Carleton University

M.C.S Project (COMP 5903)

Investigating Transportation Efficiency in Swarm Robotics with High-Fidelity Simulation

Submitted by
Nirav C. Pansuriya

Under the supervision of
Dr. Mark Lanthier

May 3, 2023

Abstract

This project aims to investigate the efficiency of the transportation process in swarm robotics by discovering, improving, and sharing efficient paths. The project presents a simulation with conditions as realistic as possible within the Webots to perform experiments. In this project, a base algorithm for path discovery, optimization, and communication is adopted, which has been previously implemented and evaluated in non-realistic simulations. While the base algorithm gave promising results in non-realistic environments, it was not designed to handle realistic conditions effectively. So in this project, the base algorithm was modified to function in an environment that simulates conditions close to reality. The algorithm was modified by taking into specific cases, such as ensuring robust communication, collision detection and avoidance. The modified algorithm was evaluated by exploring three communication modes to investigate how to maximize the efficiency of the transportation process.

Acknowledgment

I would like to express my deepest appreciation to my project supervisor, Dr. Mark Lanthier, for his invaluable guidance and support throughout the course of this project. His expertise, constructive feedback, and encouragement have greatly contributed to the completion of this work.

Dr. Lanthier's passion for the subject matter and dedication to his students' success has been truly inspiring, and I am grateful for the opportunity to learn from him. I would also like to extend my gratitude to the faculty and staff at Carleton University for providing an enriching academic environment that has fostered my growth and development.

Finally, I would like to thank my family and friends for their unwavering support, understanding, and encouragement throughout this journey. Their belief in me has been a constant source of motivation and strength.

Contents

List of Figures

1 Introduction

Swarm robotics is a rapidly evolving field that explores the collaborative behaviour of multiple robots working together to accomplish tasks. One of the applications of swarm robotics is the efficient transportation of objects within a given environment, which requires effective pathfinding and optimization strategies. Various algorithms have been developed to find the shortest path between two points or nodes in a graph, such as A* and Dijkstra's Shortest Path First algorithm. However, many of these algorithms require knowledge of the exact location of the destination, which may not always be available in real-world scenarios.

In response to this challenge, the algorithm introduced in the project named "Path Finding with Swarm Robotics" [8]. It introduced a distributed algorithm for path discovery, optimization, and communication among swarm robots. The objective was to find and optimize transportation pathways in unknown environments without prior knowledge of the destination location. By collaborating and sharing optimal pathways, the swarm of robots can efficiently deliver equipment or resources from one location to another. The use of a swarm of robots allows for the discovery of many possible pathways, increasing the likelihood of finding the most optimal one.

As that algorithm shows promising results, the algorithm was implemented with simulation in Python. That simulation was non-realistic and was more like a mathematical model. As the algorithm was showing good results in non-realistic simulation, it might face many issues and need modification when applied in the simulation with nearly realistic conditions. The motivation and main goal of this project are to investigate the transportation efficiency in Swarm robotics in simulation which has conditions very near to reality. For that, the algorithm used in the project [1] has been used as a base algorithm. In this project, that algorithm has been improved to apply in the simulator with nearly realistic conditions. I will outline the necessary modifications, improvements, and refinements required to make the algorithm compatible with high-fidelity simulations and discuss the challenges and complexities associated with this process.

In the end, the project will culminate in the evaluation of the improved algorithm's performance in the WeBots simulation, using three different communication models to assess its efficiency and versatility. I will present the results of these tests, analyze their implications for swarm robotics transportation efficiency, and discuss potential future developments and improvements that could further enhance the algorithm's performance.

