of certain robots, and the shaded areas present the confidence interval for a 90% confidence level. As shown, at the end of the simulation for all trials, all robots fit smoothly in the aggregation area. When the total number of robots n is smaller, such as $n \in \{5, 10, 15, 20\}$, the robots aggregate faster than the larger number of robots because of the number of collisions

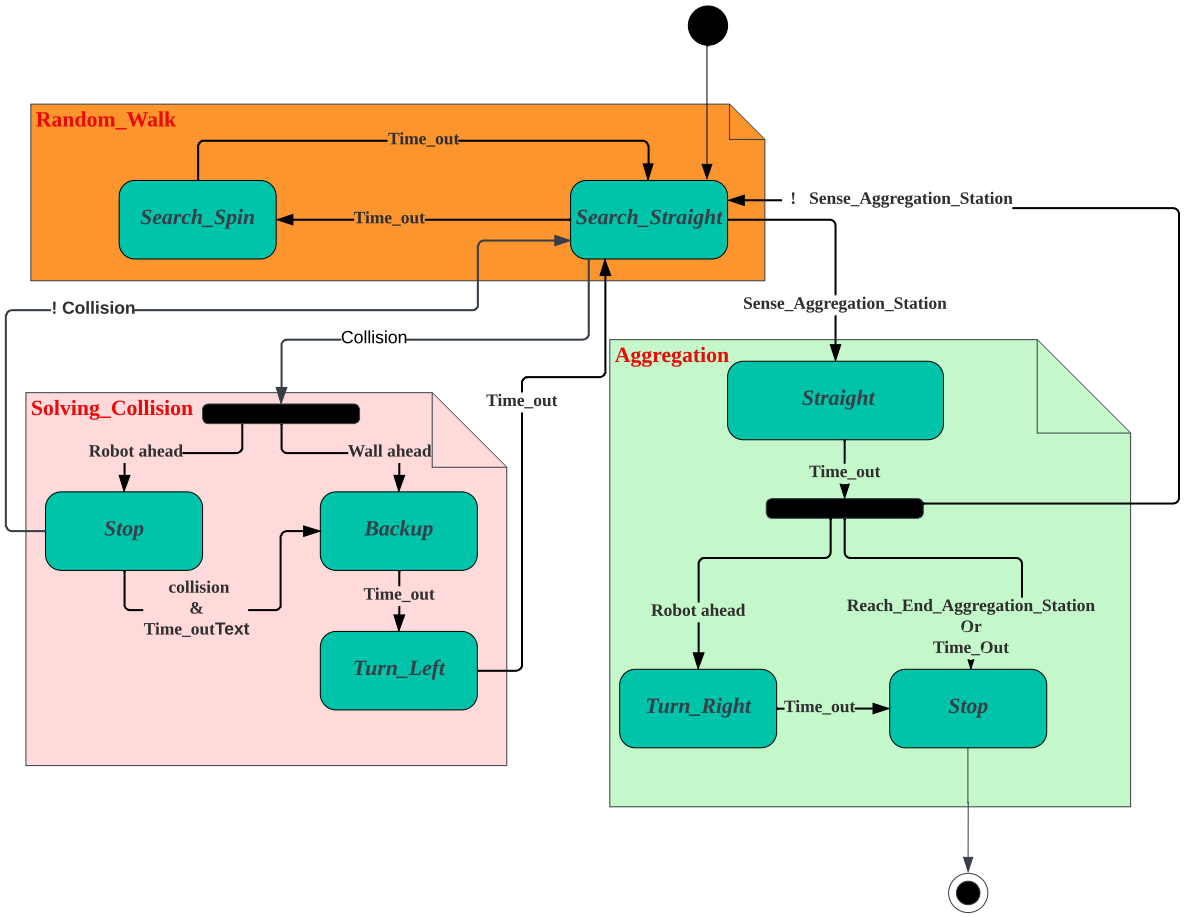


Fig. 5.14: State diagram of aggregation using Random-Walk controller.

either outside or inside the aggregation area. When we increase the number of robots, and the robots try to find an empty spot in the aggregation, this can cause some robots to be nudged out, as shown by the blue arrow with 50 robots after *20000* time steps, at which time most of the robots have reached the aggregation area.

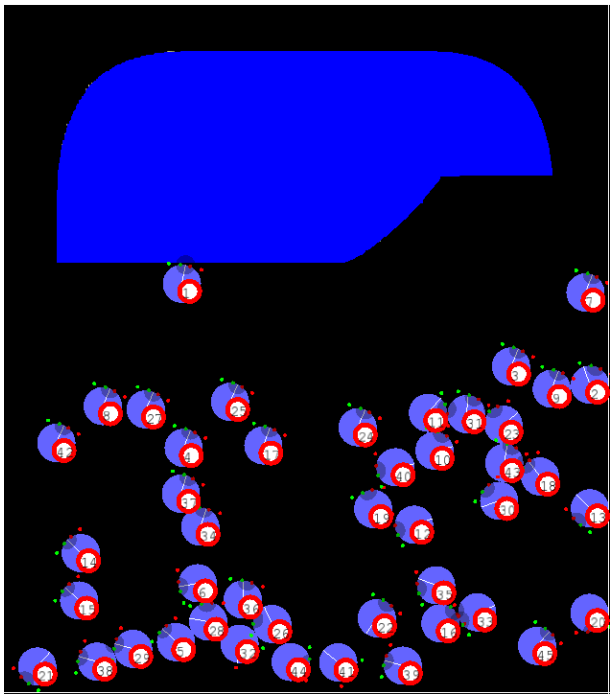
5.3.1.2 Aggregation using Random-Walk controller:

The robots are equipped with the same sensors described in Fig. 5.10; the coloured sensors are used to detect the blue aggregation area. No environmental information can

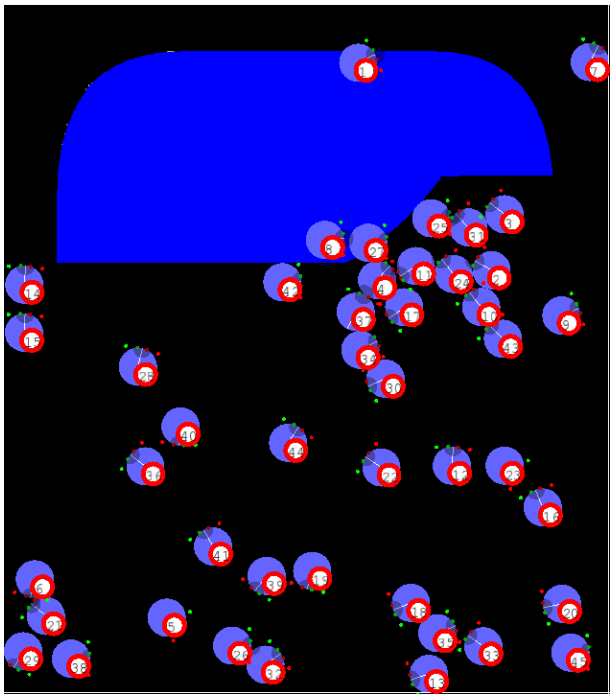
be obtained except if the robots are in the blue aggregation area and if the robots collide with other robots or walls. Random search is a basic strategy for the robots to explore the environment and find the blue aggregation area, especially for these simple robots, who depend only on their local sensing and low computational power [108, 109]. There are commonly used methods for random walks, such as Brownian Motion [110–112] and Lévy Flight [113–116]. Brownian motion is named after botanist Robert Brown, who observed the random motion of pollen grains suspended in water and changed their direction randomly. Brownian motion is a continuoustime process, meaning it is defined for all points in time. Additionally, it is a Markov process, implying that the future movements of the particle depend only on its current position and not on its past trajectory [117]. The motion of the particle can be modeled as a random walk, where the particle takes random steps in different directions at discrete time intervals. Lévy flight introduces long steps derived from Lévy probability distribution, unlike the usaul random walk, which has a fixed step size [114]. In our random walk controller, we use random step size derived from the uniform distribution and random direction also derived from uniform distribution. This approach works well within our state machine design.

Fig. 5.14, shows the state machine of the Random-Walk controller. The robots' positions and orientations are drawn from a uniform distribution. The robots start moving straight and then spin repeatedly. They do this random walks till they find the aggregation area. They solve any collision either with the wall or other robots in the same way as Road-Following controller. Once they approach the aggregation area, they move straight to reach the end of the aggregation area to leave spots for the coming robots.

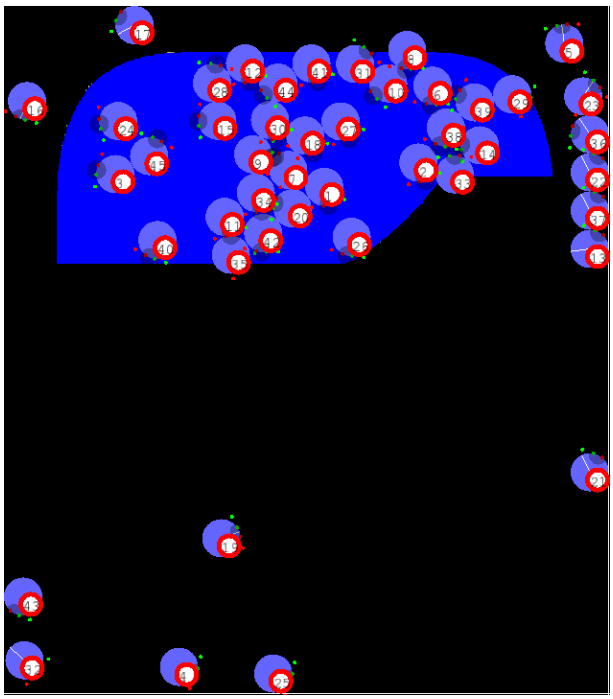
Screenshots of CWaggle simulator of the Random-Walk controller are shown in Fig. 5.15. At the beginning of the simulation, all robots are distributed randomly and start



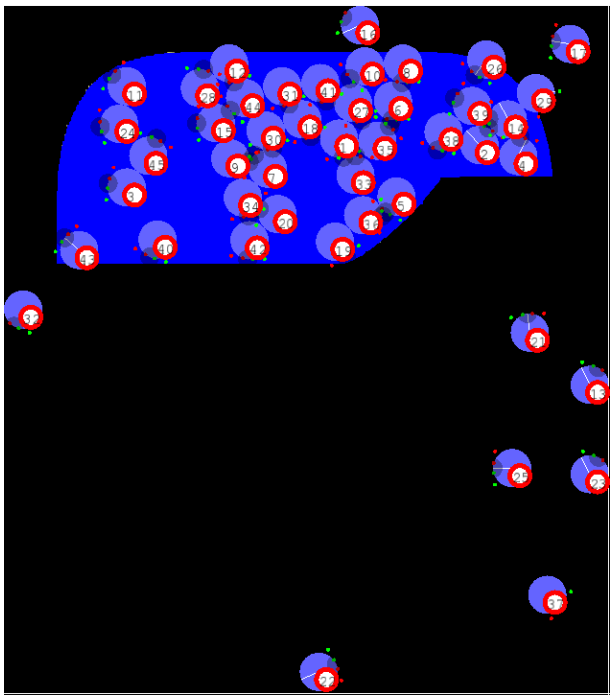
(a) $t = 600$. Robot #1 approaching the aggregation area (blue area).



(b) Screenshots for robots at $t = 1100$, Robot #1 reached the end of the aggregation area and stopped.



(c) $t = 26800$, some robots reach the aggregation area.



(d) t= 32400 Some robots are traped outside and can not reach the aggregation area.

Fig. 5.15: Screenshots CWaggle simulato of 45 robots aggregating using Random-Walk controller.

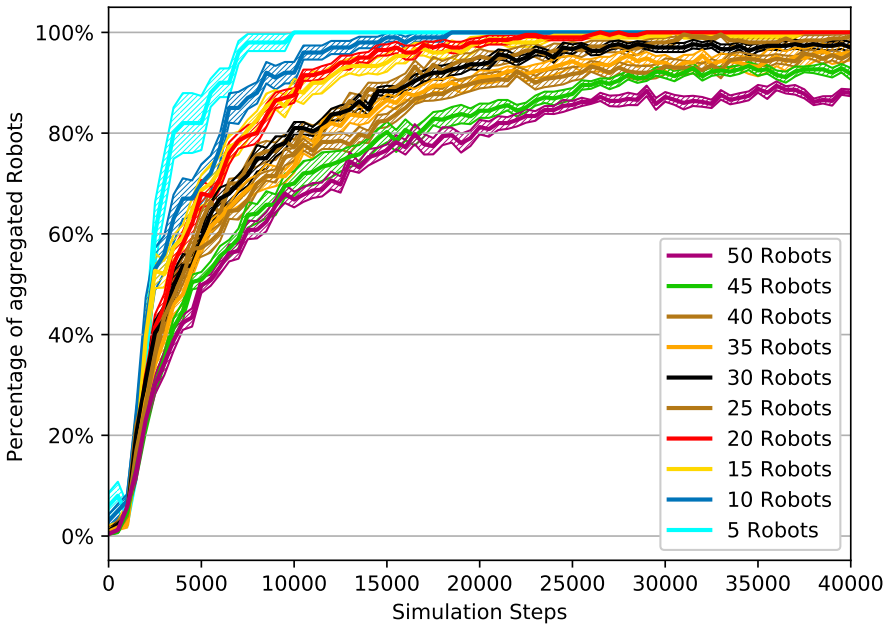


Fig. 5.16: Aggregation task using Random-Walk controller.

applying Random-Walk controller to avoid colliding with each other. At $t = 600$, robot #1 is the first to reach the aggregation area, and it keeps moving straight to the end of the aggregation area and stops there at $t = 1100$. At $t = 26800$, Robot #1 is nudged out and enters the aggregation area again with other robots. At $t = 32400$, 36 robots are in the aggregation area, some blocking other robots from entering the aggregation area like robots #5, #8, #10, #19, #29 and #43.

Fig. 5.16, shows the performance of different numbers $n \in \{5, 10, 15, 20, 25, 30, 35, 40, 45, 50\}$ of robots applying a Random-Walk controller; for each robot, the experiment is performed over ten trials, and the shaded areas are the confidence interval for 90% confidence level. All the smaller number of robots $n \in \{5, 10, 15, 20, 25\}$ are aggregated at the end of the simulation steps. On the other hand, the large number of robots $n \geq 30$ robots are achieved 90% of the total number of robots. With the increasing number of robots, the performance decreased to 83% in 50 robots because the robots entered the aggregation area from different locations. They reached the end of aggregation based

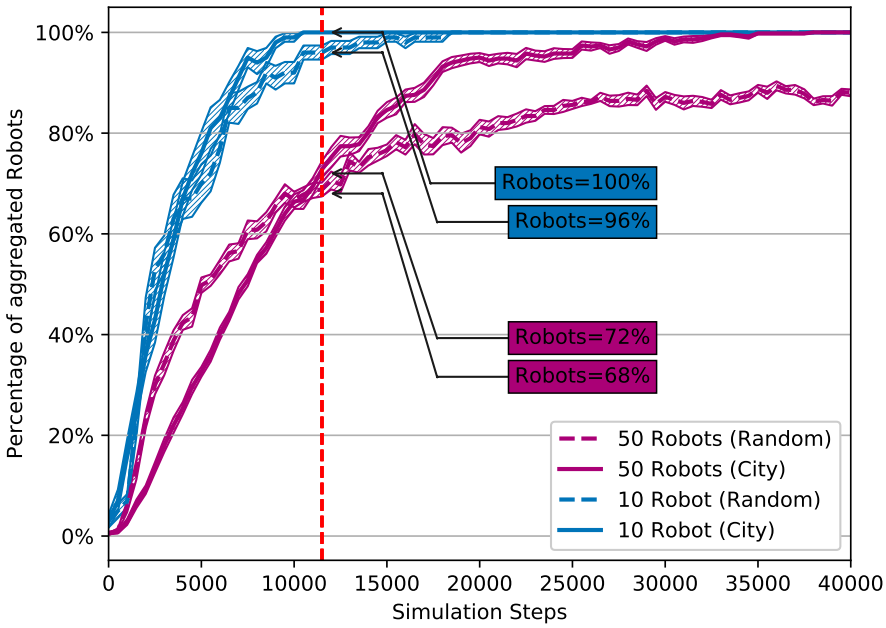


Fig. 5.17: Comparison between two controllers: Road-following and Random-Walk controllers for aggregation task.

on their entry location, which caused a collision in the aggregation area. The robots stopped around the aggregation area's boundary, which blocked others from reaching it.

5.3.1.3 Discussion

Fig. 5.17, presents the comparison between Road-following and Random-Walk controllers for 10 robots used as representatives of a small number of robot groups and 50 robots as an example of a large number of robot groups.

With a small number of robots, at the beginning of simulation before $t = 5000$, the robots in Random-Walk controller have the freedom to move in the environment, and the number of collisions is small, so they aggregate faster than Road-Following controller in which robots try to find the roads and follow them to the aggregation area. However, after $t = 5000$, the robots in the Road-Following controller find their roads and approach the aggregation area from the same entry location, organizing their entry faster than the Random-Walk controller.

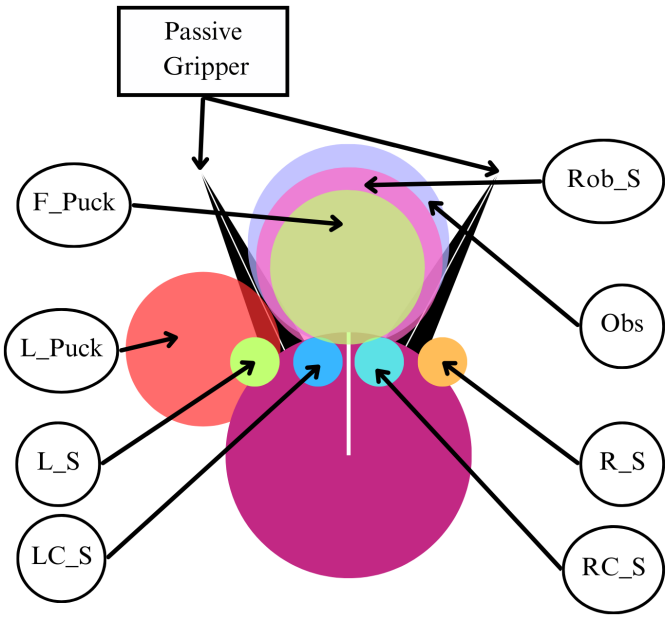


Fig. 5.18: Robot's sensors are used in foraging tasks. L_S, LC_S, RC_S and R_S are colour sensors that detect roads and collection station colours. Obs is a wall detector sensor. Rob_S indicates whether a robot is ahead. L_puck and F_Puck sensors sense the pucks on the robot's left and front, respectively.

That also happened with a larger number of robots, in our case 50 robots; the freedom of movement of the Random-Walk controller achieves good performance before $t = 10000$ compared with the Road-Following controller. However, at the end of the simulation, the robots in Road-Following are all found a place within the aggregation area. Unlike robots that applied the Random-Walk, some are blocked from entering the aggregation area.

5.3.2 Foraging task

In the foraging task, robots are required to collect pucks through the environment and deliver them to the predefined location known as a collection station. The robots have four floor-coloured sensors to detect the collection station, which is coloured blue and also can see the projected roads in the city road network environment. Robots push the pucks using their passive gripper and can see pucks on the front and left. Also, they can

detect front obstacles, either walls or other robots, as shown in Fig. 5.18. We present our Follow-Road controller using the city road network and we use the Random-Walk controller as benchmark to evaluate the performance of our controller.

5.3.2.1 Foraging using the city road network and Road-Following controller

In this implementation, we use our city road network design to help robots collect and deliver pucks to the collection station. As we mentioned earlier in Section 5.2.2, all roads use two colours, green and white; the robot keeps the green on its right and follows the roads to know the direction of the collection station. In the foraging task, robots count the yellow turns to know roughly where they are with respect to the collection station; unlike the aggregation task, robots turn left once they encounter any yellow turns.

Fig. 5.19, shows the state machine of the Road-following controller for the foraging task. In the beginning, robots are placed randomly, so they try to follow a road if it is placed on a road; otherwise, they search for the nearest road by moving straight and spinning repeatedly till they find a road. We placed the passive gripper in front of the robot, so by default, it pushes any puck in front of it. At any moment, if the robot detects if there are any left pucks and it does not push a front puck, the robot will turn to face this puck and then undo that turn to return to its road. When the robot reaches the end of the highway, it will sense the blue collection station; if the robot is pushing any pucks, it will turn left and go straight to deliver the puck, then move backward and follow the return path. However, if the robot does not have any pucks, it will take the return path, and in both cases, it will reset its yellow turn counter and increment the reference counter to know its following target location. On the robot's way, once it detects the yellow turn and its yellow counter does not equal the reference counter, it increments its counter and moves straight. However, if the robot has a puck, the priority is to deliver it, so it will take this turn and deliver it, then start over.