2 Environment

2.1 Simulator

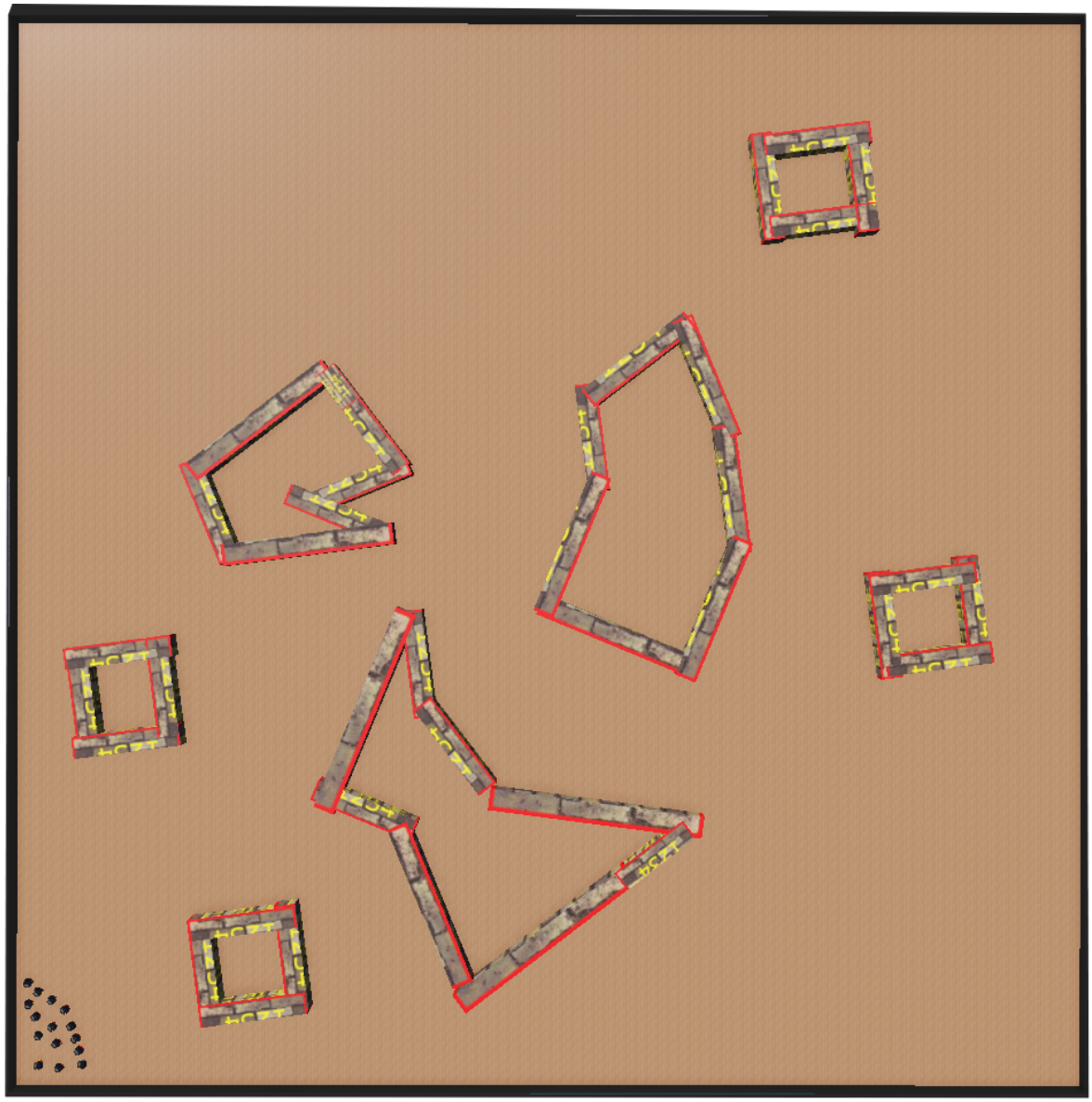


Figure 1: WeBots Environment

In this research project, WeBots [1] was used as a simulation software platform to conduct an experiment. WeBots is a widely used and open-source simulation software that provides researchers with the ability to design and program virtual robots and their systems. The software offers a user-friendly interface, and a variety of robots, sensors, and devices, making it easier to create complex robotics systems and simulate them in various environments.

Additionally, WeBots supports multiple programming languages such as C++, Python, and Java, enabling users with different programming backgrounds to utilize their preferred language. Overall, WeBots is an excellent tool to design and simulate robotic systems, testing algorithms, and gain valuable experience in robotics development. In this project, we were attempting to produce a simulation with conditions as realistic as possible within the WeBots.

2.2 Simulation Time

In Webots, a timestamp is a numerical value representing a specific point in the simulation time. This simulation time is crucial for synchronizing events, controlling robot behaviour, and measuring the duration of processes in the virtual environment. Webots' simulation time advances according to the basicTimeStep parameter, which defines the time interval between each simulation step in milliseconds. By utilizing timestamps, developers can coordinate actions among multiple robots, schedule events, and accurately execute tasks at precise moments in the simulation. It is important to note that the simulation time in Webots may differ from the real-world time due to factors such as computational power and simulation complexity.

In Webots, a timestamp is a numerical value representing a specific point in the simulation time. This simulation time is crucial for synchronizing events, controlling robot behaviour, and measuring the duration of processes in the virtual environment. Webots' simulation time advances according to the basicTimeStep parameter, which defines the time interval between each simulation step in milliseconds. By utilizing timestamps, developers can coordinate actions among multiple robots, schedule events, and accurately execute tasks at precise moments in the simulation. It is important to note that the simulation time in Webots may differ from the real-world time due to factors such as computational power and simulation complexity.

2.3 Robot

For the simulation, the Pioneer 3-DX robot [6] has been used. The Pioneer 3-DX is a versatile mobile robot suitable for a variety of research and application areas such as mapping, teleoperation, localization, monitoring, reconnaissance, and other behaviours. Its omnidirectional base allows for movement in any direction, making it particularly useful in confined spaces. Moreover, the robot can be customized and equipped with various sensors, tools, and attachments to cater to specific research requirements. The Pioneer 3-DX has proven to be a reliable and versatile platform for research and educational purposes in the robotics community.

The robots used in the simulation have a length of 485 mm, a width of 381 mm, and a height of 217 mm, with a weight of 9 kg. The maximum forward/backward speed of the robots is 1.6 m/s. In addition, the robots are equipped with 16 distance sensors, an emitter and receiver for communication, GPS, and a compass for accurate path following and orientation maintenance. Further details about the sensors are discussed in the subsequent sub-sections.

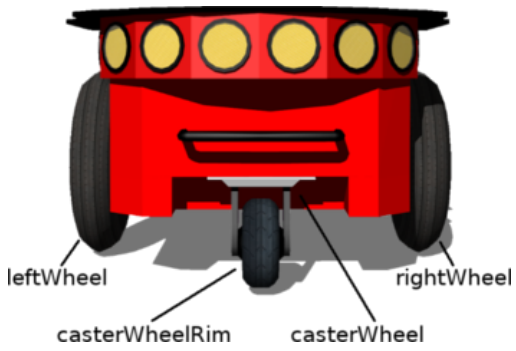


Figure 2: Pioneer 3-DX Robot image obtained from the Cyberbotics website [6]

2.4 Distance Sensors

In this project, sonar distance sensors [3] are used for detecting obstacles and determining the distance between a robot and the nearest object. The sonar sensors function by emitting high-frequency sound waves, which bounce back after hitting an object. The time taken for the sound wave to return is measured, from which the distance to the object can be computed.

In the simulation, each robot was equipped with 16 sonar distance sensors, with each sensor containing 8 rays, as shown in Figure 3. Only the front distance sensors were used in the simulation, as the back distance sensors were not deemed necessary.

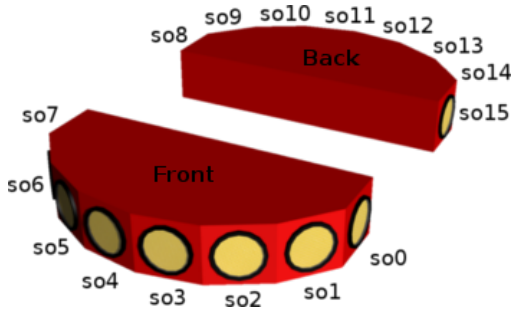


Figure 3: Distance Sensors in Pioneer 3-DX Robot image obtained from the Cyberbotics website [6]

2.5 GPS and Compass Sensors

In the proposed algorithm, the robot must retain information about its path and follow it accordingly. To accomplish this, latitude and longitude coordinates are used. A GPS sensor [5] is necessary to obtain these coordinates. Furthermore, as the robot transitions from one position to another, it must make specific turns based on the coordinates of both locations. To measure these turns accurately, the robot is equipped with a compass sensor [2].

2.6 Receiver and Emitter

In the proposed algorithm, robots communicate with one another using emitters [4] and receivers[7]. These devices enable message transmission and reception between the robots. Radio-type emitters and receivers are employed in this application, offering a defined range of communication.