Besides the yellow turn helping the robot to turn left to find the highway, we also

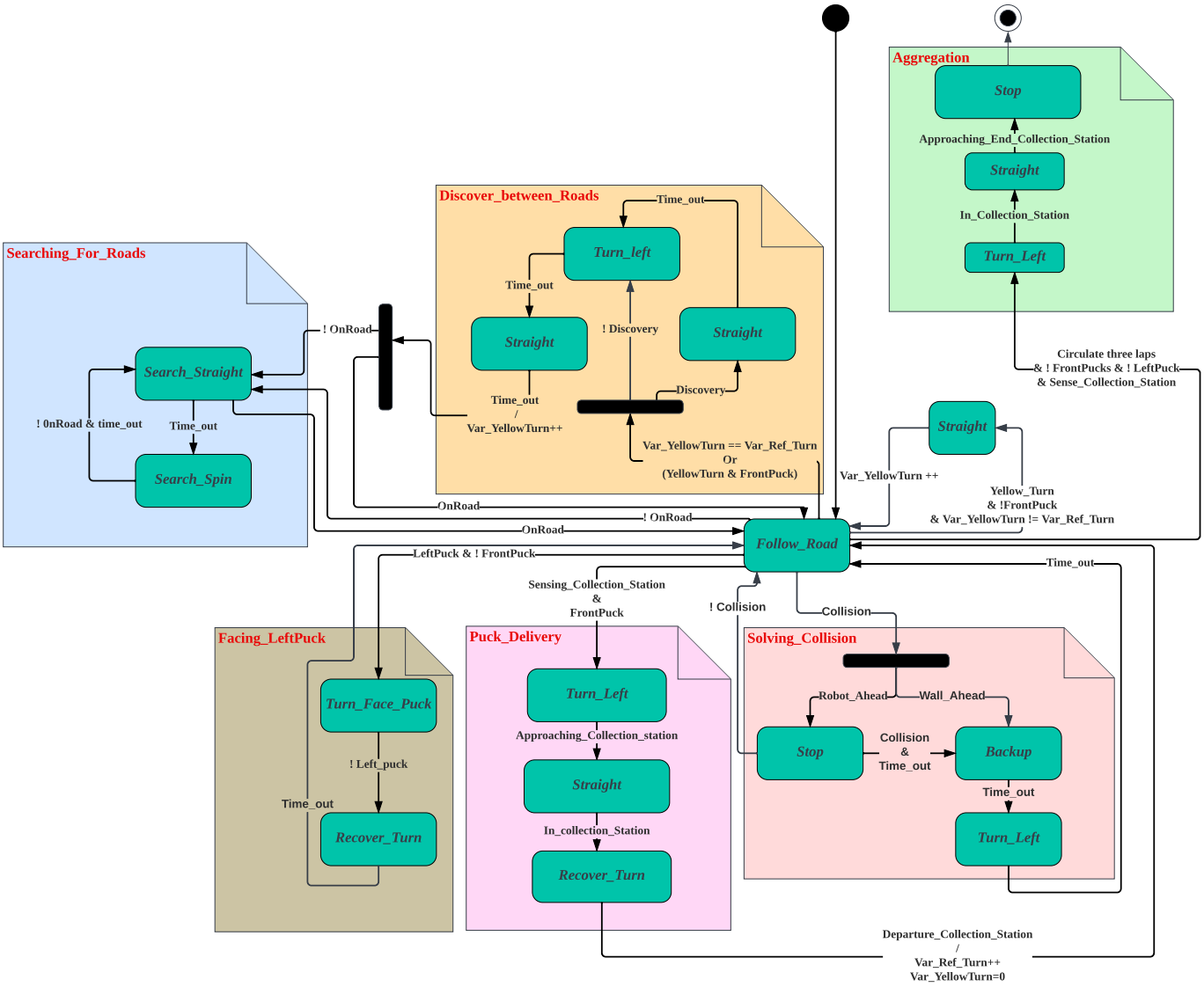
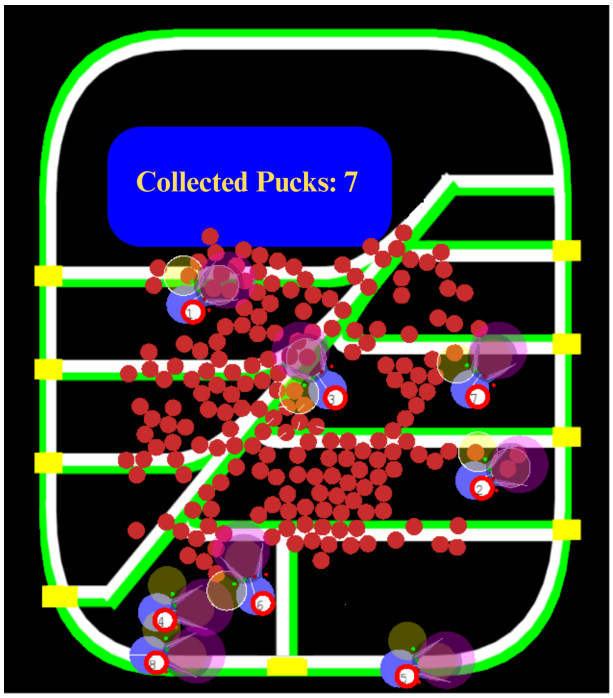


Fig. 5.19: State diagram of the pucks delivery using the city scenario controller.

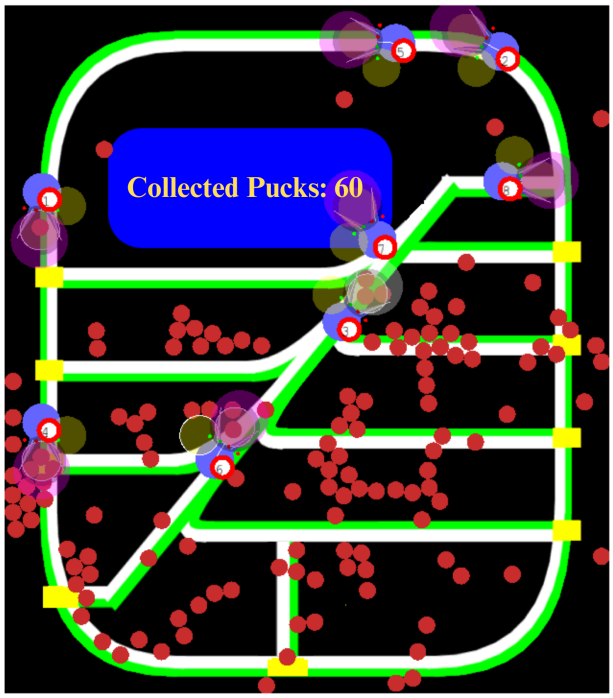
use it to discover between roads. We set a random variable drawn from a uniform distribution, and based on this value, even or odd, we set the discovery boolean variable to true or false, respectively. If the discovery variable is true, the robot moves straight, takes a left between two yellow turns, and then moves straight. The purpose of discovery between roads is that some pucks might be placed between two roads so the robots can reach these pucks. Eventually, if the robot circulates the whole environment three times and does not detect any front or left pucks, it indicates that the job is done, so the robot will reach the collection station and stop there.

We use the Cwaggle simulator, and the puck positions are drawn from a uniform distribution where $\frac{Width}{4} < x < \frac{3 \times width}{4}$, and $\frac{Hight}{4} < y < \frac{3 \times Hight}{4}$, so that the pucks will be placed around the center of the environment at the beginning of the simulation. The pucks will disappear when pushed and placed in the blue area. In Fig. 5.20(a), the robots are randomly distributed over the working area; robots #5 and #8 are placed on the road, so they follow this road, but the others move straight then spin, looking for the nearest road to follow. Fig. 5.20(b), shows that robots #1 and #4 are pushing pucks and will take the nearest yellow turn to reach the collection station. While robots #6 and #3 are moving on the highway to deliver their pucks. Robot #7 is in the collection station delivered its pucks, and is about to leave. Robots #8, #2 and #5 take the return path after reaching the collection station. Fig. 5.20(d), shows that around 70% of pucks are delivered. Also, there is a collision about to happen between robots #6 and #1; robot #6 detects there is a robot in front of it, so robot #6 will stop for a time to give robot #1 a chance to clear its way.

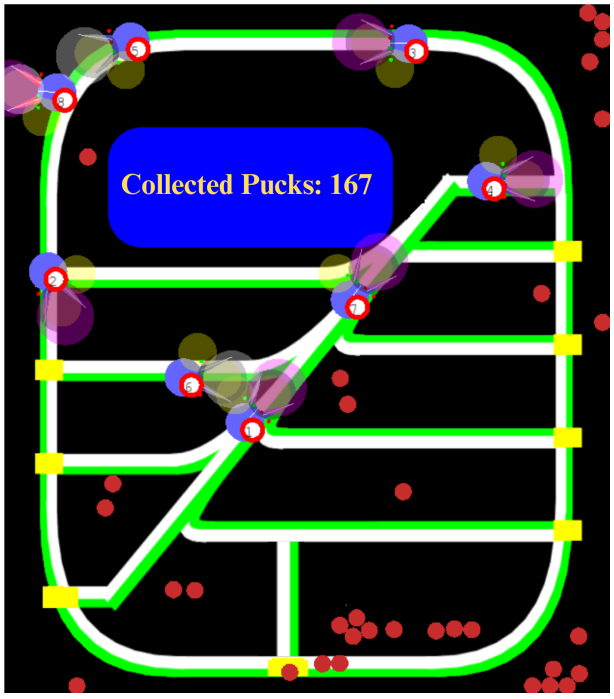
We tested the Road-Following controller for foraging tasks by placing 200 pucks in the center and bottom-right corner of the working environment. We ran ten trials with different numbers of robots $n \in \{1, 2, 4, 6, 8, 10, 12, 14\}$. In Fig. 5.21(a), and 5.22(b), the shaded area is the confidence interval for 90% confidence level. The thick lines are the



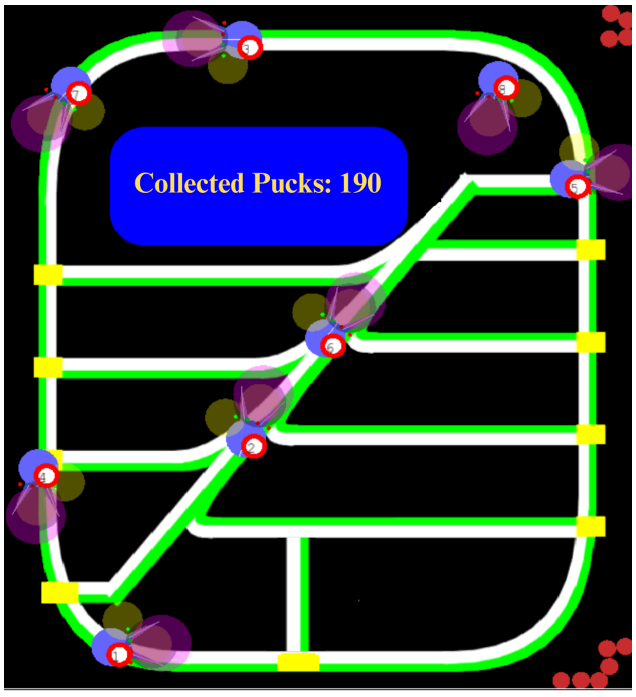
(a) $t = 100$, robots are searching for the nearest road to follow.



(b) $t = 16200$, robots are collecting pucks and deliver them to the collection station (blue area).



(c) $t = 143800$, the pucks are partially delivered.



(d) $t = 294200$, 95% of the pucks are delivered.

Fig. 5.20: Screenshots of 8 robots delivering pucks to the collection station using Road-Following controller. The pucks disappear when they hit the blue area.

average collected pucks over the simulation time steps.

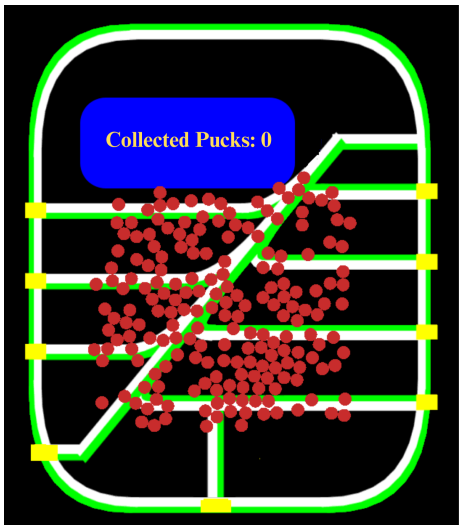
When the pucks were placed around the center (see Fig. 5.21(a)), the performance increased when we increased the number of robots. A single robot can collect over 87% of pucks. For robots $n \in \{4, 6, 8\}$, 95% of the pucks are collected. However, robots $n \in \{10, 12, 14\}$, when some of them deposit the pucks in the collection station and turn to take the return path, this movement causes collisions happened near the collection station and at the beginning of the return path. We can enhance the performance for these robots if we design multiple entries for the collection station and multiple return paths as well.

When we placed the pucks in the bottom-right corner, the robots took time to find their roads and scan the environment until they reached the pucks' location. Also, pushing these pucks long distances compared to the centred placement, consumes time. In Fig. 5.22(b), when one robot collects pucks alone, it collects 75% of the pucks, and when we increase the number of robots to 2 , they collect 90% of the pucks. For robots $n \in \{4, 6, 8\}$, 93% of the pucks are collected.

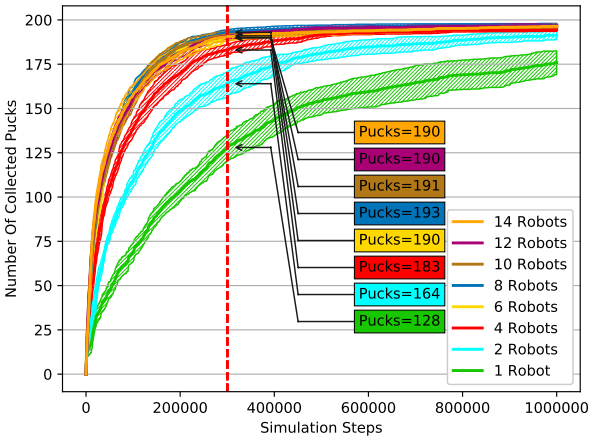
Fault tolerance in Road-following controller:

To study the fault tolerance for this system, we present two case studies of the two robots' failure during the execution and measure the overall performance of the swarm to collect the pucks and deliver them to the collection station.

The first case study is when the fault occurs on the main roads. At the beginning of simulations, all robots work together, as explained earlier, but at $t = 10000$, robot #1 and robot #2 are stopped on the main roads on both sides of the highway, as shown in Fig. 5.23. The other robots keep working on their roads and delivering pucks until they reach the main road where the robot is stopped; they consider that robot an obstacle and perform avoiding collision behaviour. For example, at $t = 300000$, robot #6 has

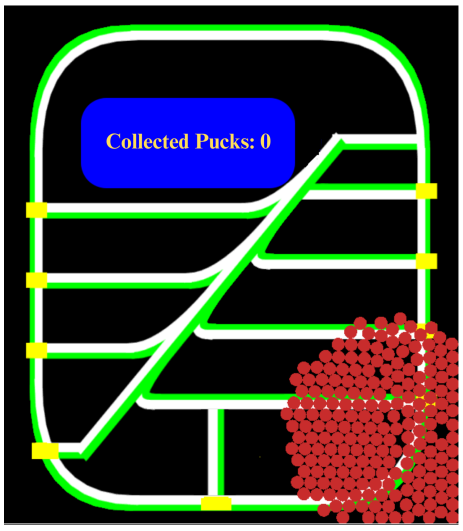


(a) Pucks start position at t= 0

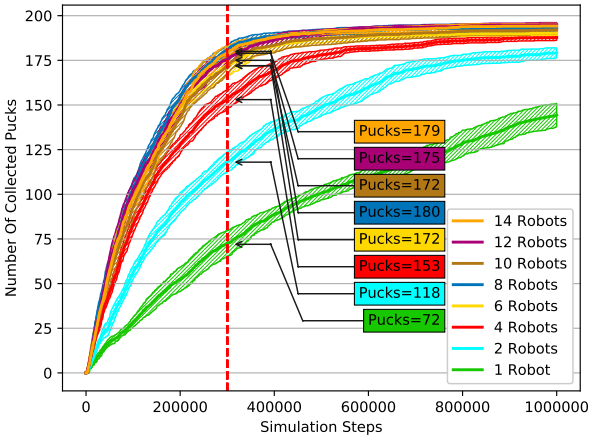


(b) Performance with different numbers of robots.

Fig. 5.21: Performance with different numbers of robots; The dashed red line shows the performance at t= 300000. Pucks are positioned in center of the environment; the **Road-Following Controller** is applied.

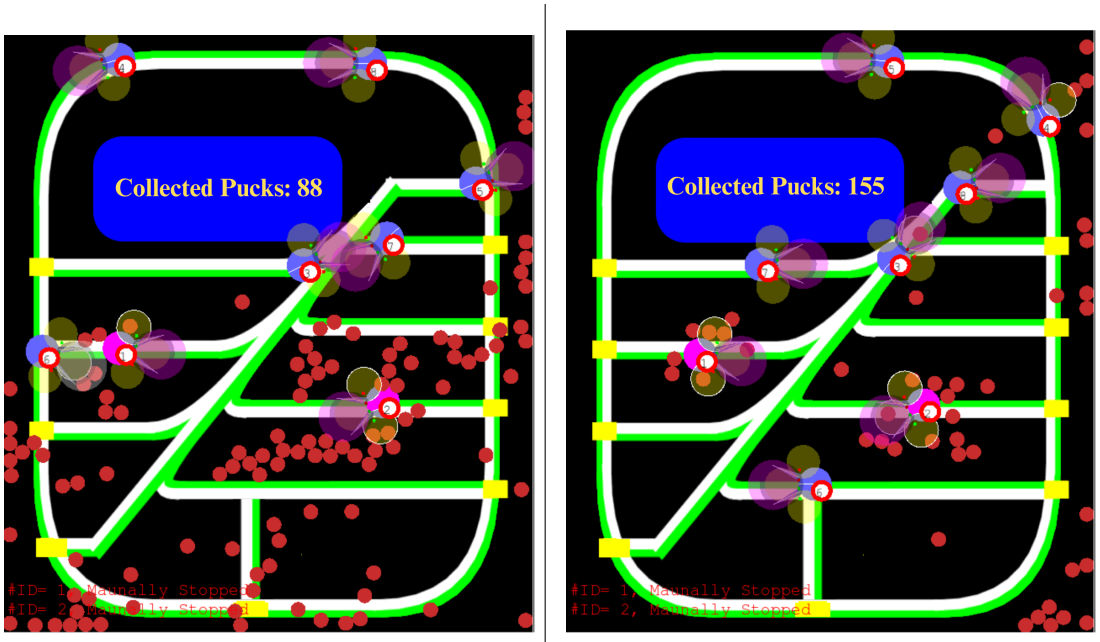


(a) Pucks start position at t= 0



(b) Performance with different numbers of robots; The dashed red line shows the performance at t= 300000.

Fig. 5.22: Pucks are positioned in the bottom-right corner of the environment; the **Road-Following Controller** is applied.

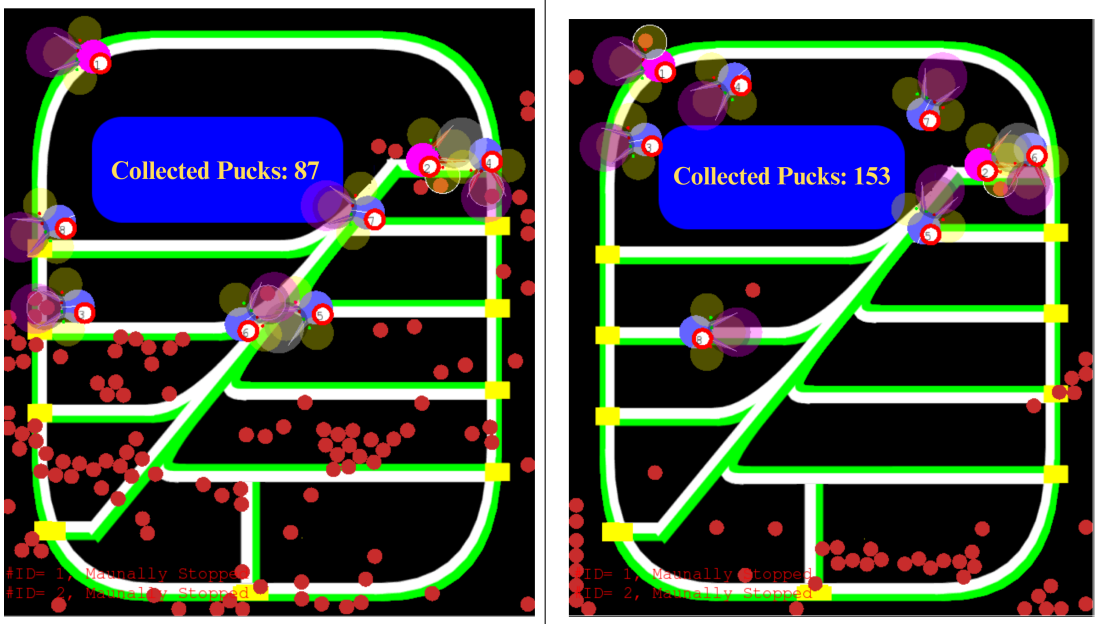


(a) $t = 300000$ (b) $t = 600000$

Fig. 5.23: Screenshots of the CWaggle simulator. Robots #1 and #2 stopped on two main roads on both sides of the highway.

robot #1 in its way, so it stops for a time and then turns left. At $t = 600000$, most pucks are collected except the very closest to that faulty robot.