Emitters and receivers can operate on different channels, including a specific broadcast channel for disseminating messages to all receivers tuned to that channel. Robots can also configure their emitters and receivers to communicate on specific channels, allowing for selective communication between robots with matching channels.

Receivers are equipped with a queue that stores incoming messages. This allows the robot to access and process the received information at its own pace, ensuring efficient communication and data processing.

3 Algorithm

3.1 Overview

It is important to mention that in this project, the algorithm is not developed from scratch. Instead, it is adapted from an existing project named "Pathfinding with Swarm Robotics" [8] and further refined to be applied in a simulation that replicates realistic conditions. The following section provides an overview of the algorithm, which is executed individually by each robot at every time step.

The main goal of the robots is to transfer items from source to destination. Initially, the robots do not have a path to the destination and must explore the environment to locate it. Each robot departs from the starting point and heads in randomly generated trajectories, exploring the area in a pseudo-random manner. The robots store the path they traverse by recording the coordinates they visit. The robots are equipped with sonar distance sensors, which function as their eyes, monitoring the environment while in motion and measuring the

distance to nearby objects. If the sensors detect an obstacle, the robots avoid it.

During the journey, the robots actively search for the destination. In this project, the destination coordinates are predetermined, and when a robot reaches a near to destination with a defined threshold (4 meters), it heads towards the destination. In the real world, the destination could be marked by Bluetooth, beacons, or stickers with specific colours that the robots can detect using different sensors like cameras or radio signal receivers.

When a robot successfully establishes a path from the starting point to the destination, it switches from exploration mode to transportation, path-refinement, and information-sharing mode. The robot no longer explores the environment pseudo-randomly in search of the destination. Instead, it leverages the path it has created, to transfer items from source to destination. During transportation, the robot shares its path information with other robots that it encounters.

The transportation part involves the robots moving between the source and the destination. Once a robot locates the destination, it travels towards the source. Upon arrival at the source, the robot picks up an item and begins moving towards the destination using the established path. Upon reaching the destination, the robot delivers the equipment and returns to the start to pick up another item. This process repeats until all equipment has been transported from the source to the destination. It is important to note that in this project, item detection at the source and the physical act of picking up and dropping off items at the destination has not been implemented. The robot only simulates transportation between the source and destination.

During the transportation process, the robot continually refines its path to ensure the most efficient delivery of items. As the robot initially found its path through a randomly exploring environment, it tends to be overly long and requires optimization to increase efficiency. Path-refining involves the robot identifying sections of its path that are in its range of vision and attempting to make its connecting path as short as possible. By straightening out any path sections that are not as short as possible, the robot can remove unnecessary curves and loops. The process of path-smoothing can be observed in action in the attached figure 6. In the first picture, the blue exploration path (generated through random movements) is shown alongside the pink optimized path in its early stage. The optimized path has already removed unnecessary loops and reduced curves. The second picture shows the optimized path (pink) becoming shorter and straight until it becomes the shortest path possible. The third image displays the final optimized path.

As the robots have access to path information and are optimizing their paths, this information is highly valuable. As a result, robots can now exchange this knowledge with one another