The second case study is these two robots stopped in the return path right after delivering their pucks, as shown in Fig. 5.24. This is a critical problem because this is the only way for the robots to return to the environment and continue collecting pucks from following main roads. At the beginning of the simulation, all robots are operating normally, but at $t = 10000$, robot #1 and robot #2 delivered their pucks and they were triggered to stop in the return path at this time specifically for this case study. The following robots detect these robots and perform collision avoidance behaviour. For example, at $t = 300000$, robot #4 collide with robot #2, so it stops and turns left. Then it will detect it is in the wrong direction, so robot #4 will fix its direction and follow the return path until it detects robot #1, so robot #4 will turn left to avoid it and go straight and turn to find any road again. At $t = 600000$, the pucks are partially collected



(a) $t = 300000$ (b) $t = 600000$

Fig. 5.24: Screenshots of two different time steps where robot one and robot two stopped on the return path after leaving the collection station.

even when the return path is congested because of the faulty robots.

Fig. 5.25, compares the performance of a swarm with no fault, two faults on main roads, and two faults on the return path. In the three scenarios, all robots are operating normally at the simulation's beginning until t= 10000, shown as a red circle.

When the faulty robots stop on the main path, it directly affected the performance compared with the no-failure scenario because the robots avoided these main roads, which was reflected in their counting of the yellow turn and kept switching between the main roads. At the end of the simulation, they collected 96% of the total pucks. The other pucks are unreachable either in the faulty robot region or near the borders.

When the failure occurs on the return path, the performance equals the no failures scenario before t= 20000 because the robots are still collecting the pucks on the main roads. The performance degrades gracefully that due to the congestion, and the robots tried to avoid collision and find a road as much as possible. At the end of the simulation,

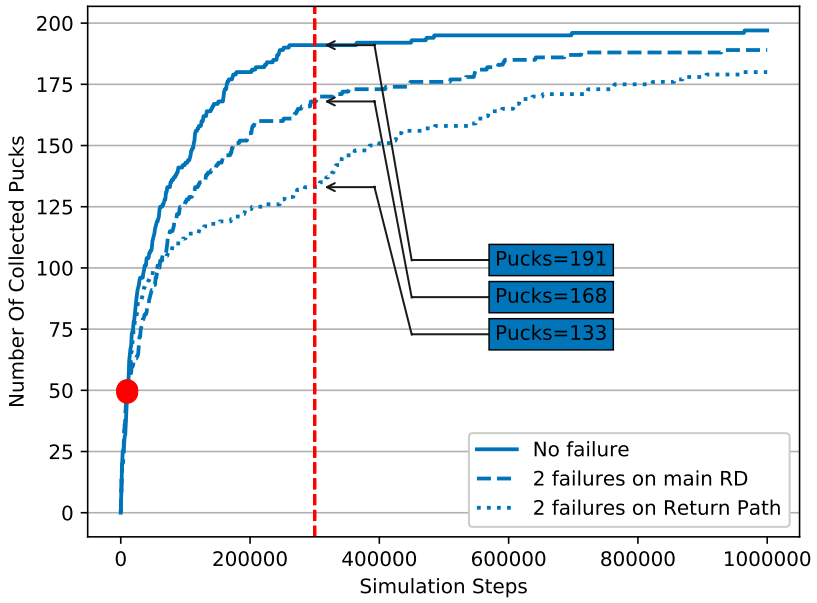


Fig. 5.25: Performance with different failure scenarios; the dashed red line shows the performance at $t = 300000$. The red circle indicates the time when the failure happened at $t = 10000$

the robots successfully collected 90% of the pucks.

5.3.2.2 Foraging using Random-Walk controller

In the Random-Walk controller, the collection station's position or the puck locations are unknown to the robots. They can detect the pucks if they are in front of them or on their left. Also, they can sense the blue collection station using their floor colour sensors. Figure 5.26 shows the state diagram of the robots running Random-Walk controllers to deliver pucks to the collection station. The main part of the controller is as follows: the robot moves straight for a random time, then spins and repeats that until it senses the collection station. The robot deposits the puck and continues searching for other pucks. If a robot collides with a wall, it moves backward and turns left. Additionally, if the robot detects another robot in front of it, it stops to give the other robot a chance to get away from it; if that collision has not been solved, it moves backward and turns left.

Fig. 5.27 shows the screenshots of the Random-Walk controller to collect 200 pucks

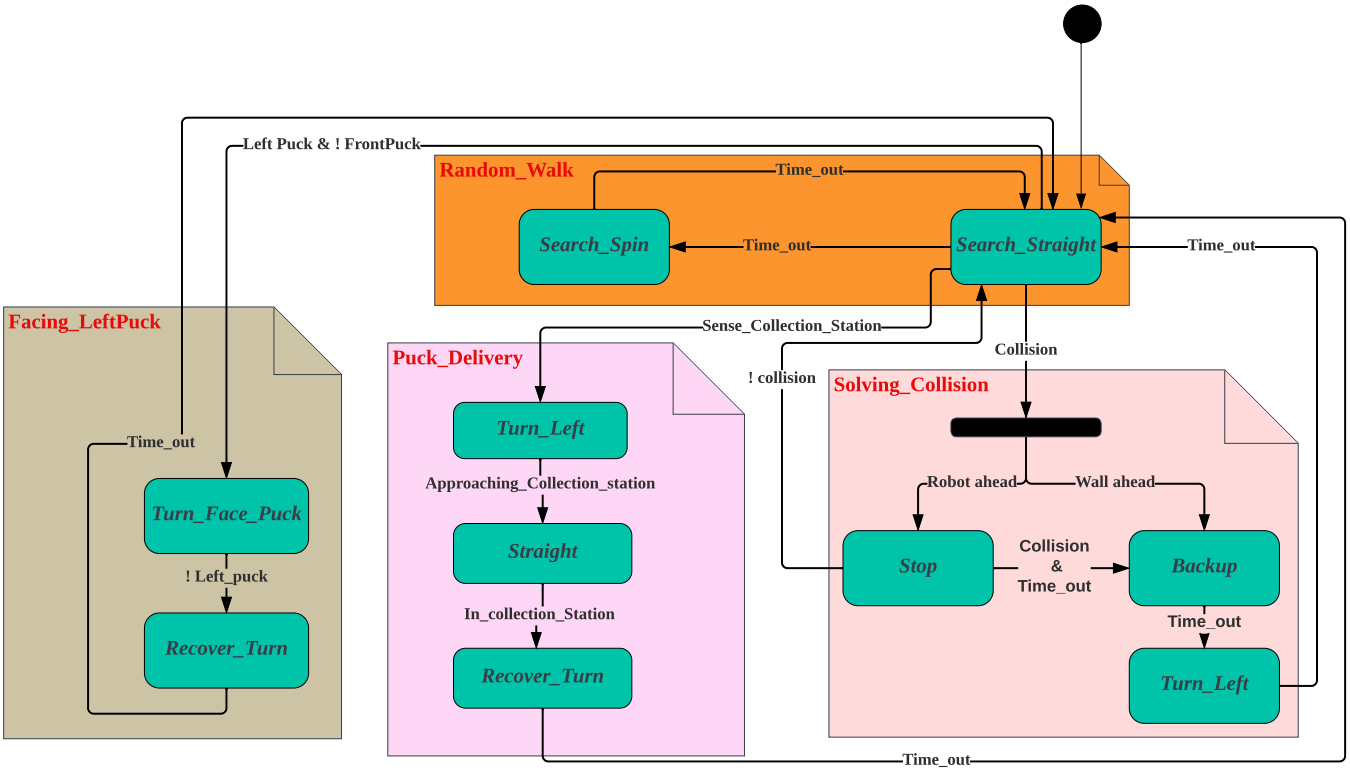
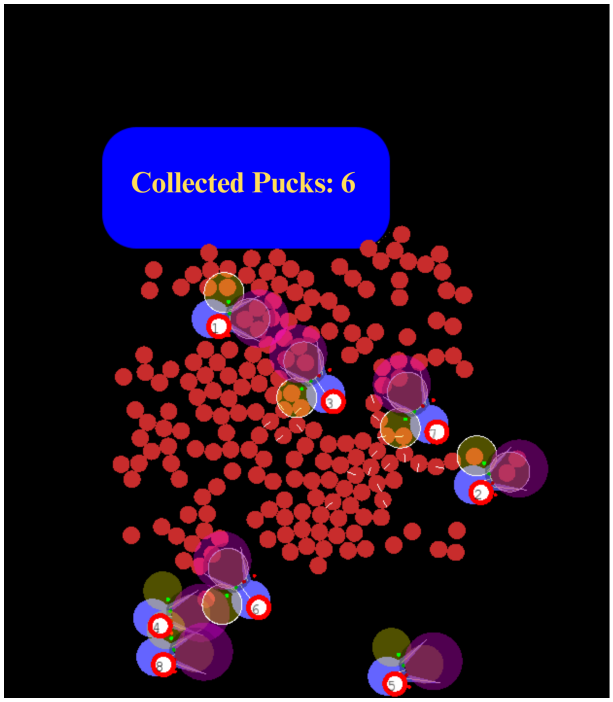
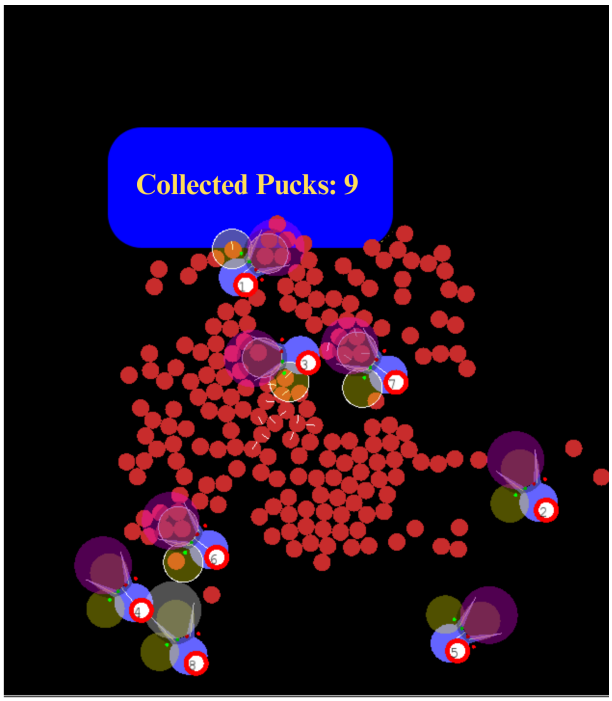


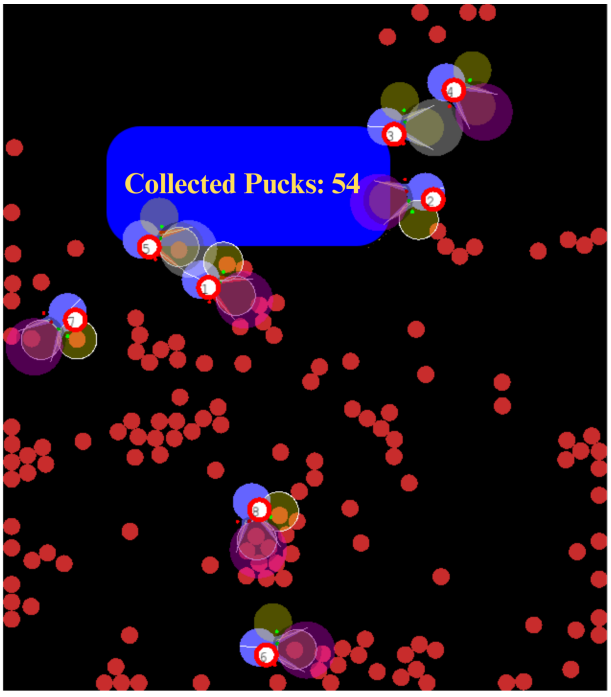
Fig. 5.26: State diagram of the pucks delivery using Random-Walk controller.



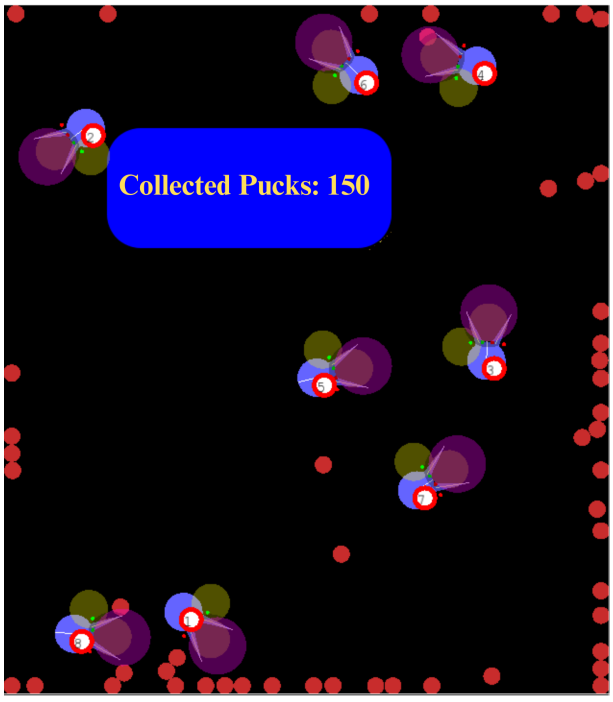
(a) $t = 100$, robots are searching for the pucks. There is a collision between robots #4 and #8.



(b) $t = 200$, collision is solved.

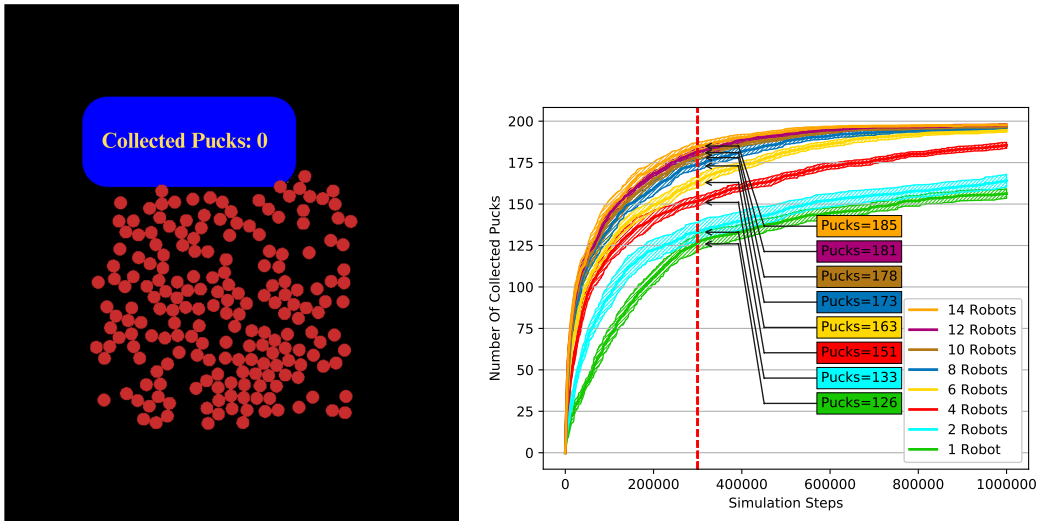


(c) $t = 10600$, The pucks are pushed all over the environment by robots.



(d) $t = 294200$, 75% of the pucks are delivered.

Fig. 5.27: Screenshots of 8 robots delivering pucks to the collection station using Random-Walk controller. The pucks disappear when they hit the blue area.



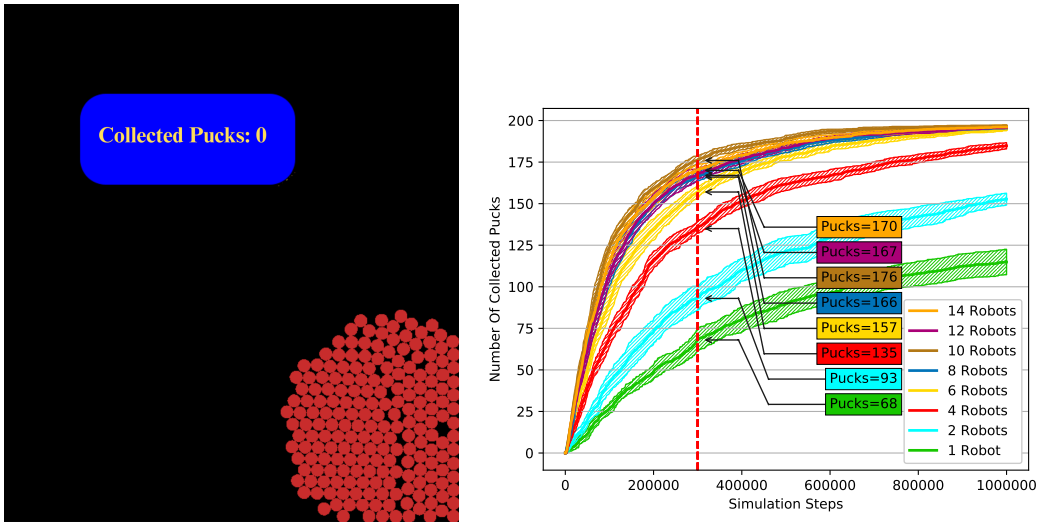
(a) Pucks start position at t= 0

(b) Performance with different numbers of robots. The dashed red line shows the performance at t= 300000.

Fig. 5.28: Pucks are positioned in center of the environment; the **Random-Walk controller** is applied.

placed in the center of the environment. At $t = 100$, robots walk through the environment searching for pucks. Robots #4 and #8 detect each other, so they wait for different random times for anyone to take action and rotate away from the other. At the same time, robot #6 detects pucks on its left and front, so it decides to move straight and push the front pucks. At $t = 200$, robot #4 moves first backward and turns left away from robot #8. Robot #1 is about to approach the blue station and deliver five pucks there. At $t = 10600$, some pucks are delivered, but the other pucks are pushed all over the environment due to the random movement of the robots. At $t = 294200$, 75% of the pucks are successfully delivered to the collection station.

Fig. 5.28 and 5.29 present the results for varying numbers of robots for the 200 pucks placed in the center and bottom-right of the environment, respectively. In Fig. 5.28, increasing the number of robots increases the performance, while in Fig. 5.29, performance starts declining after 10 robots. Although the robots follow the Random-



(a) Pucks start position at t= 0

(b) Performance with different numbers of robots. The dashed red line shows the performance at t= 300000.

Fig. 5.29: Pucks are positioned in the **bottom-right corner** of the environment; the **Random-Walk controller** is applied.

Walk controller, increasing collisions among robots and avoiding these collisions give them a good chance to explore the environment better than a small number of robots.

When the pucks are placed in the center, it is easy for the robot to find them and collect them faster than the pucks placed bottom-right in a specific position. That clearly appears in the performance of one robot in both graphs; the one robot collects around 153 and 120 pucks in the center and bottom-right environments, respectively.

5.3.2.3 Discussion

We present Fig. 5.30 and 5.31 as a comparison between the Road-Following and Random-Walk controllers for the two puck placements in the center and bottom-right of the environment, respectively.

In Fig. 5.30, at the beginning of the simulation, when the pucks are placed in the center, robots in the Random-Walk can find them quickly, so sometimes performance is

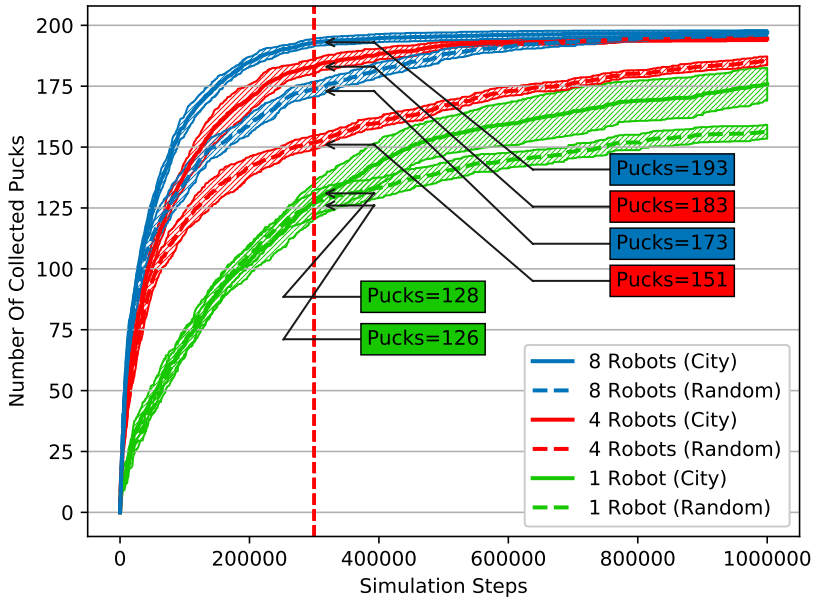


Fig. 5.30: Comparison between Random-Walk and Road-Following controllers with pucks placed in the **center** of the environment; the dashed red line shows the performance at t=300000.

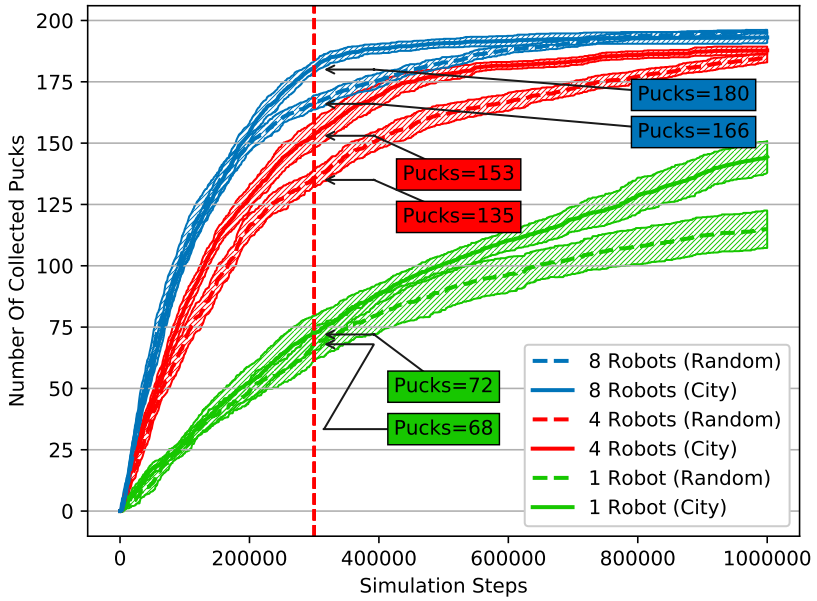


Fig. 5.31: Comparison between Random-Walk and Road-Following controllers with pucks placed in the **bottom-right** of the environment; the dashed red line shows the performance at t=300000.

equal to the obtained from the Road-Following controller, even better in the case of one robot (in green), because the robot has freedom in their movement. Unlike the one robot in the Road-Following controller it searches for a road first and follow it. However, after $t = 300000$ in the Random-Walk, the pucks are pushed over the environment due to their random walk. So, the performance is significantly reduced compared to the Road-Following controller, in which robots know their roads and have organized movement towards the collection station. Hence, the performance of 4 robots in Road-Following can outperform 8 robots applied Random-Walk controller.

In Fig. 5.31, placement of the pucks bottom-right is challenging for the robots to find in both controllers. When one robot operates alone, it collects 60% of pucks using Random-Walk and 74% using the Road-Following controller. Increasing the number of robots in both controllers increases the controller's performance because there are more chances to find and collect these pucks. In the case of using 8 robots, at the beginning of the simulation, Random-Walk outperforms the Road-Following due to the large number of robots and randomness that can easily find the pucks. However, the 8 robots in the Road-Following took some time to organize their movements and set their yellow turn counters, but it outperformed the Random-Walk after 180000-simulation steps.

5.4 Hardware Implementation

As we mentioned in Section 2.3.4, there are different ways to provide the potential field to the robots. The potential field could be sent to the robots based on their position, robots can sense the potential from the environment, or robots could calculate the potential field based on their surroundings.

In our implementation, the robots rely on local sensing only, so we precalculated the scalar field and generated the coloured image representing the roads the robots will follow. The generated image is projected on the LCD television to evaluate our

(a) Top view

(b) Side view

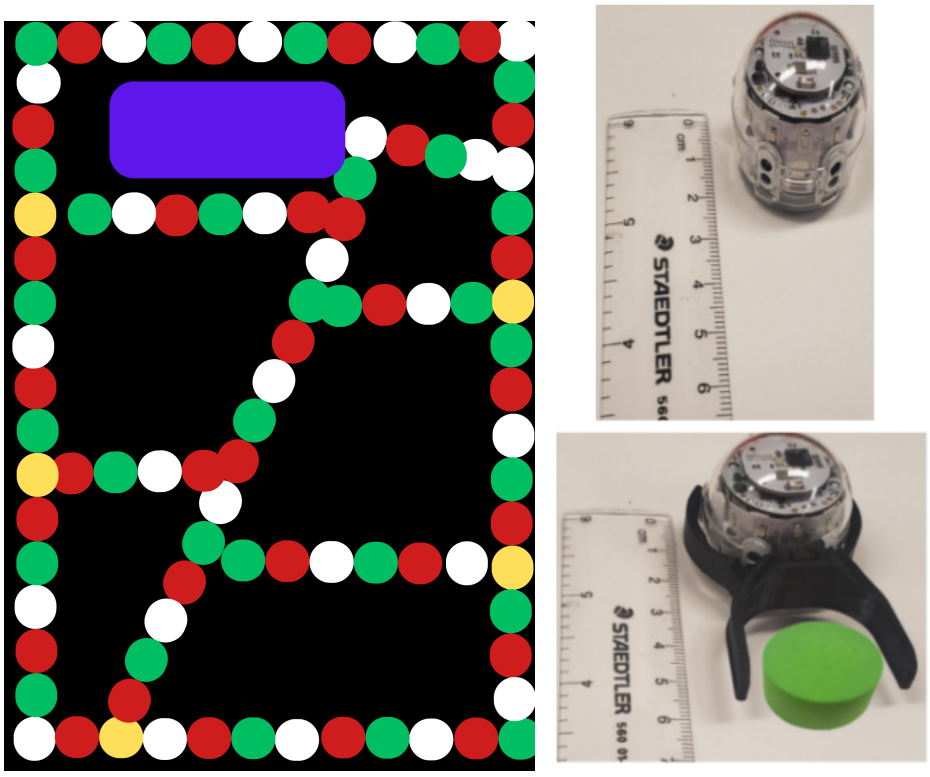
Fig. 5.32: Working environment, 75-inch diagonal LCD television.

controllers, and our robots equipped with colour sensors are built to detect the image colours. We use a 75-inch diagonal LCD television as a working environment to project that image, as shown in Fig. 5.32.

5.4.1 Candidate Robots for Hardware Experiment

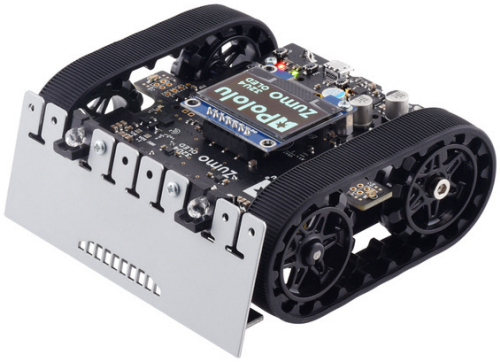
We have two available robots in our lab, OZOBOT and Zumo 32U4 OLED; they are potential candidates for evaluating our proposed controllers. The OZOBOT [118] is a simple, small programmable robot. It is primarily used as an educational tool in schools and homes to teach programming. Its sensors can detect lines, colours, and codes drawn on paper or digital screens. The drawback is that it has only a single colour sensor, so the robots will not know the direction of the line. However, there is a solution if we redesign the roads as a pattern of colour dots line, for example (red, green, then white) dots,and the radius of these dots should be a function of the robot's speed, so at every simulation step, the robot should encounter a new dot. OZOBOT can follow that pattern and know the direction of the road as shown in the left part of Fig. 5.33. The robot can distinguish between the left and right sides of the highway if the robot detects two consecutive red dots, which means the robot is coming from the left side. It should turn left to follow the highway. However, if it sees two consecutive green dots, it indicates the robot is coming from the right. It should turn right to follow the highway. After the OZOBOT deposits the puck, it takes the return path declared as two white colours, the turning places coloured yellow. The other problem is the need for a gripper for the foraging task, so we 3D printed its passive gripper as shown in the right part of Fig. 5.33. Another challenge is that we could not form a swarm of OZOBOT due to its availability on the market at that time.

The second candidate is the Zumo 32U4 OLED robot [119] (Fig. 5.34(a)). It is a programmable robot kit developed by Pololu Robotics and Electronics. The robot is



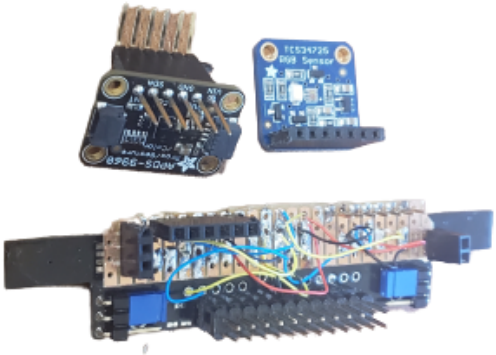
(a) (b)

Fig. 5.33: Using OZOBOT to perform Road-Following controller in foraging task. Left figure is sketch of the city roads when we use one colour sensor. Right figure is OZOBOT with and without the passive gripper.

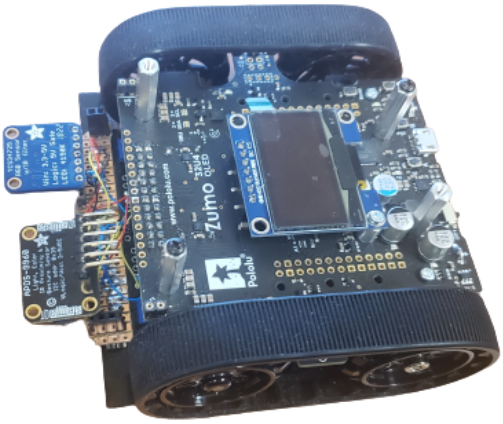


(a) Zumo 32U4 OLED robot

(b) Front Sensor Array



(c) Front Sensor Array with two colour sensors.



(d) Zumo robot with two colour sensors.

Fig. 5.34: The first robot design adds two colour sensors to the Front Sensor Array.

based on the ATmega32U4 microcontroller, which provides a wide range of input and output options and is programmable using the Arduino development environment. Zumo supports various sensors and actuators that can be easily integrated into the robot's platform, allowing for customization and expansion of its capabilities. Also, Zumo provides a separate board called Front Sensor Array [120], shown in Fig. 5.34(b). This board has five line sensors and three proximity sensors connected to Zumo's microcontroller and it incorporates I2C lines for communication with the existing sensors on that board. The five-line sensors attached to the Zumo chassis face downward and can help the Zumo distinguish between light and dark surfaces.

5.4.2 I2C Protocol to Communicate with Extra Sensors

I2C (Inter-Integrated Circuit) is a serial communication protocol used for connecting low-speed peripherals to a motherboard or microcontroller. I2C uses two wires for bidirctional communication SDA (Serial Data Line) and SCL (Serial Clock Line), as shown in Fig. 5.35. The Master (ATmega32U4) initiates the communication and generates the clock signal, while the slave responds to the master's command. Each device on the I2C bus should have a unique address; in our case, we use two colour sensors: RGB Color Sensor (TCS34725) [121] and Proximity, RGB, and Gesture Sensor (APDS-9960) [122]. TCS34725's address is 0x29, and APDS-9960 's address is 0x39.

5.4.3 Building Robot with Color Sensors

5.4.3.1 First Prototype

In order to use Zumo 32U4 OLED in our experiment, we should add colour sensors to detect the coloured scalar field projected on TV. We connected two I2C colour sensors, TCS34725 [121] and APDS-9960 [122] to the Front Array sensor as shown in Fig. 5.36. Our first prototype has two colour sensors mounted over the front sensor array, as shown

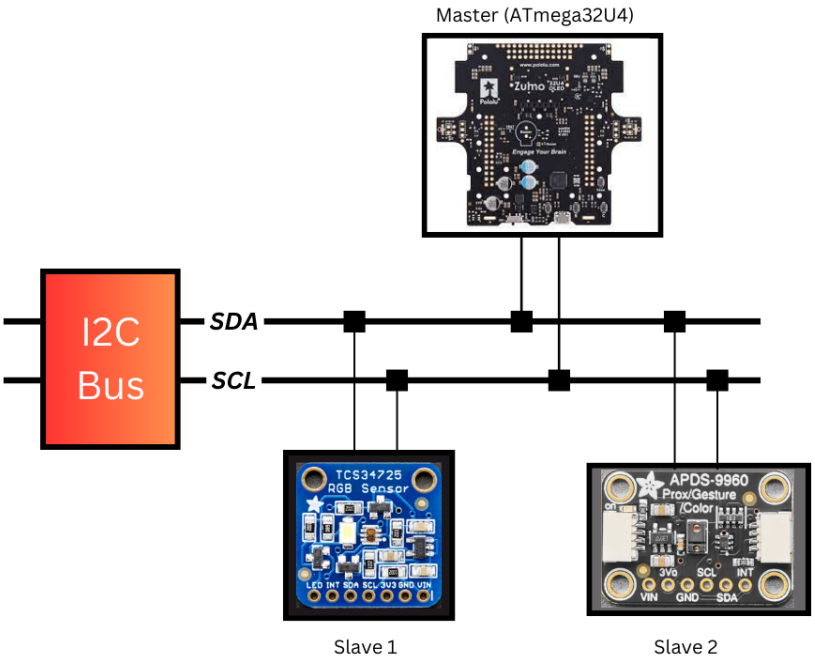


Fig. 5.35: I2C communication protocol.

in Fig. 5.34(c).

5.4.3.2 Second Prototype

In the first prototype, there was sometimes a latency in reading the colour for one sensor, which affected the robot's movements. In the second prototype, we provided the robot with additional information and improved its moving accuracy, especially when the robot encountered curves or intersections; we added 4 colour sensors facing downward to give more information about the robot's position relative to the line. If one sensor fails or encounters an issue (e.g., due to changes in lighting conditions), the other sensors can compensate.

For the second prototype, we added four colour sensors APDS-9960 and designed a 3D model as a sensor holder instead of using the front sensor array board as shown in Fig. 5.37(a). Also, we added three extra APDS-9960 sensors, to work as proximity

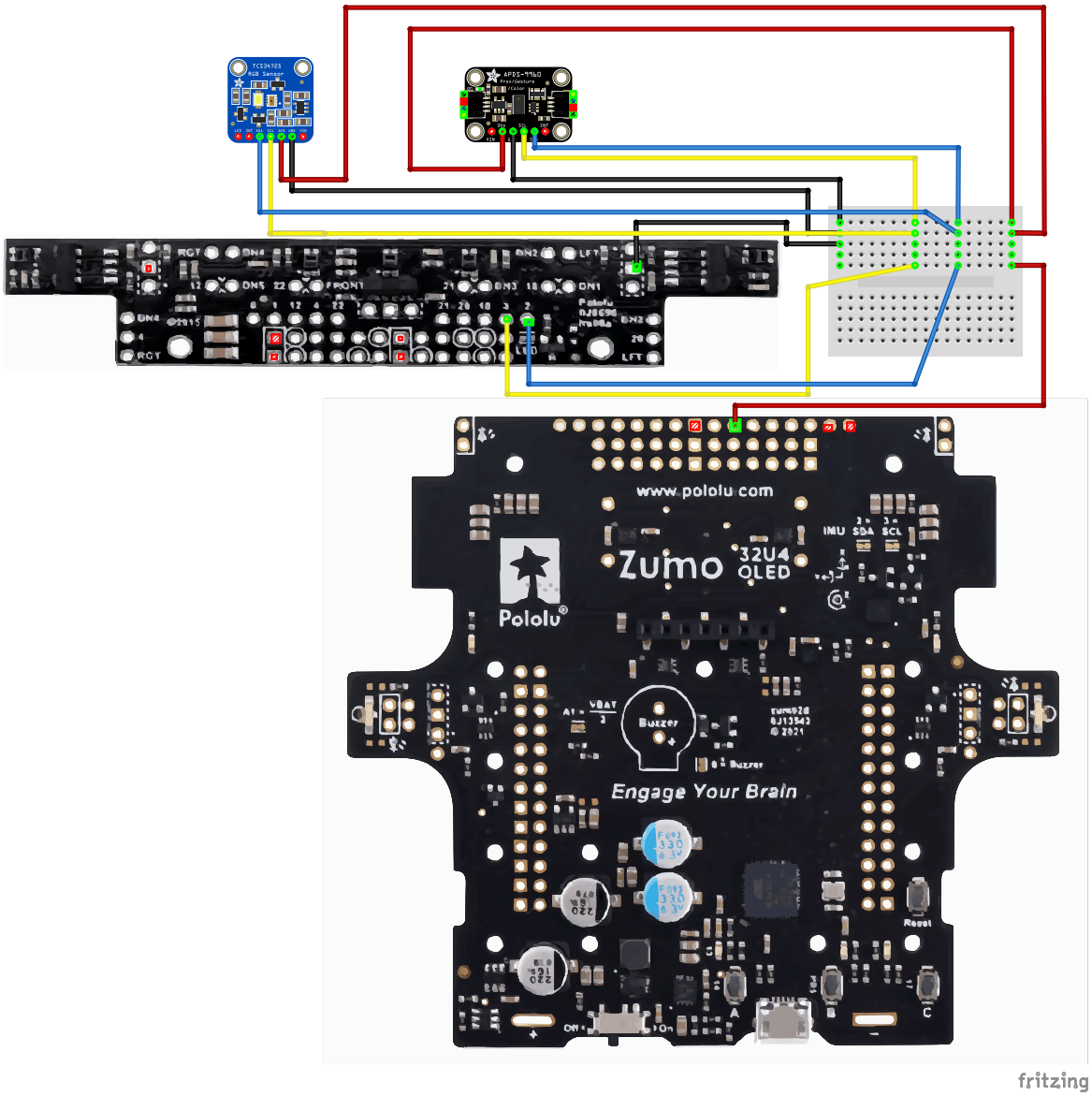
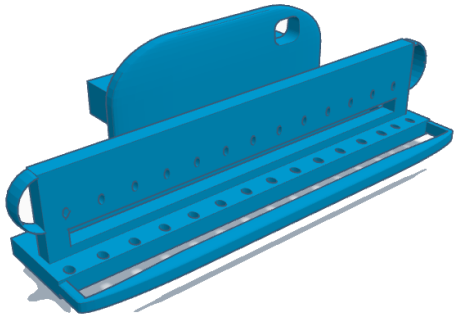
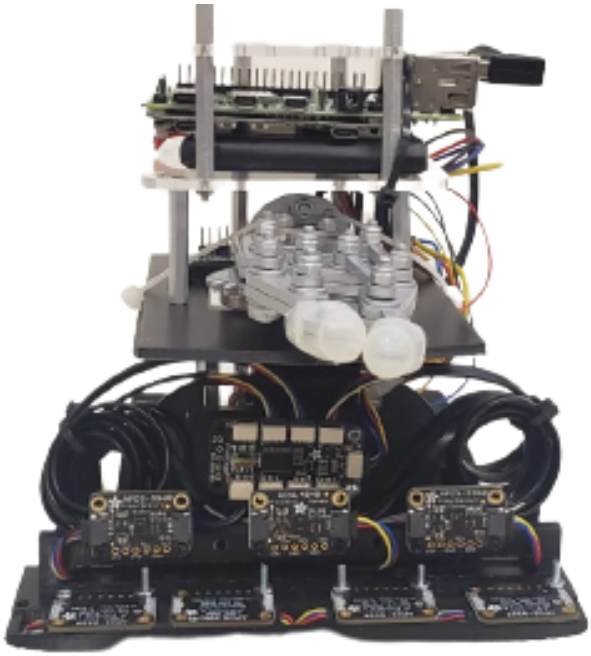


Fig. 5.36: The first prototype scheme for the robot used Fritzing software.

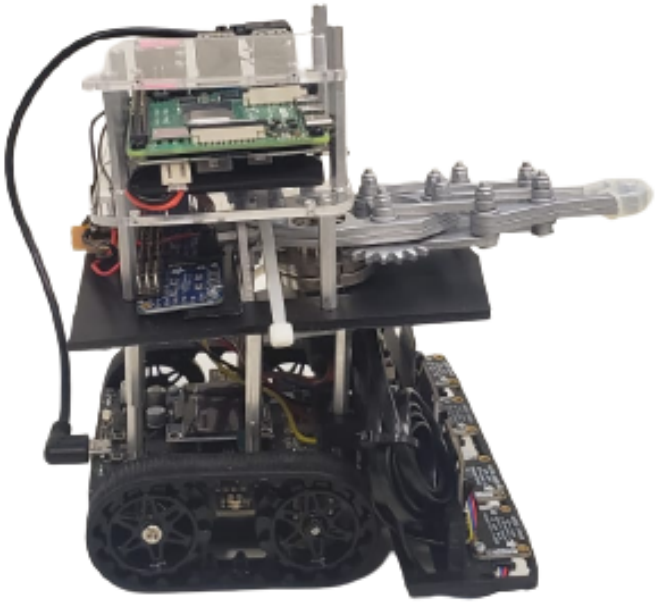
and colour sensors to detect the distance and the colour of any obstacles in front of this romultiplexerbot. Because all of these sensors are the same type and have a fixed I2C address (0x39), we connected all sensors to a multiplexer (PCA9548) [123], and then this multiplexer connected to the I2C ports on Zumo. The multiplexer has an address 0x70. We also extended this design by attaching a gripper to collect the objects in case we use these robots in foraging task. The motorized gripper was connected to the



(a) 3D model for 4-floor sensor holder.