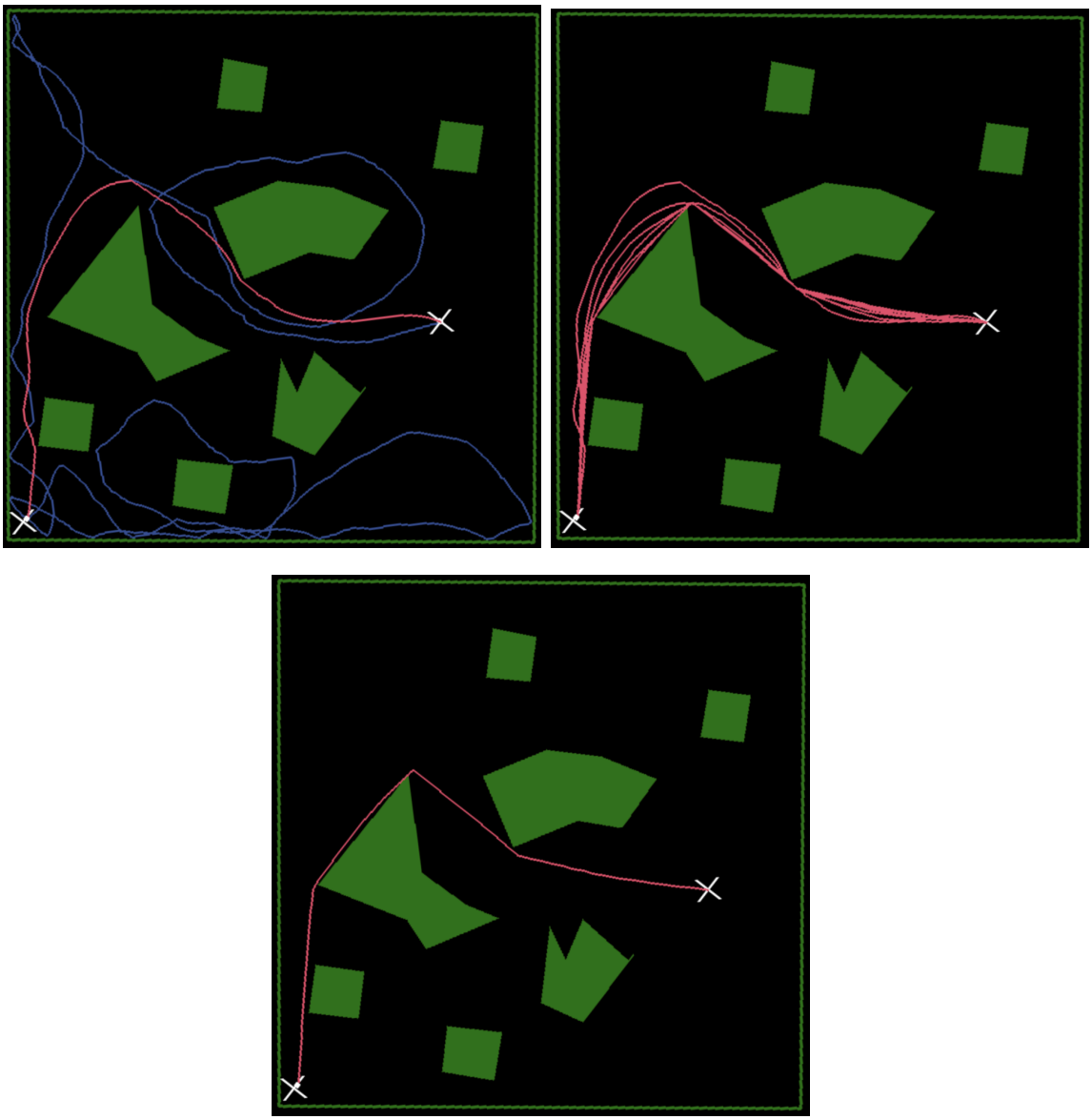


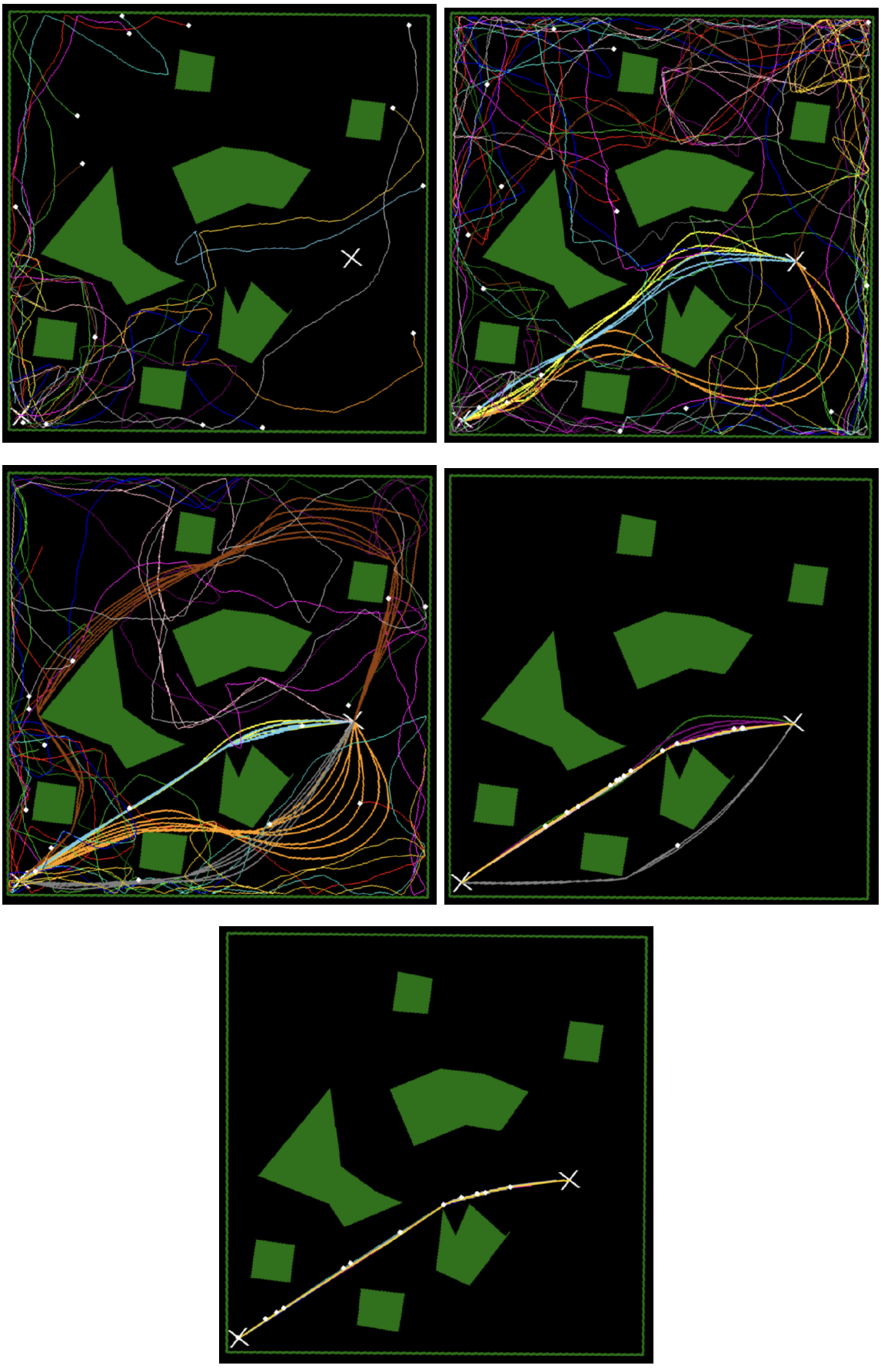
Figure 4: Path Smoothing

to leverage each other's knowledge and identify more efficient paths. To allow sufficient time for each robot to explore its environment and develop its own path, communication between robots is restricted initially for a given amount of time. During the travelling, when two robots come within communication range, it will try to establish a connection with that bot. If communication will be successful, they will exchange path information. If neither robot is aware of the destination, no information exchange will occur. If one robot knows the destination and the other does not, the robot without the destination will adapt to the other robot's path. This robot will then begin transporting items, refining its path, and will share it with other robots. Should two robots that already have successful paths communicate, then they will compare their paths and improve each other's paths from start to finish wherever possible. For example, if one robot has an efficient first half of its path but an inefficient second half and a second robot has an inefficient first half but an efficient second half, both robots will compare their paths to establish a more efficient combined path.

Algorithm 1 Modified Main Algorithm

```
// This algorithm will execute in parallel and separately for each individual robot.
Broadcast WAIT message
if Robot does not have path to the destination then
    if Robot can reach to the destination then
        Move to destination
    else if Robot can see the destination then
        Move toward the destination
    else
        Generate new location to explore
    end if
else
    Move to next location in the path
    ApplyPathSmoothing()
    Receive a message from the receiver queue
    if Robot has a communication partner then
        Establish the communication
        Exchange path information
    end if
end if
```

The overview of the algorithm has shown in the algorithm 1. Every robot will first explore the environment, then find the destination, refine the path and will exchange the path information with other robots. As mentioned before, this algorithm is taken from the project [8] as a base algorithm. While the base algorithm showed promising results, the issue was that a Python environment lacked realistic conditions, such as obstacle detection, robot collision, robot traffic, usage of sensors and communication in a chaotic environment with many robots



13
Figure 5: Visualization of the algorithm

attempting to communicate simultaneously. In this project, the algorithm was enhanced to enable implementation in a simulation environment that closely approximates real-world conditions. The overall flow of the algorithm is the same in the base algorithm and this project. C++ language has been used to write the algorithm in WeBots.