(b) Front View



(c) Side View

(d) 3D model for an object to be collected by the robot.

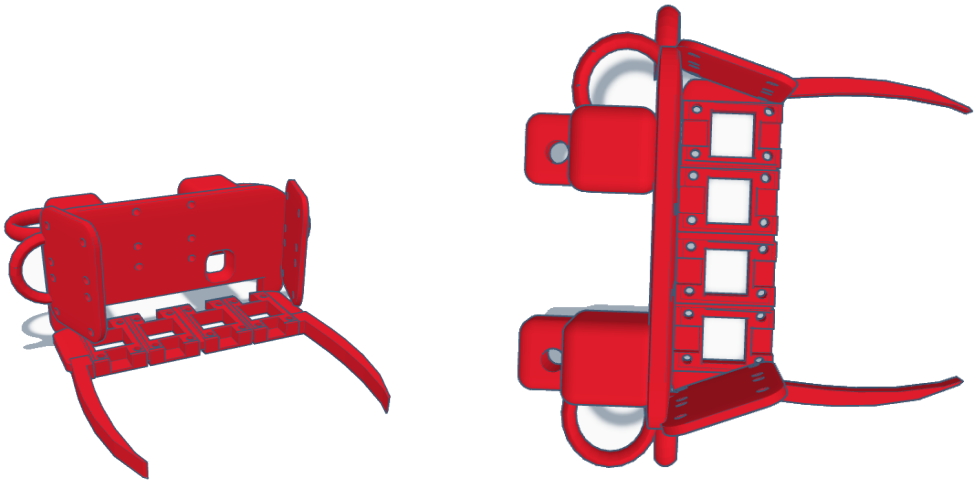
Fig. 5.37: Prototype 2 incorporates 4 floor-coloured sensors (APDS-9960), 3 proximity sensors (APDS-9960) and a gripper.

servo driver (PCA9685) [124] as shown in Fig. 5.37(b) and 5.37(c). The servo driver also used the I2C protocol; its address was 0x40. We used the Raspberry Pi 4B single-board computer [125] (RPi) to upload the control program to the Zumo's microcontroller remotely. We installed Arduino CLI package on a Raspberry Pi, and we sent our arduino sketch from our computer to Raspberry; then Raspberry uploaded the program to Zumo using Arduino CLI. Arduino CLI is a software for uploading our Arduino sketch from the command line [126].

5.4.3.3 Third Prototype

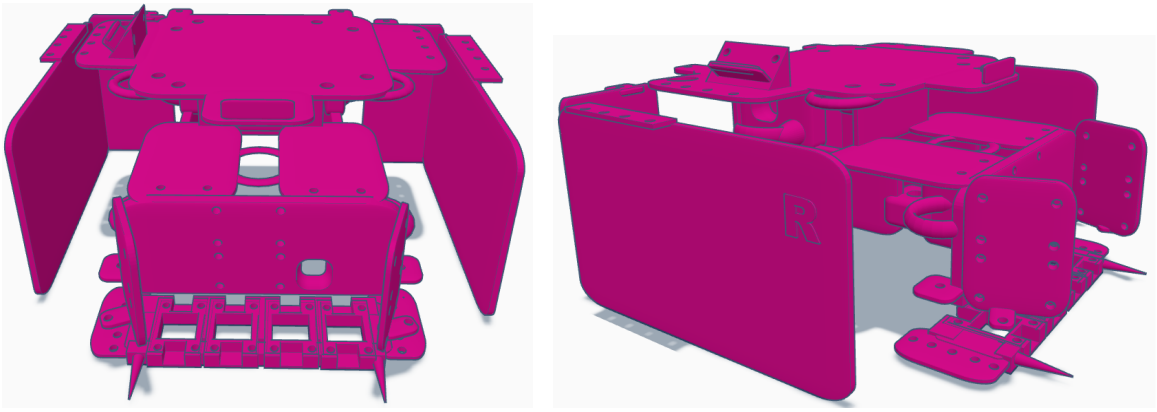
We decided to switch to the passive gripper, as shown in Fig. 5.38, instead of the motorized (active) gripper due to the simplicity of the passive gripper design and to reduce the cost of the active gripper and its servo driver (PCA9685). We added three proximity sensors (VL53L4CD) to increase the robot's sensing coverage of its environment, particularly the ability to sense and distinguish pucks, other robots, and walls. Fig. 5.39 shows the final 3D design for our robots to carry the sensors and two multiplexers. Fig. 5.40 shows our robot in its final version. This design incorporates four-floor colour proximity sensors (APDS-9960) facing downwards, three proximity sensors (VL53L4CD), and three coloured proximity sensors (APDS-9960). We placed a proximity sensor (VL53L4CD) up and a colour proximity sensor (APDS-9960) under it to distinguish between walls, robots and pucks by their colour and distance.

We define walls, robots and pucks in different colours: green, red and blue, respectively. The height of the puck should be lower than the proximity sensors (VL53L4CD), so the pucks will be detected only by the colour proximity sensors (APDS-9960). Fig. 5.41 shows the high-level visual representation of the interaction of robots and other objects (pucks, walls, other robots). We represent the VL53L4CD as a yellow block and APDS-9960 as a purple block. Robot #1 detects the right green walls, blue puck and red robot #2. Robot #2 detects the left green walls and the puck placed on its left.



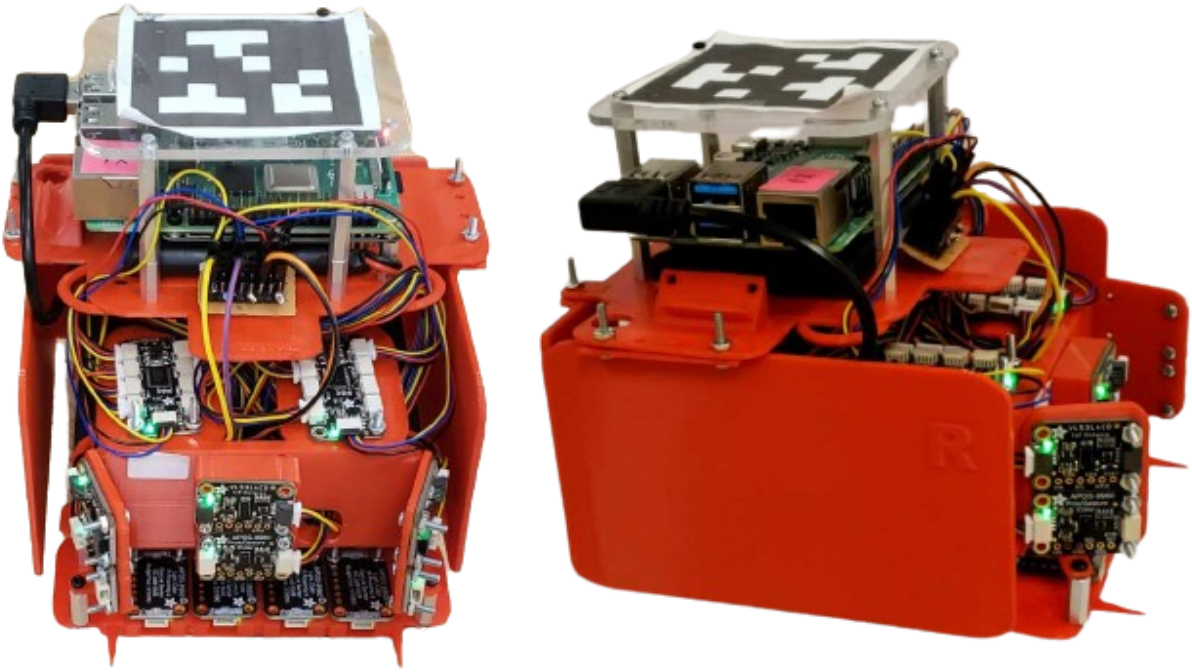
(a) Front view (b) Top view

Fig. 5.38: 3D model for floor colour sensor holder horizontally aligned.



(a) Front view (b) Side view

Fig. 5.39: 3D model for colour and proximity sensors and Raspberry Pi holder (Third robot prototype).



(a) Front view (b) Side view

Fig. 5.40: The final design for the third robot prototype.

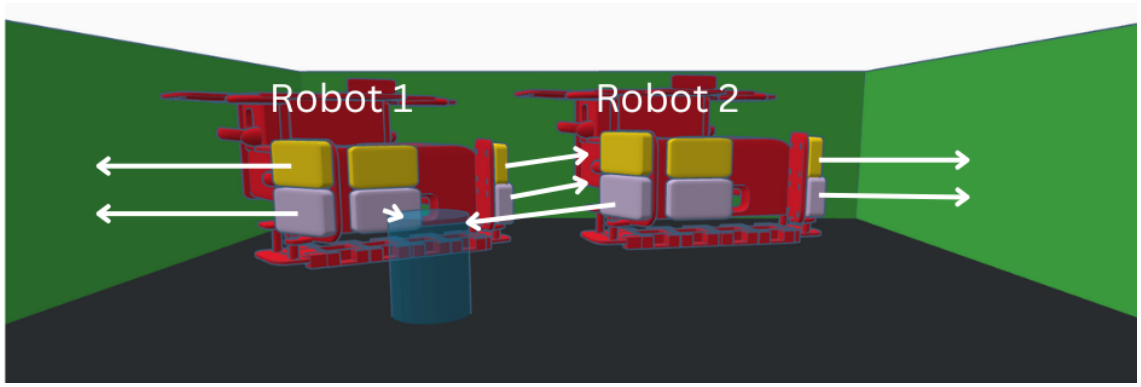


Fig. 5.41: A high-level visual representation of how we used and placed the proximity sensors (VL53L4CD). This is represented by yellow blocks, and the colour proximity sensors (APDS-9960), represented by purple blocks.

We connect these I2C sensors with two I2C Multiplexers (PCA9548) which connected to the Zumo extension ports as shown in Fig. 5.42.

5.4.4 Experiments

We used the third prototype for real-world experiments. The robots operate on the 75-inch LCD television. We projected the coloured scalar field, our Road-city network design, on the screen, and the robots followed the Follow-Road controller, which was used in the simulation implementation and presented in Fig. 5.11. Fig. 5.43 shows overhead images captured when one robot is placed on one of the main roads. The robot keeps moving on the roads until it reaches the aggregation area. Fig. 5.44 shows one robot starting from 10 different positions and showing its trajectories. For example, in trials #1,#2, #4, #9, and #10, robots started on one of the main roads and then took the highway to the blue area. In trial #8, the robot was placed on the black area, so it moved randomly until it found the main road and moved on it. In trial #5 was placed the robot in the black area, it moved randomly until it found the return path, followed it, then took the yellow turn to one of the main roads.

To show how the robot handles collisions, we used one robot as an obstacle and placed our robot in three different positions facing that obstacle. We repeated this experiment four times with different obstacle positions. Fig. 5.45 shows the red robot, which is used as an obstacle and the other robot trajectories to avoid the red robot and reach the aggregation area. Fig. 5.45(a) shows the first experiment in which we placed the static obstacle represented by the red robot in the blue area, and our robot in the trial #1 and #3 avoided the collision by moving backward then turning left. In trial #2, the robot was placed on the right of the highway; it did not face the obstacle because it entered the blue area with a different angle. Fig. 5.45(c) and 5.45(e) show the second and third experiments, where we placed the obstacle in a black area on the right and left of the

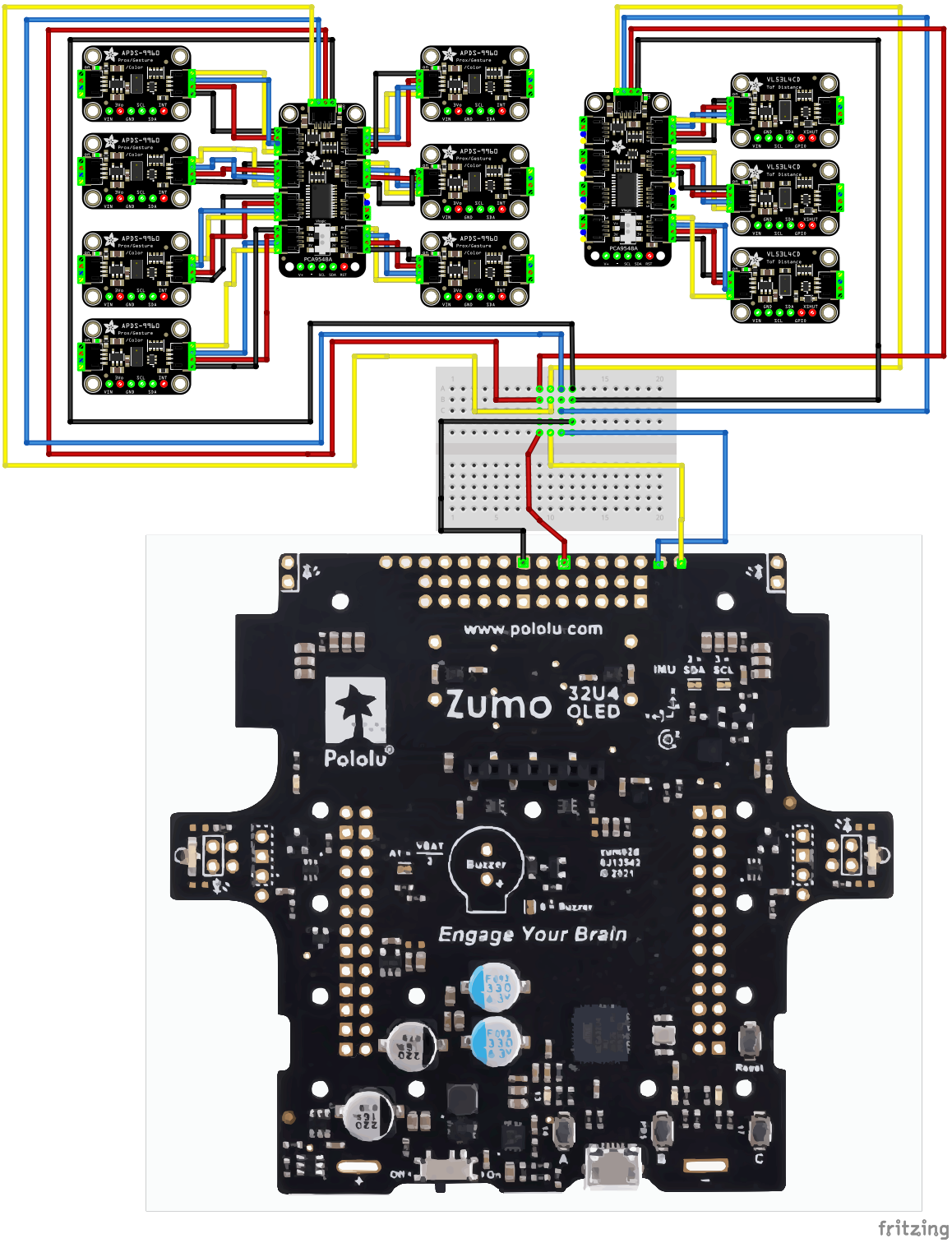
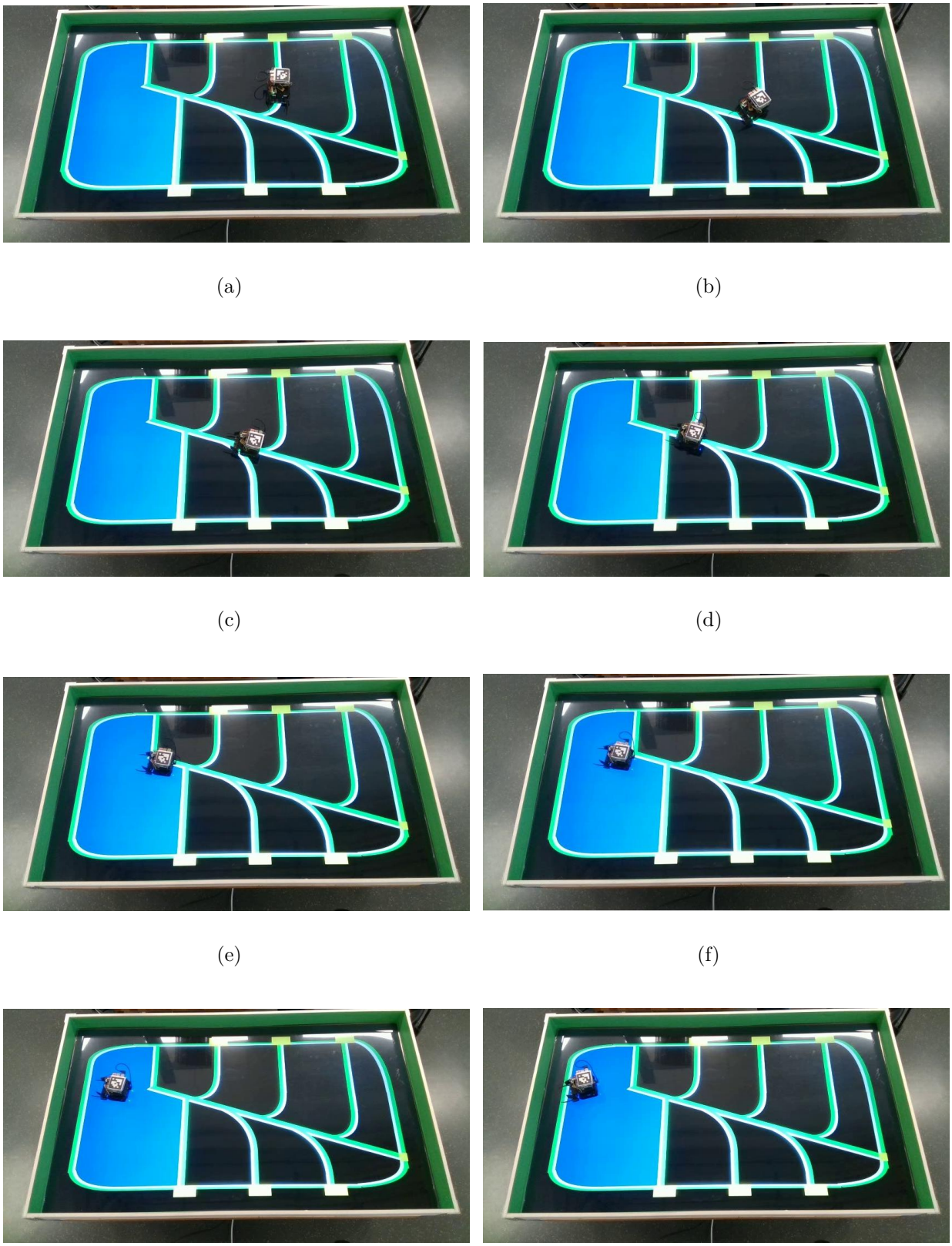


Fig. 5.42: The third prototype scheme for the robot used Fritzing software.