To understand the algorithm more easily, look at the image 5. This image shows the movement of the robots over a certain period of time. In this example, 15 robots are tasked with transferring 100 items from the bottom left cross to the right middle cross. Obstacles are represented in green. As the algorithm runs in the simulation, it also generates logs, and these images are generated from that log data.

The first sub-image shows that, initially, the robots are moving randomly while avoiding obstacles. At this point, none of them have found the destination. In the second sub-image, some robots have found their destination and are now transporting items. Here, you can see that their paths are slowly becoming shorter. Since communication is not enabled yet, other robots are still searching for the path. The third sub-image shows that more robots have discovered their destination, and they are optimizing their paths. At this stage, communication has been enabled. The fourth sub-image reveals that all robots have found the path, thanks to the enabled communication. The robot with the orange path has chosen a much more efficient path (the middle path) by communicating with other bots. Finally, in the fifth sub-image, you can see that all robots have adopted the most efficient path. Now, each robot has an optimized path, allowing them to transport items more efficiently.

The implementation of the base algorithm is briefly discussed here as it requires for the next section. In the base algorithm, the environment that the robots work in was represented by a 3-dimensional grid of square cells. To simulate the movement of robots, a unique colour was assigned to each robot, and the cells on the robot's path were coloured with the same colour. This allows for identifying the path taken by each robot as consecutive cells with the same colour. It was like a mathematical model. The implementation did not involve the use of any simulated robots or sensors. In the following subsection, we will discuss the detailed algorithm with appropriate modifications and the reasoning behind these changes.

3.2 Detailed Algorithm

This section provides a detailed discussion of every part of the main algorithm. The first step in the algorithm for the robot is to find the destination by exploring the environment randomly. The algorithm 2 simply gives the new random location to move. This algorithm loops until the new location is not produced. This involves generating a new direction close to the current heading of the robot and checking if it is possible to go in that direction. If

Algorithm 2 GenerateNewLocationToExplore()

```
1: while true do
2:     generate a new direction based on robot's current direction
3:     use sensors to get environment information for heading in new direction
4:     if no impassable terrain between robot and potential new location then
5:         move to new location
6:         break out of loop
7:     else
8:         allow robot to choose from a wider range of directions
9:     end if
10: end while
```

there is an obstacle in the way, the robot discards that direction and continues looping until it finds a new location to move.

In this project, a similar approach to the base algorithm was adopted, for finding a new location. But the implementation of the algorithm required some changes. In the base algorithm, the coordinates of the obstacle or non-passable terrain were coloured on the grid. In simple words, the coordinates of the obstacles were statically defined, while that was not the case here. A Sonar distance sensor was used to identify any impassable terrain or obstacles 1. The base algorithm calculated the coordinates of the next location cell to move based on the randomly generated direction and coloured it with the robot's unique colour. In the Webots implementation, a Compass sensor was used to calculate the robot's turns so that it can turn towards the correct direction. Additionally, the GPS sensor was used to get the coordinates of the robot's current location and identify when the robot reaches the next location.