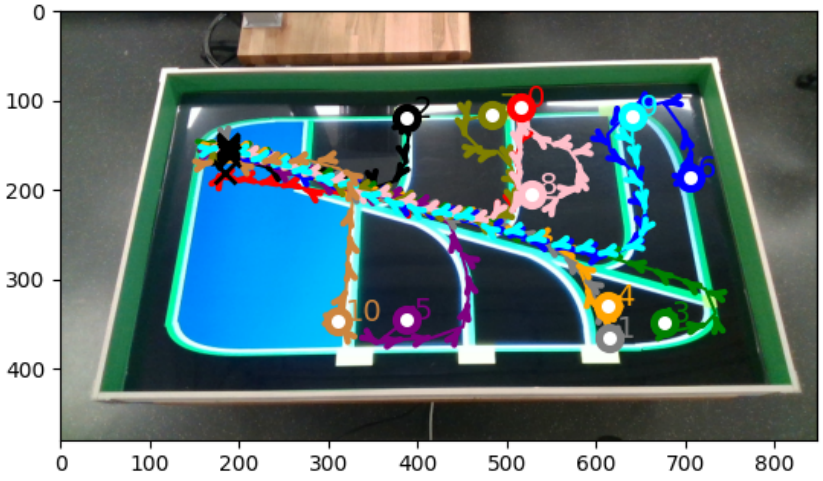
(a) (b)

(c) (d)

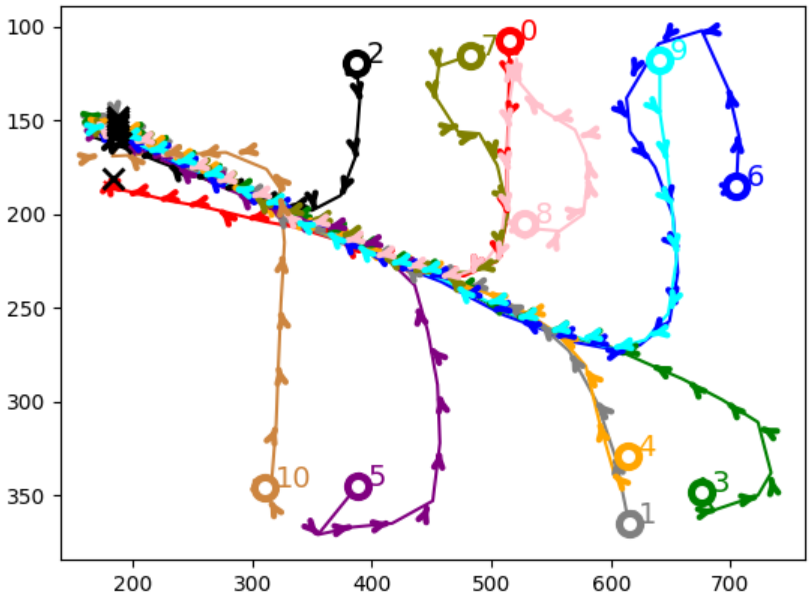
(e) (f)

(g) (h)

Fig. 5.43: Overhead image of one robot homing using city road network and Road-Following controller.



(a)



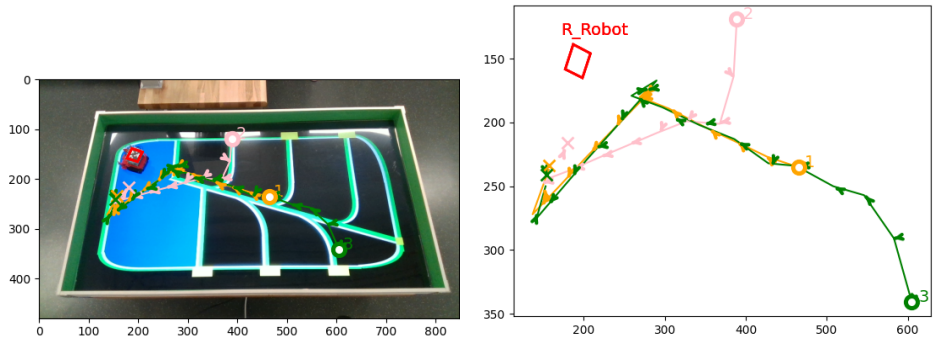
(b)

Fig. 5.44: Aggregation to blue destination using one robot from different start positions. The colour circles are the start points.

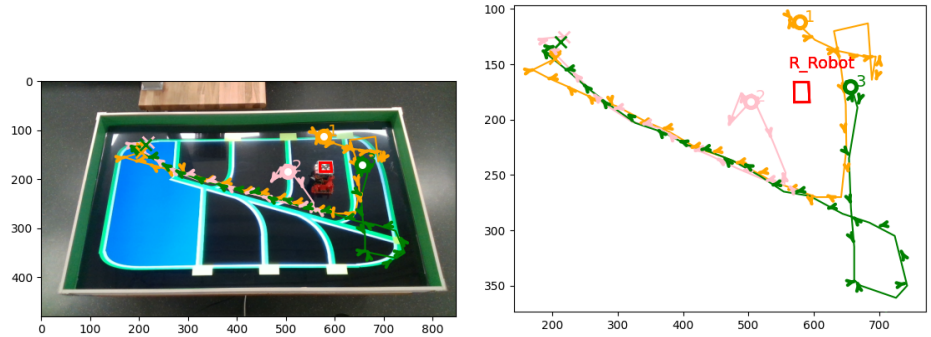
highway, respectively. In the fourth experiment, we put the robot on a main road, as shown in Fig. 5.45(g).

We conducted four trials to show the performance of 2 robots together in the environment as shown in Fig. 5.46 and 5.47. We attached an AprilTag to the robots; then, we tracked the robots using an overhead camera to show their trajectories while they performed the aggregation task. The printed numbers on the figures are the frame numbers to indicate which robot reached the aggregation area earlier.

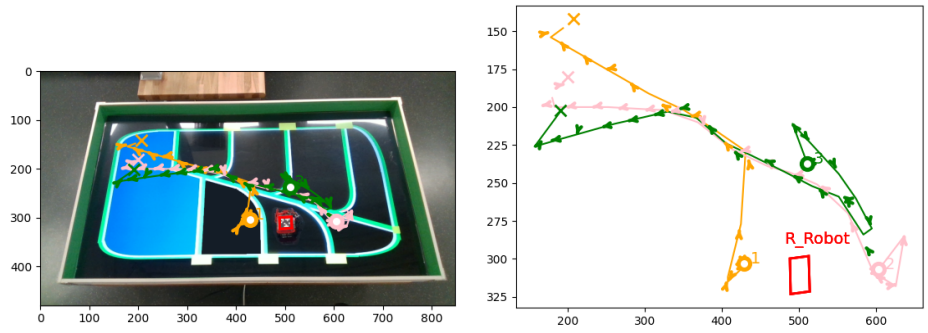
Fig. 5.48, shows overhead images of two robots in operation to find the aggregation area. We started our experiment by placing the robots in the collision position to see how they handled this situation, as shown in Fig. 5.48(a). Both robots detected the collision, moved backward, then turned left. Eventually, they reached the blue aggregation area by following the roads. A video showing 1, 2, 3 and 4 robots aggregating in real-world trials is available at https://youtu.be/Ljkx7Itc_YQ.



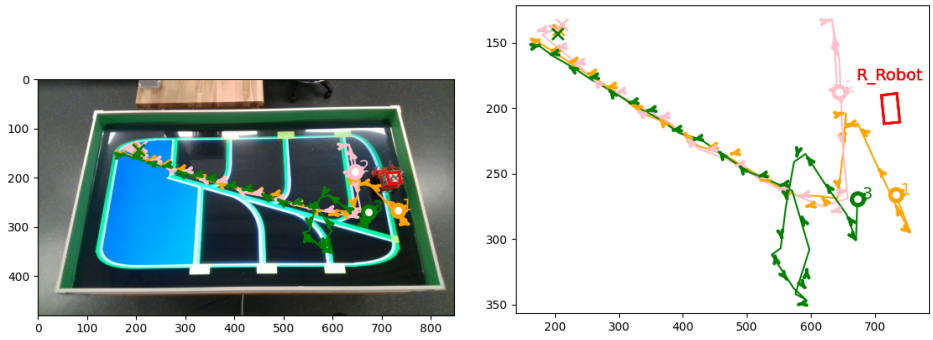
(a) (b)



(c) (d)

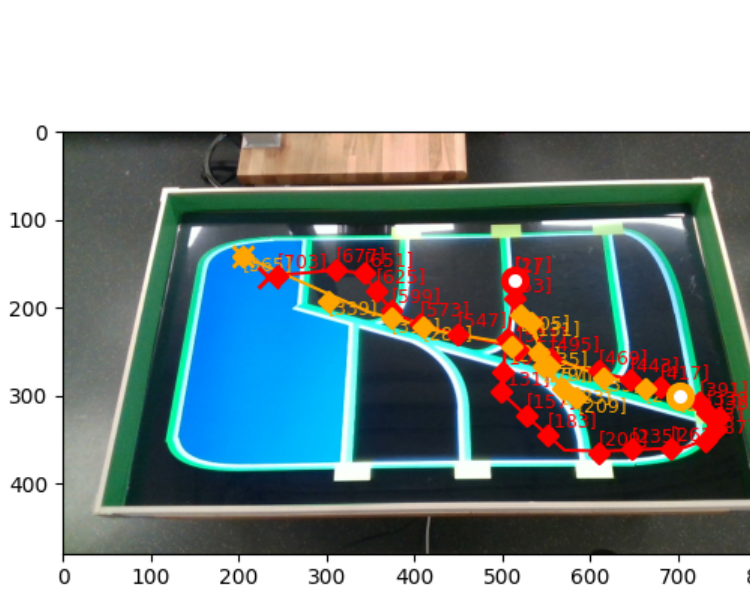


(e) (f)

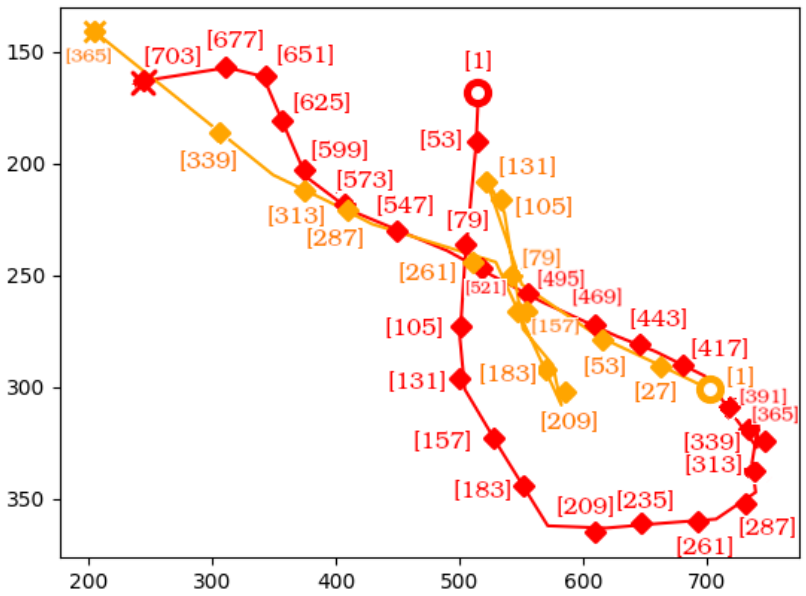


(g) (h)

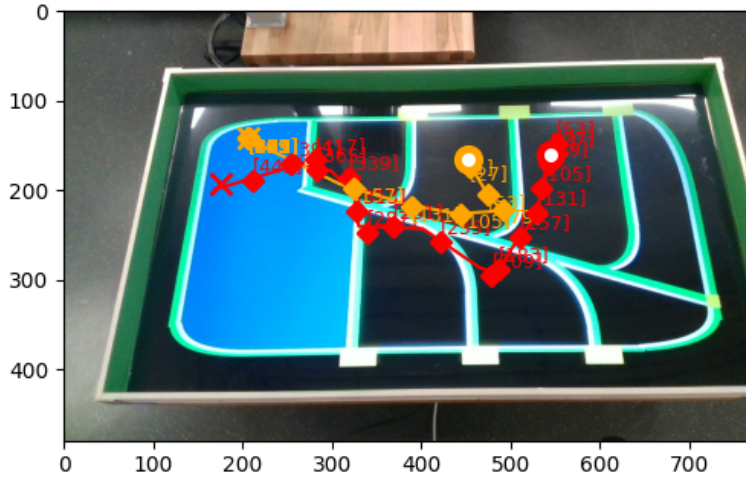
Fig. 5.45: The overhead image for the two robots is in red and black colours. The red robot was used as a static obstacle to test the collision behaviour of the second robot. For each position, we have 3 trials represented in orange, pink and green colours. In each trial, we placed the black robot facing the red robot in different directions.



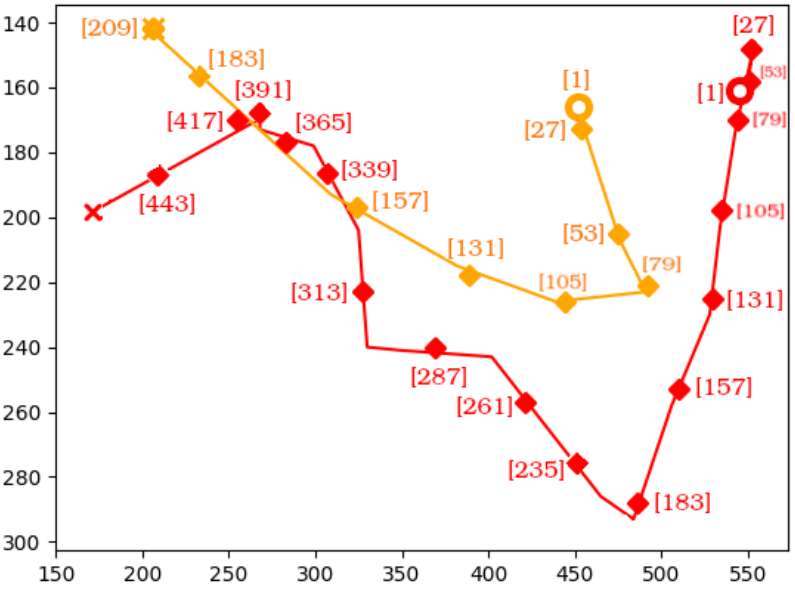
(a)



(b)



(c)



(d)

Fig. 5.46: The first two trials for two robots aggregate to a blue area. The red and orange trajectories belong to red and black robots, respectively. The numbers depicted on the images are the frame numbers to show which robot reaches the blue area quicker.

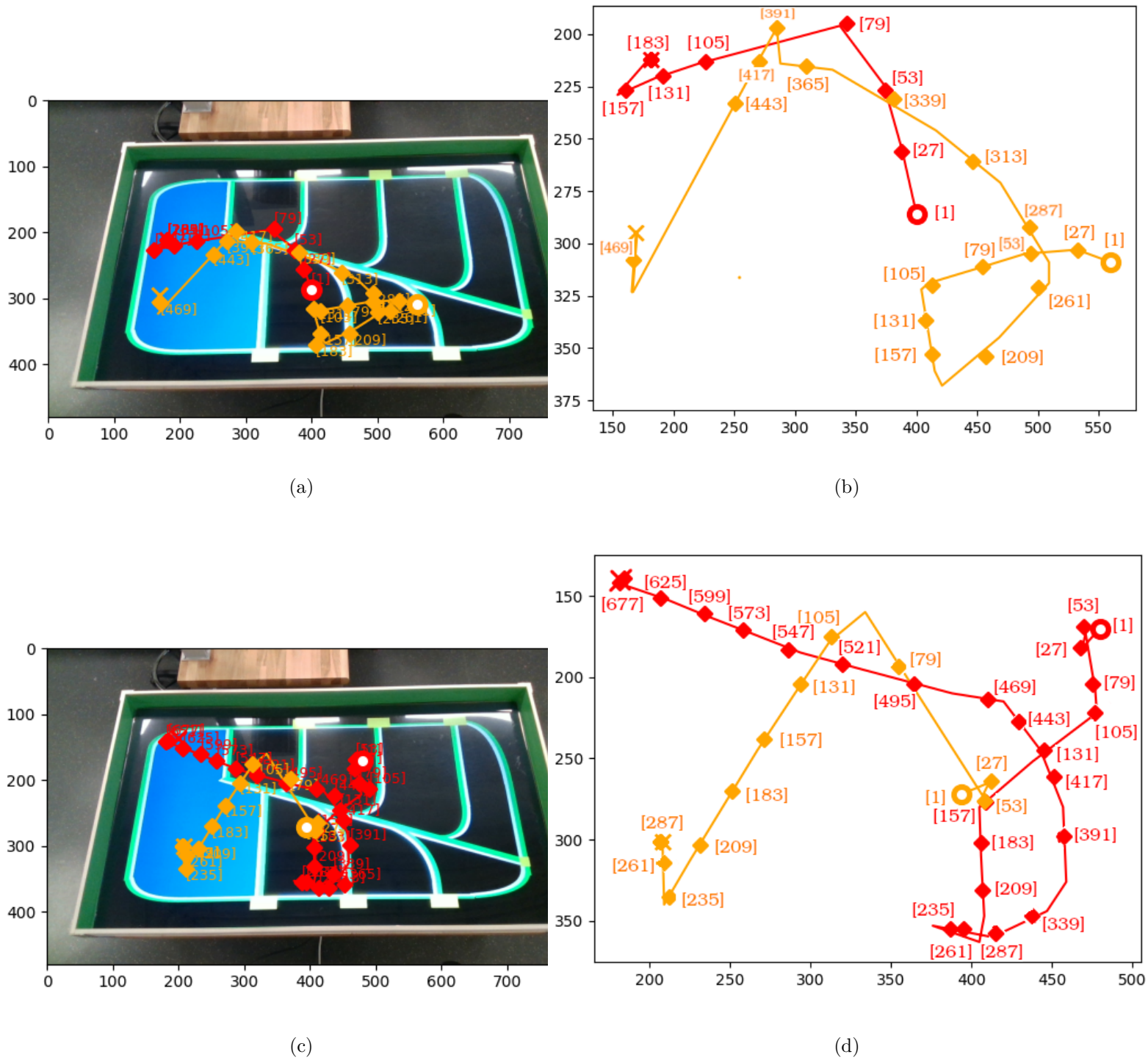
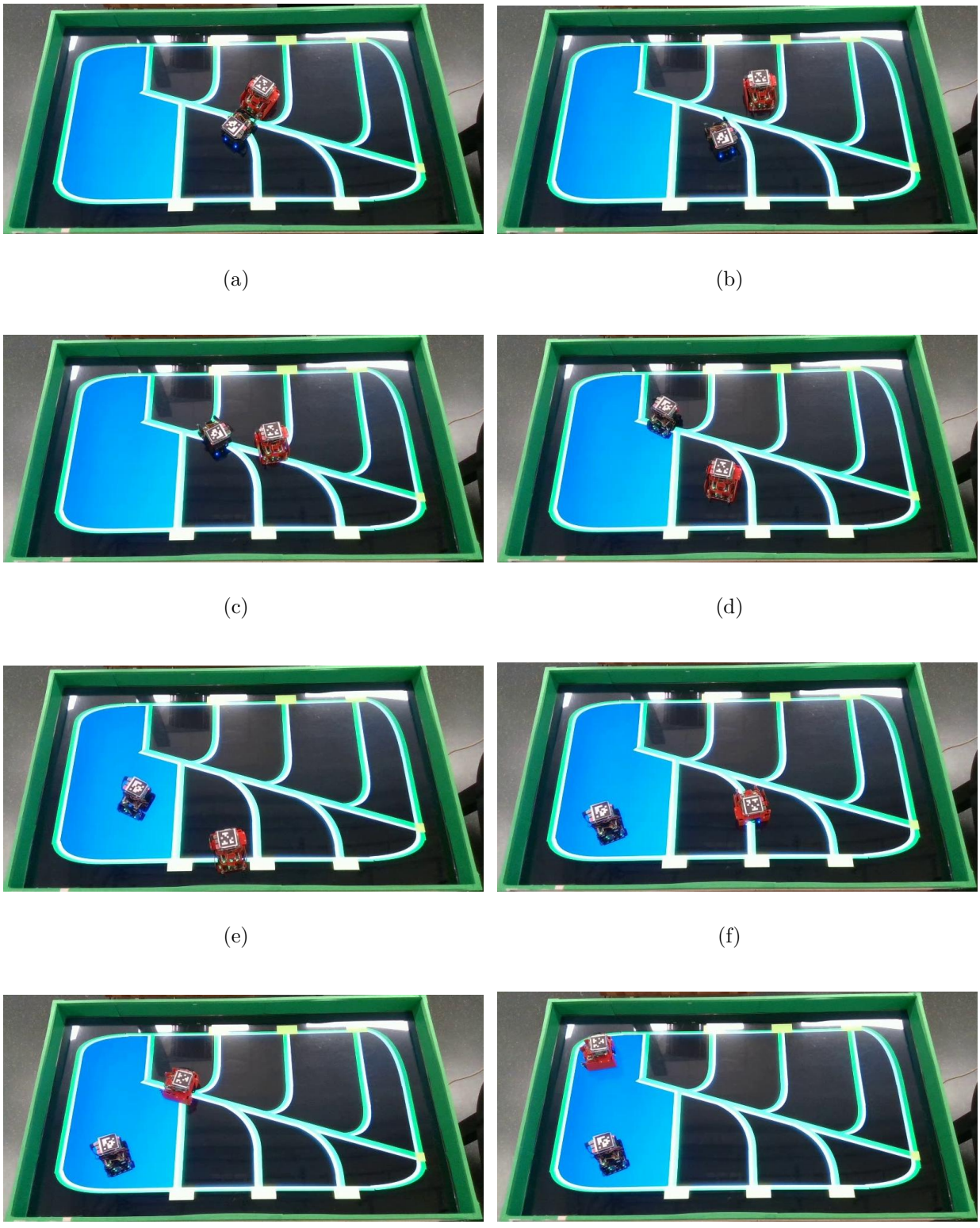


Fig. 5.47: The third and fourth trials for two robots aggregate to a blue area. The fourth trial collides around frame = 27 in orange trajectories (black robot) and frame = 53 in red trajectories (red robot) at the beginning of the experiment.



(a) (b)

(c) (d)

(e) (f)

(g) (h)

Fig. 5.48: The overhead image for the fourth trial of two robots aggregated to the blue area using a city road and a Road-Following controller.

5.4.5 Discussion

We presented three prototypes to build our robots for real-world experiments. Our robots depend only on their local sensing to detect the scalar field and the surrounding obstacles without external help. Our Follow-Road controller is executed directly on ATmega32U4 (Zumo's microcontroller). We performed an aggregation task using two robots, and our experiment results show that robots could aggregate using our scalar field and handle collision situations.

Chapter 6

Conclusion and Future work

This chapter summarises the accomplished work to guide simple robots relying only on their local sensing to achieve different tasks like planar construction, finding the largest covering network, aggregation and foraging tasks. Future work directions are presented on how to generate the scalar field automatically.

6.1 Summary

In our work, we guided a swarm of simple robots to achieve various tasks using a scalar field by designing the scalar field for the given task and projecting it onto the environment. The robots detected that scalar field using their local sensing and accomplished that task with scalar field guidance. Also, we showed that the scalar field could be used to help divide labour among the swarm in the shape construction task (Section 3.1). With the presence of the scalar field, the control algorithm for the robots could be simple, like we presented in the shape construction (Section 3.1.2), finding the Largest Coverage Network (Section 4.1), aggregating and foraging tasks (Sections 5.3.1 and 5.3.2). Also, reinforcement learning was presented to find alternative solutions for the Largest Coverage Network task (Section 4.3). In addition to providing a scalar field for the robots,

we investigated different communication techniques among the robots to construct the annulus shape as described in Section 3.1.3. Finally, we presented prototypes (Section 5.4.3) and built robots to perform real-world experiments using a decentralized approach relying on local sensing (Section 5.4.4).

6.1.1 Publications

Our research has led to the following publications:

- Dalia S. Ibrahim, and Andrew Vardy. "Adaptive task allocation for planar construction using response threshold model." Theory and Practice of Natural Computing: 8th International Conference, TPNC 2019, Kingston, ON, Canada, December 9–11, 2019, Proceedings 8. Springer International Publishing, 2019.
- Dalia S. Ibrahim, and Andrew Vardy. "Largest coverage network in a robot swarm using reinforcement learning." Artificial Life and Robotics 27.4 (2022): 652-662.
- Dalia S. Ibrahim, and Andrew Vardy "Swarm in the City: Inspirations from Urban Street Networks for Swarm Robotic Guidance." Submitted.

6.2 Contributions

The primary focus of this thesis is how to guide simple robots that rely only on their local sensing by using the scalar field to achieve the required task. The conducted research formed four main contributions:

- We presented the response threshold function along with the scalar field to guide and help in the division of labour among the swarm of robots while they create an annulus shape. We also incorporated different communication techniques among the swarm.

- We designed the low/high-resolution scalar field with reinforcement learning to find the Largest Coverage Network by the swarm of robots.
- We designed a novel scalar field inspired by a road street network to reduce the spatial interference among robots. Using this scalar field, we presented a simple reactive system used by robots to perform aggregation and foraging tasks.
- We designed and built simple real robots that rely on their local sensing and tested these robots on the aggregation task.

6.3 Future work

6.3.1 Introduction

In this work, we generated the scalar field in a task-dependent manner as we presented in Chapters 3 and 4. For the planar construction task, we projected light onto the center of the environment. Robots detected the intensities of the light as a scalar value to guide them in their movement. Using this scalar field, we guided the robots to construct an annulus shape within certain bounds of light intensities. Also, we helped the robots establish the Largest Coverage Network by finding the least light intensity and establishing a coverage network there. For aggregation and foraging tasks, we generated the scalar field using one of the sample-based methods (RRT*) to attract the robots to the desired area as described in Chapter 5.

For future work, we propose calculating a scalar field in a task-independent manner. The proposed system could take the task description and the configuration space and generate a scalar field for the given task. The scalar field generator monitors the robot's performance and updates the generated scalar field until it reaches the accepted performance.

We could extend this proposed system to handle dynamic task allocation for multi-robot systems. If the required task can be divided into N independent sub-tasks, the proposed system will generate the N scalar fields for these independent sub-tasks. Fig. 6.1 shows the proposed block diagram for the dynamic task allocation using the scalar field to guide the robots. We will feed the text description for the required task to the 'Task Decomposition' function as depicted, which analyzes and decomposes it into independent sub-tasks (N). The 'Task Allocation' function takes information about system constraints such as resources or time constraints, environmental constraints, information about the robots' locations or moving obstacles, and the sub-tasks description. Then, it divides a group of robots into subgroups to perform separate tasks. The 'Task Assignment' function will be responsible for assigning subtasks to the robot given its execution abilities and based on the chosen task assignment algorithm, such as the Random-Choice algorithm, in which robots randomly select the tasks. Another task assignment is the Extreme-Comm Algorithm, in which the robot communicates with all robots in its range and assigns tasks based on its position [127]. Once the robot knows the exact task requirement, the 'Scalar Field Generator' function is invoked to provide a scalar field for this task. It sends the generated scalar field to the control execution, which gives it feedback about the robot's performance. The 'Scalar Field Generator' keeps updating the scalar field until the robots achieve the accepted performance. Then, the robot can take another task using the 'Reassignment' and 'Task Assignment' functions.

This proposed system is also helpful if we have a heterogeneous swarm of robots; the scalar field generator can provide a suitable scalar field for each robot based on its capabilities, which are taken as input, as shown in Fig. 6.1.

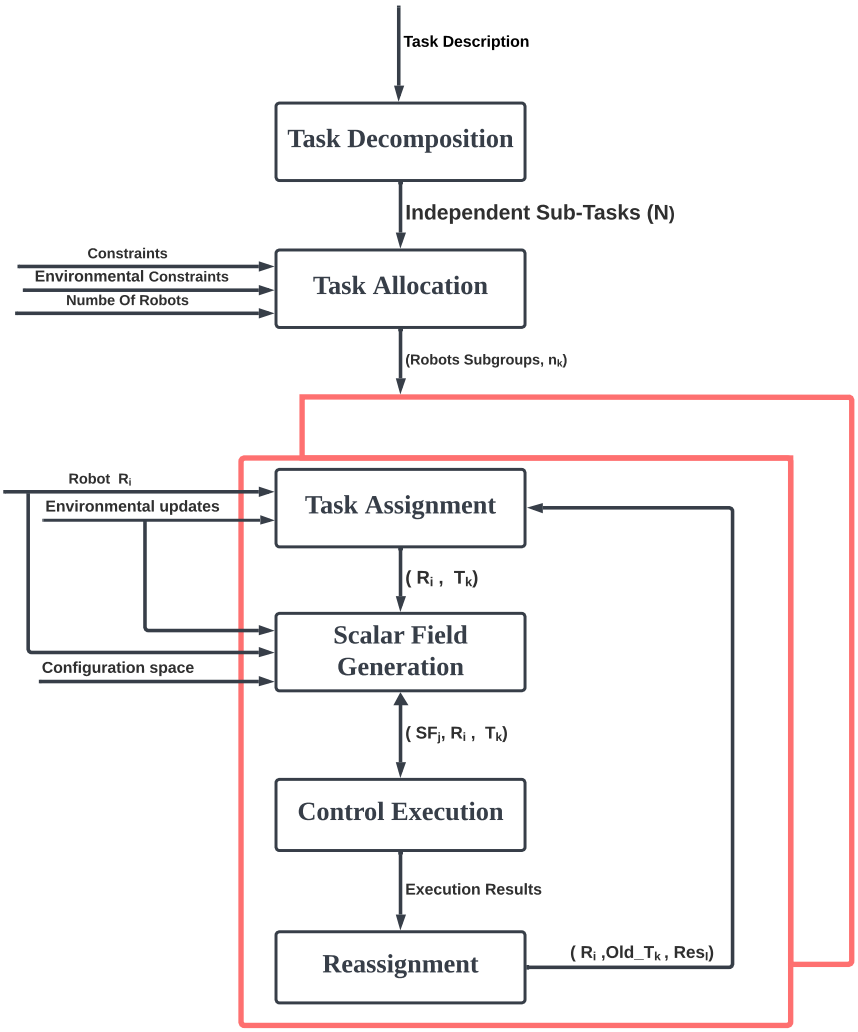


Fig. 6.1: The proposed block diagram for dynamic task allocation using an online scalar field generator to guide a swarm of robots.

6.3.2 Scalar Field Generation:

The previous subsection addressed the overall proposal system to generate a scalar field in a task-independent manner. This subsection will consider the specific sub-task of scalar field generation.

6.3.2.1 Using Deep Reinforcement Learning:

Haung et al. [128], teach a machine how to paint human portraits by decomposing the target image and learning the sequence of the strokes as shown in Fig. 6.2. Their training process used deep reinforcement learning to make a long-term plan to get the correct order and position of the strokes to paint the target image.

Proposed Research Direction

We can adapt the approach of Haung et al. to teach the machine the rules for generating the scalar field. We could start with the primitive vector fields, as shown in Fig. 2.15, and our system will learn the correct order and position for the vector field that enhances the robot's performance to achieve the given task. The generated scalar field will be calculated from different vector fields. The vector fields could also be automatically generated by utilizing artificial intelligence and optimization techniques. One common approach is to employ machine learning algorithms, such as reinforcement learning or evolutionary algorithms, to automatically learn the optimal potential field based on the swarm's sub-objectives and the environment's characteristics.

6.3.2.2 Using Generative Adversarial Networks (GANs):

Elhoseiny et al. [129] aims to generate unseen categories without any training examples of these categories. They started their model with seen categories associated with class description. They try to generate unseen, realistic classes, as shown in Fig. 6.3. They

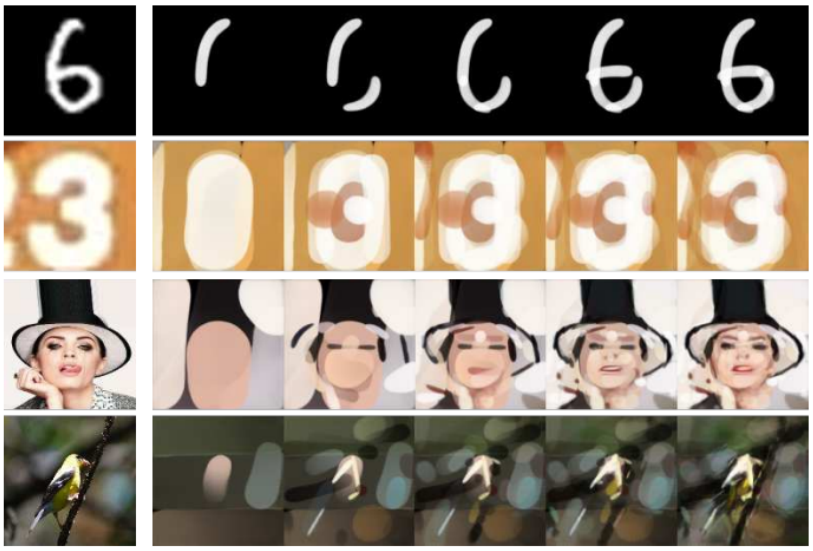


Fig. 6.2: This figure shows the printing process for four images. The target images are placed in the first column [128].

try to deviate from the seen classes and produce novel generations, which should still be realistic. Their distribution exhibits high entropy across existing classes and has variations in the loss function. Fig. 6.4 describes their system. They fed the generator with hallucinated text to trick the discriminator with an unseen image descriptor, and they adjusted the loss function in the discriminator to accept the generated image only if it is an unseen image created from all seen classes but not from one category.

Proposed Research Direction

We can utilize some of the ideas from Elhoseiny et al. to automatically build a database with the labelled vector fields, as mentioned in the previous research direction, using artificial intelligence and optimization techniques such as evolutionary algorithms. The database should be split into well-described categories. Then, we can use Generative Adversarial Networks (GANs) [130] as a scalar field generator for a given task's description. The discriminator can evaluate the generated scalar field by sending it to the swarm robot simulator for feedback.

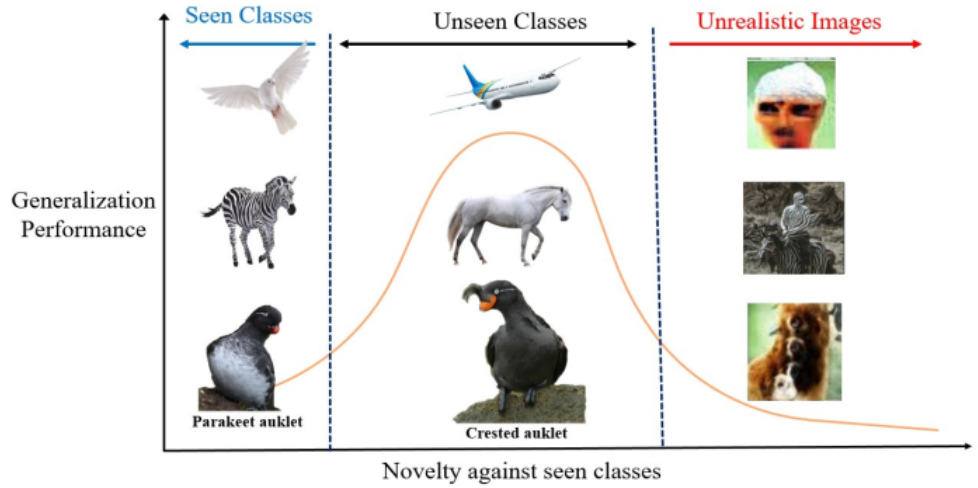


Fig. 6.3: This curve shows the novelty verus seen classes [129] and inspired from Wundt Curve when he discribe the Human Creativity literature [131].

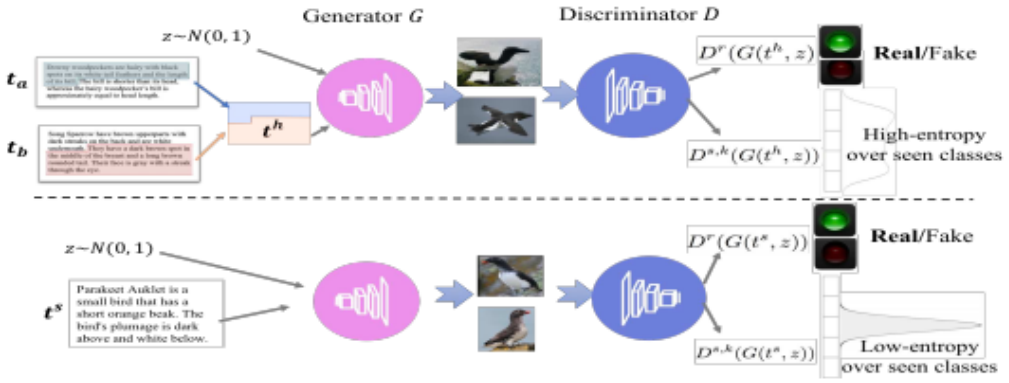


Fig. 6.4: The top figure shows how they fed hallucinated text th with the noise vector to the generator to trick the discriminator to believe that the generated image was real. That helped the generator maximize the entropy over seen classes and deviate from them. Unlike the bottom figure, the generator has the text form seen class ts, so the guarantees images have low entropy over classes [129].

References

[1] Y. Mitaka and T. Akino, "A review of termite pheromones: multifaceted, context-dependent, and rational chemical communications," *Frontiers in Ecology and Evolution*, vol. 8, p. 595614, 2021.

[2] E. Bonabeau, M. Dorigo, and G. Theraulaz, "Self-Organization and Templates: Application to Data Analysis and Graph Partitioning," in *Swarm Intelligence*. Oxford University Press.

[3] J. L. Carvalho, P. C. Farias, and E. F. Simas Filho, "Global Localization of Unmanned Ground Vehicles Using Swarm Intelligence and Evolutionary Algorithms," *Journal of Intelligent & Robotic Systems*, vol. 107, no. 3, p. 45, 2023.

[4] Güzel, Mehmet Serdar and Gezer, Emir Cem and Ajabshir, Vahid Babaei and Bostancı, Erkan, "An adaptive pattern formation approach for swarm robots," in *2017 4th International Conference on Electrical and Electronic Engineering (ICEEE)*, 2017, pp. 194–198.

[5] O. Soysal and E. Sahin, "Probabilistic aggregation strategies in swarm robotic systems," in *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.* IEEE, 2005, pp. 325–332.

[6] H.-X. Wei, Q. Mao, Y. Guan, and Y.-D. Li, "A Centroidal Voronoi Tessellation based Intelligent Control Algorithm for the Self-assembly Path Planning of Swarm Robots," *Expert Systems with Applications*, vol. 85, pp. 261–269, 2017.

[7] M. Schranz, M. Umlauft, M. Sende, and W. Elmenreich, "Swarm Robotic Behaviors and Current Applications," *Frontiers in Robotics and AI*, vol. 7, pp. 36–56, 2020.

[8] S. Oberst, J. C. Lai, R. Martin, B. J. Halkon, M. Saadatfar, and T. A. Evans, "Revisiting stigmergy in light of multi-functional, biogenic, termite structures as communication channel," *Computational and Structural Biotechnology Journal*, vol. 18, pp. 2522–2534, 2020.

[9] M. Salman, D. Garzón Ramos, and M. Birattari, "Automatic design of stigmergy-based behaviours for robot swarms," *Communications Engineering*, vol. 3, no. 1, p. 30, 2024.

[10] S. A. Ocko, A. Heyde, and L. Mahadevan, "Morphogenesis of termite mounds," *Proceedings of the National Academy of Sciences*, vol. 116, no. 9, pp. 3379–3384, 2019.

[11] A. Heyde, L. Guo, C. Jost, G. Theraulaz, and L. Mahadevan, "Self-organized biotectonics of termite nests," *Proceedings of the National Academy of Sciences*, vol. 118, no. 5, p. e2006985118, 2021.

[12] D. Payton, R. Estkowski, and M. Howard, "Pheromone robotics and the logic of virtual pheromones," in *Swarm Robotics: SAB 2004 International Workshop, Santa Monica, CA, USA, July 17, 2004, Revised Selected Papers 1.* Springer, 2005, pp. 45–57.

[13] F. Arvin, T. Krajník, A. E. Turgut, and S. Yue, "Cosϕ: Artificial pheromone system for robotic swarms research," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 407–412.

[14] J. Werfel, K. Petersen, and R. Nagpal, "Distributed multi-robot algorithms for the termes 3d collective construction system," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011.

[15] C. R. Tinoco and G. Oliveira, "Pherocom: decentralised and asynchronous swarm robotics coordination based on virtual pheromone and vibroacoustic communication," *arXiv preprint arXiv:2202.13456*, 2022.

[16] H. Wang and M. Rubenstein, "Shape formation in homogeneous swarms using local task swapping," *IEEE Transactions on Robotics*, vol. 36, no. 3, pp. 597–612, 2020.

[17] J. Hunt and F.-J. Richard, "Intracolony vibroacoustic communication in social insects," *Insectes Sociaux*, vol. 60, pp. 403–417, 2013.

[18] C. Grüter and W. M. Farina, "The honeybee waggle dance: can we follow the steps?" *Trends in ecology & evolution*, vol. 24, no. 5, pp. 242–247, 2009.

[19] R. Zelick, D. A. Mann, and A. N. Popper, "Acoustic communication in fishes and frogs," *Comparative Hearing: Fish and Amphibians*, pp. 363–411, 1999.

[20] D. K. Dechmann and K. Safi, "Studying communication in bats," *Cognition, Brain, Behavior*, vol. 9, no. 3, pp. 479–496, 2005.

[21] P. J. Hart, R. Hall, W. Ray, A. Beck, and J. Zook, "Cicadas impact bird communication in a noisy tropical rainforest," *Behavioral Ecology*, vol. 26, no. 3, pp. 839–842, 2015.

[22] R. R. Murphy, *Introduction to AI robotics*. MIT press, 2019.

[23] W. T., "Collision-Free Path Planning using Potential Field Method For Highly Redundant Manipulators." [Online]. Available: https://taylorwang.wordpress.com/2012/04/06/collision-free-path-planning-using-potential-field-method-for-highly-redundant-manipulators/

[24] "Scalar field." [Online]. Available: https://en.wikipedia.org/wiki/Scalar_field

[25] A. Vardy, "The lasso method for multi-robot foraging," in *2022 19th Conference on Robots and Vision (CRV)*. IEEE, 2022, pp. 106–113.

[26] A. Vardy and D. S. Ibrahim, "A swarm of simple robots constructing planar shapes," *arXiv preprint arXiv:2004.13888*, 2020.

[27] C. Martinez and F. Jimenez, "Implementation of a potential field-based decision-making algorithm on autonomous vehicles for driving in complex environments," *Sensors*, vol. 19, no. 15, pp. 3318–3334, 2019.

[28] R. L. Stewart and R. A. Russell, "Building a loose wall structure with a robotic swarm using a spatio-temporal varying template," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 1. IEEE, 2004, pp. 712–716.

[29] A. Vardy, "Orbital Construction: Swarms of Simple Robots Building Enclosures," in *2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS* W)*. IEEE, 2018, pp. 147–153.

[30] G. Theraulaz, E. Bonabeau, and J. Denuebourg, "Response threshold reinforcements and division of labour in insect societies," *Proceedings of the Royal Society of London. Series B: Biological Sciences*, vol. 265, no. 1393, pp. 327–332, 1998.

[31] E. Bonabeau, D. d. R. D. F. Marco, M. Dorigo, and G. Theraulaz, *Swarm intelligence: from natural to artificial systems*. Oxford university press, 1999, no. 1.

[32] A. Vardy. (2018) Waggle: Visual programming for swarm robotics. [Online]. Available: https://github.com/BOTSlab/waggle

[33] J. Panerati, L. Gianoli, C. Pinciroli, A. Shabah, G. Nicolescu, and G. Beltrame, "From swarms to stars: Task coverage in robot swarms with connectivity constraints," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7674–7681.

[34] G. Francesca, M. Brambilla, A. Brutschy, L. Garattoni, R. Miletitch, G. Podevijn, A. Reina, T. Soleymani, M. Salvaro, and C. Pinciroli, "Automode-chocolate: automatic design of control software for robot swarms," *Swarm Intelligence*, vol. 9, no. 2, pp. 125–152, 2015.

[35] D. Bajaj, "Maximum coverage heuristics (mch) for target coverage problem in wireless sensor network," in *2014 IEEE International Advance Computing Conference (IACC)*. IEEE, 2014, pp. 300–305.

[36] M. Cardei and D.-Z. Du, "Improving wireless sensor network lifetime through power aware organization," *Wireless networks*, vol. 11, no. 3, pp. 333–340, 2005.

[37] A. Vardy, "Orbital construction: Swarms of simple robots building enclosures," in *2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS* W)*. IEEE, 2018, pp. 147–153.

[38] C. Strickland, D. Churchill, and A. Vardy, "A reinforcement learning approach to multi-robot planar construction," in *2019 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*. IEEE, 2019, pp. 238–244.

[39] "Reinforcement learning: Difference between Q and deep Q learning." [Online]. Available: https://www.globaltechcouncil.org/reinforcement-learning/reinforcement-learning-difference-between-q-and-deep-q-learning/

[40] C. Szepesvári, "Algorithms for reinforcement learning," *Synthesis lectures on artificial intelligence and machine learning*, vol. 4, no. 1, pp. 1–103, 2010.

[41] Y. C. Zhang, "Spatial interference reduction for multi-robot systems using rational and team-based aggression," 2006.

[42] K. Bhaskar, A. Mansooor, K. R. Anand, and A. Ramani, "Intelligent mail delivery robot," in *2011 2nd International Conference on Intelligent Agent & Multi-Agent Systems*. IEEE, 2011, pp. 1–6.

[43] N. Pinkam, F. Bonnet, and N. Y. Chong, "Robot collaboration in warehouse," in *2016 16th International Conference on Control, Automation and Systems (ICCAS)*. IEEE, 2016, pp. 269–272.

[44] S. Brown, M. Zuluaga, Y. Zhang, and R. Vaughan, "Rational aggressive behaviour reduces interference in a mobile robot team," in *ICAR'05. Proceedings., 12th International Conference on Advanced Robotics, 2005.* IEEE, 2005, pp. 741–748.

[45] T. Carlson and Y. Demiris, "Collaborative control for a robotic wheelchair: evaluation of performance, attention, and workload," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 42, no. 3, pp. 876–888, 2012.

[46] M. Skoczeń, M. Ochman, K. Spyra, M. Nikodem, D. Krata, M. Panek, and A. Pawłowski, "Obstacle detection system for agricultural mobile robot application using RGB-D cameras," *Sensors*, vol. 21, no. 16, p. 5292, 2021.

[47] C. Pang, X. Zhong, H. Hu, J. Tian, X. Peng, and J. Zeng, "Adaptive obstacle detection for mobile robots in urban environments using downward-looking 2D LiDAR," *Sensors*, vol. 18, no. 6, p. 1749, 2018.

[48] J. O. Gaya, L. T. Gonçalves, A. C. Duarte, B. Zanchetta, P. Drews, and S. S. Botelho, "Vision-based obstacle avoidance using deep learning," in *2016 XIII Latin American Robotics Symposium and IV Brazilian Robotics Symposium (LARS/SBR)*. IEEE, 2016, pp. 7–12.

[49] X. Xue, Z. Li, D. Zhang, and Y. Yan, "A deep reinforcement learning method for mobile robot collision avoidance based on double dqn," in *2019 IEEE 28th International Symposium on Industrial Electronics (ISIE)*. IEEE, 2019, pp. 2131–2136.

[50] P. Wang, S. Gao, L. Li, B. Sun, and S. Cheng, "Obstacle avoidance path planning design for autonomous driving vehicles based on an improved artificial potential field algorithm," *Energies*, vol. 12, no. 12, p. 2342, 2019.

[51] A. N. A. Rafai, N. Adzhar, and N. I. Jaini, "A review on path planning and obstacle avoidance algorithms for autonomous mobile robots," *Journal of Robotics*, vol. 9, 2022.

[52] S. Lin, A. Liu, J. Wang, and X. Kong, "A review of path-planning approaches for multiple mobile robots," *Machines*, vol. 10, no. 9, p. 773, 2022.

[53] L. Shao, H. Liu, C. Chen, D. Du, J. Li, and H. Liu, "Path planning for mobile robots based on improved RRT algorithm," in *2020 IEEE International Conference on Mechatronics and Automation (ICMA)*. IEEE, 2020, pp. 1240–1244.

[54] R. Elvik, "To what extent can theory account for the findings of road safety evaluation studies?" *Accident Analysis & Prevention*, vol. 36, no. 5, pp. 841–849, 2004.

[55] J. McLurkin and J. Smith, "Distributed algorithms for dispersion in indoor environments using a swarm of autonomous mobile robots," in *Distributed autonomous robotic systems 6*. Springer, 2007, pp. 399–408.

[56] M. S. Couceiro, R. P. Rocha, and N. M. Ferreira, "A novel multi-robot exploration approach based on particle swarm optimization algorithms," in *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*. IEEE, 2011, pp. 327–332.

[57] S. Sharma, A. Shukla, and R. Tiwari, "Multi robot area exploration using nature inspired algorithm," *Biologically Inspired Cognitive Architectures*, vol. 18, pp. 80–94, 2016.

[58] X. Huang, F. Arvin, C. West, S. Watson, and B. Lennox, "Exploration in extreme environments with swarm robotic system," in *2019 IEEE International Conference on Mechatronics (ICM)*, vol. 1. IEEE, 2019, pp. 193–198.

[59] C. R. Tinoco, D. A. Lima, and G. M. Oliveira, "An improved model for swarm robotics in surveillance based on cellular automata and repulsive pheromone with discrete diffusion," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 34, no. 1, pp. 53–77, 2019.

[60] M. Masar, "A biologically inspired swarm robot coordination algorithm for exploration and surveillance," in *2013 IEEE 17th International Conference on Intelligent Engineering Systems (INES)*. IEEE, 2013, pp. 271–275.

[61] G. Saldamli and A. Razavi, "Surveillance missions deployment on the edge by combining swarm robotics and blockchain," in *2020 Fourth International Conference on Multimedia Computing, Networking and Applications (MCNA)*. IEEE, 2020, pp. 106–112.

[62] J. L. Farrugia and S. G. Fabri, "Swarm robotics for object transportation," in *2018 UKACC 12th International Conference on Control (CONTROL)*. IEEE, 2018, pp. 353–358.

[63] M. Jurt, E. Milner, M. Sooriyabandara, and S. Hauert, "Collective transport of arbitrarily shaped objects using robot swarms," *Artificial Life and Robotics*, vol. 27, no. 2, pp. 365–372, 2022.

[64] Z. Zhou, Z. Hou, and Y. Pei, "Reconfigurable particle swarm robotics powered by acoustic vibration tweezer," *Soft Robotics*, vol. 8, no. 6, pp. 735–743, 2021.

[65] J. L. Farrugia and S. G. Fabri, "Swarm robotics for object transportation," in *2018 UKACC 12th International Conference on Control (CONTROL)*. IEEE, 2018, pp. 353–358.

[66] L. Pitonakova, R. Crowder, and S. Bullock, "Information flow principles for plasticity in foraging robot swarms," *Swarm Intelligence*, vol. 10, pp. 33–63, 2016.

[67] J. Wilson and S. Hauert, "Information transport in communication limited swarms," *Artificial Life and Robotics*, vol. 27, no. 4, pp. 632–639, 2022.

[68] S. Bullock, R. Crowder, and L. Pitonakova, "Task allocation in foraging robot swarms: The role of information sharing," in *Artificial Life Conference Proceedings*. MIT Press, 2016, pp. 306–313.

[69] Y. Nishida, K. Kaneko, S. Sharma, and K. Sakurai, "Suppressing chain size of blockchain-based information sharing for swarm robotic systems," in *2018 Sixth In-*

ternational Symposium on Computing and Networking Workshops (CANDARW). IEEE, 2018, pp. 524–528.

[70] M. Machaiah and S. Akshay, "Iot based human search and rescue robot using swarm robotics," *Int. J. Eng. Adv. Technol*, vol. 8, no. 5, pp. 1797–1801, 2019.

[71] G. A. Cardona and J. M. Calderon, "Robot swarm navigation and victim detection using rendezvous consensus in search and rescue operations," *Applied Sciences*, vol. 9, no. 8, p. 1702, 2019.

[72] Y. Li, Y. Gao, S. Yang, and Q. Quan, "Swarm robotics search and rescue: A bee-inspired swarm cooperation approach without information exchange," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 1127–1133.

[73] D. Paez, J. P. Romero, B. Noriega, G. A. Cardona, and J. M. Calderon, "Distributed particle swarm optimization for multi-robot system in search and rescue operations," *IFAC-PapersOnLine*, vol. 54, no. 4, pp. 1–6, 2021.

[74] J. P. Queralta, J. Taipalmaa, B. C. Pullinen, V. K. Sarker, T. N. Gia, H. Tenhunen, M. Gabbouj, J. Raitoharju, and T. Westerlund, "Collaborative multi-robot systems for search and rescue: Coordination and perception," *arXiv preprint arXiv:2008.12610*, 2020.

[75] I. Rausch, Y. Khaluf, and P. Simoens, "Scale-free features in collective robot foraging," *Applied Sciences*, vol. 9, no. 13, p. 2667, 2019.

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

[77] L. Pitonakova, R. Crowder, and S. Bullock, "The information-cost-reward framework for understanding robot swarm foraging," *Swarm Intelligence*, vol. 12, pp. 71–96, 2018.

[78] L. Schenk, *Designing cities: basics, principles, projects.* Birkhäuser, 2023.

[79] A. Yaseen, N. Jawad, and W. O. Khel, "A study of the evolution and physical traits of two competing street patterns: Narrow meandering and wide gridiron," *Journal of Arts & Social Sciences*, vol. 9, no. 1, pp. 100–110, 2022.

[80] M. Smets, *Foundations of Urban Design.* Actar Publishers, 2022.

[81] S. Gondet and C. Benech, "Application of the space syntax to the study of city planning from Syrian late bronze age circular cities," *ArcheoSciences. Revue d'archéométrie*, no. 33 (suppl.), pp. 217–219, 2009.

[82] M. A. Abdeen, M. H. Ahmed, H. Seliem, T. R. Sheltami, T. M. Alghamdi, and M. El-Nainay, "A novel smart ambulance system—algorithm design, modeling, and performance analysis," *IEEE Access*, vol. 10, pp. 42 656–42 672, 2022.

[83] Google. (n.d.) Google maps for Madinah in Saudi Arabia . [Online]. Available: https://maps.app.goo.gl/3CVRoGum28jF9VaV6

[84] G. Boeing, "Urban spatial order: Street network orientation, configuration, and entropy," *Applied Network Science*, vol. 4, no. 1, pp. 1–19, 2019.

[85] M. Heyen. (2012) Understanding the chicago grid: How to never be lost in chicago. [Online]. Available: https://terribuseman.wordpress.com/tag/map/

[86] P. Bindzar, J. Saderova, M. Sofranko, P. Kacmary, J. Brodny, and M. Tutak, "A case study: Simulation traffic model as a tool to assess one-way vs. two-way traffic

on urban roads around the city center," *Applied Sciences*, vol. 11, no. 11, p. 5018, 2021.

[87] One-way/two-way street conversions. [Online]. Available: https://safety.fhwa.dot.gov/saferjourney1/Library/countermeasures/13.htm

[88] H. Karimi, B. Ghadirifaraz, S. N. Shetab Boushehri, S.-M. Hosseininasab, and N. Rafiei, "Reducing traffic congestion and increasing sustainability in special urban areas through one-way traffic reconfiguration," *Transportation*, pp. 1–24, 2021.

[89] W. Riggs and J. I. Gilderbloom, "How multi-lane, one-way street design shapes neighbourhood life: collisions, crime and community," *Local Environment*, vol. 22, no. 8, pp. 917–933, 2017.

[90] ——, "How multi-lane, one-way street design shapes neighbourhood life: collisions, crime and community," *Local Environment*, vol. 22, no. 8, pp. 917–933, 2017.

[91] I. Noreen, A. Khan, and Z. Habib, "Optimal path planning using RRT* based approaches: a survey and future directions," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 11, 2016.

[92] ——, "A comparison of RRT, RRT* and RRT*-smart path planning algorithms," *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 16, no. 10, p. 20, 2016.

[93] K. Naderi, J. Rajamäki, and P. Hämäläinen, "RT-RRT* a real-time path planning algorithm based on RRT," in *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games*, 2015, pp. 113–118.

[94] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[95] J. Li and Y. Tan, "A probabilistic finite state machine based strategy for multi-target search using swarm robotics," *Applied Soft Computing*, vol. 77, pp. 467–483, 2019.

[96] O. Mısır, L. Gökrem, and M. S. Can, "Fuzzy-based self organizing aggregation method for swarm robots," *Biosystems*, vol. 196, p. 104187, 2020.

[97] O. Misir and L. Gökrem, "Dynamic interactive self organizing aggregation method in swarm robots," *Biosystems*, vol. 207, p. 104451, 2021.

[98] Z. Firat, E. Ferrante, Y. Gillet, and E. Tuci, "On self-organised aggregation dynamics in swarms of robots with informed robots," *Neural Computing and Applications*, vol. 32, pp. 13 825–13 841, 2020.

[99] J. D. Hasbach and M. Bennewitz, "The design of self-organizing human–swarm intelligence," *Adaptive Behavior*, vol. 30, no. 4, pp. 361–386, 2022.

[100] A. Sadeghi Amjadi, M. Raoufi, and A. E. Turgut, "A self-adaptive landmark-based aggregation method for robot swarms," *Adaptive Behavior*, vol. 30, no. 3, pp. 223–236, 2022.

[101] F. Arvin, S. C. Doraisamy, K. Samsudin, F. A. Ahmad, and A. R. Ramli, "Implementation of a cue-based aggregation with a swarm robotic system," in *Knowledge Technology: Third Knowledge Technology Week, KTW 2011. Revised Selected Papers*. Springer, 2012, pp. 113–122.

[102] F. Arvin, A. E. Turgut, F. Bazyari, K. B. Arikan, N. Bellotto, and S. Yue, "Cue-based aggregation with a mobile robot swarm: a novel fuzzy-based method," *Adaptive Behavior*, vol. 22, no. 3, pp. 189–206, 2014.

[103] F. Arvin, A. E. Turgut, T. Krajník, and S. Yue, "Investigation of cue-based aggregation in static and dynamic environments with a mobile robot swarm," *Adaptive Behavior*, vol. 24, no. 2, pp. 102–118, 2016.

[104] L. König, S. Mostaghim, and H. Schmeck, "Decentralized evolution of robotic behavior using finite state machines," *International Journal of Intelligent Computing and Cybernetics*, vol. 2, no. 4, pp. 695–723, 2009.

[105] W. Li, A. Miyazawa, P. Ribeiro, A. Cavalcanti, J. Woodcock, and J. Timmis, "From formalised state machines to implementations of robotic controllers," in *Distributed Autonomous Robotic Systems: the 13th International Symposium*. Springer, 2018, pp. 517–529.

[106] W. Liu, A. F. Winfield, and J. Sa, "Modelling swarm robotic systems: A case study in collective foraging," *Towards autonomous robotic systems (TAROS 07)*, vol. 23, pp. 25–32, 2007.

[107] J. Li and Y. Tan, "A probabilistic finite state machine based strategy for multi-target search using swarm robotics," *Applied Soft Computing*, vol. 77, pp. 467–483, 2019.

[108] B. Pang, Y. Song, C. Zhang, H. Wang, and R. Yang, "A swarm robotic exploration strategy based on an improved random walk method," *Journal of Robotics*, vol. 2019, pp. 1–9, 2019.

[109] C. Dimidov, G. Oriolo, and V. Trianni, "Random walks in swarm robotics: an experiment with kilobots," in *International conference on swarm intelligence*. Springer, 2016, pp. 185–196.

[110] D. W. Sims, N. E. Humphries, R. W. Bradford, and B. D. Bruce, "Lévy flight and Brownian search patterns of a free-ranging predator reflect different prey field characteristics," *Journal of Animal Ecology*, vol. 81, no. 2, pp. 432–442, 2012.

[111] A. A. Munishkin, D. Milutinović, and D. W. Casbeer, "Safe navigation with the collision avoidance of a Brownian motion obstacle," in *Dynamic Systems and Control Conference*, vol. 58295. American Society of Mechanical Engineers, 2017, p. V003T39A009.

[112] A. Deshpande, M. Kumar, and S. Ramakrishnan, "Robot swarm for efficient area coverage inspired by ant foraging: the case of adaptive switching between Brownian motion and lévy flight," in *Dynamic Systems and Control Conference*, vol. 58288. American Society of Mechanical Engineers, 2017, p. V002T14A009.

[113] J. Nauta, S. Van Havermaet, P. Simoens, and Y. Khaluf, "Enhanced foraging in robot swarms using collective lévy walks," in *24th European Conference on Artificial Intelligence (ECAI)*, vol. 325. IOS, 2020, pp. 171–178.

[114] Y. Katada, A. Nishiguchi, K. Moriwaki, and R. Watakabe, "Swarm robotic network using Lèvy flight in target detection problem," *Artificial Life and Robotics*, vol. 21, pp. 295–301, 2016.

[115] V. Fioriti, F. Fratichini, and S. Chiesa, "Lèvy flight search of moving objects," *Bio-inspired Robotics*, 2014.

[116] A. Deshpande, M. Kumar, and S. Ramakrishnan, "Robot swarm for efficient area coverage inspired by ant foraging: the case of adaptive switching between Brownian motion and Lèvy flight," in *Dynamic Systems and Control Conference*, vol. 58288. American Society of Mechanical Engineers, 2017, p. V002T14A009.

[117] A. Mubayi, C. Kribs, V. Arunachalam, and C. Castillo-Chavez, "Studying complexity and risk through stochastic population dynamics: Persistence, resonance, and extinction in ecosystems," in *Handbook of Statistics*. Elsevier, 2019, vol. 40, pp. 157–193.

[118] Ozobot. [Online]. Available: https://ozobot.com/

[119] Pololu zumo 32u4 robot. [Online]. Available: https://www.pololu.com/docs/0J63

[120] Zumo 32u4 front sensor array. [Online]. Available: https://www.pololu.com/docs/0J63/3.5#:~:text=The%20Zumo%2032U4%20Front%20Sensor,microcontroller%20at%20any%20given%20time.

[121] RGB color sensor with ir filter and white led - tcs34725. [Online]. Available: https://www.adafruit.com/product/1334

[122] Adafruit APDS9960 Proximity, Light, RGB, and Gesture Sensor - STEMMA QT / Qwiic. [Online]. Available: https://www.adafruit.com/product/3595

[123] Adafruit PCA9548 8-Channel STEMMA QT. [Online]. Available: https://learn.adafruit.com/adafruit-pca9548-8-channel-stemma-qt-qwiic-i2c-multiplexer/arduino

[124] Adafruit 16-channel PCA9685. [Online]. Available: https://www.adafruit.com/product/815

[125] [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

[126] Arduino CLI (command line interface). [Online]. Available: https://www.arduino.cc/pro/software-pro-cli/

[127] J. McLurkin and D. Yamins, "Dynamic task assignment in robot swarms." in *Robotics: Science and Systems*, vol. 8, no. 2005. Cambridge, USA, 2005.

[128] Z. Huang, S. Zhou, and W. Heng, "Learning to paint with model-based deep reinforcement learning," pp. 8709–8718, 2019.

[129] M. Elhoseiny and M. Elfeki, "Creativity inspired zero-shot learning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 5784–5793.

[130] D. Saxena and J. Cao, "Generative adversarial networks (gans) challenges, solutions, and future directions," *ACM Computing Surveys (CSUR)*, vol. 54, no. 3, pp. 1–42, 2021.

[131] C. Martindale, *The Clockwork Muse: The Predictability of Artistic Change.* Basic Books, 1990.