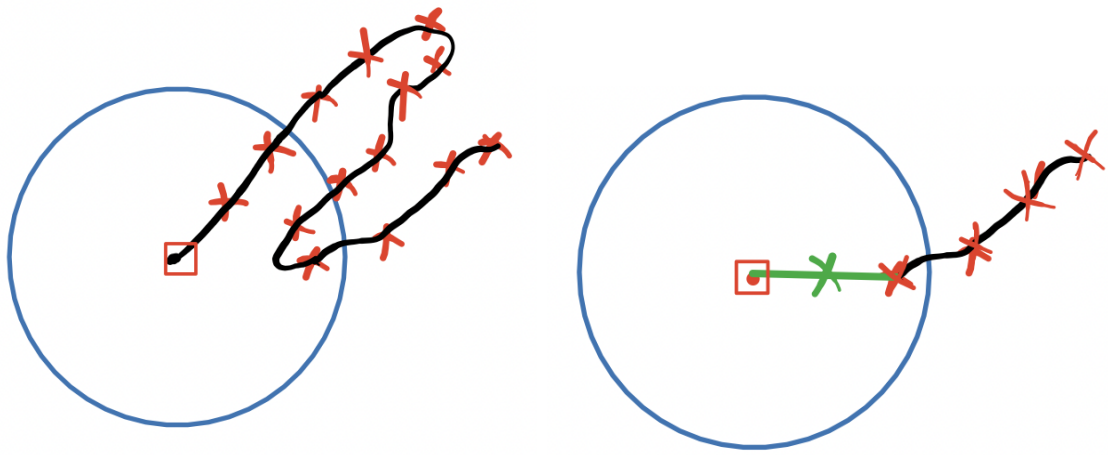


Figure 6: Path Smoothing Explaination

The path smoothing algorithm adopted in this project is derived from the base algorithm with

Algorithm 4 Modified ApplyPathSmoothing()

```
1: candidates ← empty list
2: if Robot is moving from source to destination then
3:     for each location in path[current path index to end of the path] do
4:         if location is within the radius of botVisionRedius and location is reachable then
5:             candidates.add(location)
6:         end if
7:     end for
8: else
9:     for each location in path[start to current path index] do
10:        if location is within the radius of botVisionRedius and location is reachable then
11:            candidates.add(location)
12:        end if
13:    end for
14: end if
15: if candidates are not empty then
16:    Find the candidate point that provides the best shortcut
17:    Generate new locations between current location and selected candidate
18:    Replace old locations between current location and candidate with new locations
19: else
20:    Return
21: end if
```

appropriate changes. The base algorithm determines the perimeter coordinates of a circle with a given radius and a bot at its centre. The algorithm then identifies points that are both on the perimeter and part of the robot's path, examines each such point, and determines if it is a valid location that can be reached from the current position of the robot. If so, the point is added to the candidate points, and the robot selects the best candidate that provides the shortest path and adjusts its path accordingly.

Implementing this algorithm in the Webots simulation presented two challenges. Firstly, unlike the base algorithm that uses coloured cells to track the path, the path in Webots is composed of GPS coordinates. To identify the candidate points, a linear search of the path was necessary. Additionally, as the path in Webots is composed of coordinates on given equal distances, some path coordinates might not be precisely on the perimeter of the circle. To address this, all points on the path that is under the construction side of the robot's vision are identified as potential candidates. To determine if the candidate is reachable, the robot moves towards the candidate, and its distance sensors are used to determine if the candidate's location is reachable. The robot then selects the best candidate that provides the shortest path and generates coordinates on a straight line between itself and the candidate. The path locations between the robot and the candidate are then replaced with the newly generated

coordinates. Algorithm 4 is the modified algorithm.

Refer to figure 4 for a better understanding of the process. The left side of the image shows the original path of the robot. The black line represents the path, and the crosses denote the locations on the path. Once the robot identifies the best candidate in terms of path shortcut, it generates a new location on the straight line between itself and the candidate, replaces all path locations between it and the candidate with the newly updated ones, and adjusts its path accordingly.

In the path-following strategy, robots utilize a combination of emitter, receiver, and sonar distance sensors to effectively distinguish between stationary obstacles and other robots in their path. Upon sensing an obstacle, the robot checks its receiver queue for any "WAIT" messages from nearby robots. If any "WAIT" messages are present, the robot assumes the obstacle is another robot and takes measures to avoid a collision. Both robots involved then perform a slight left turn and move forward for a short distance before attempting to resume their original paths. This approach ensures that robots can navigate efficiently and safely within the environment while avoiding collisions with both stationary obstacles and their robotic peers.

Algorithm 5 Modified CanRobotCommunicate()

```
1: if own mode is explore and partner mode is explore then
2:     Return False
3: else if Partner ID is not in table then
4:     Return True
5: else
6:     if commHistory[partnerId] > THRESHOLD then
7:         Return True
8:     else
9:         Return False
10:    end if
11: end if
```

Effective communication is a crucial aspect of the algorithm. In the base algorithm, the location and presence of other robots could be easily detected as the 3D grid used. The RobotsCanCommunicate() function in the base algorithm determines if two given robots can communicate with each other based on their distance and the non-passable terrain between them. In every cycle, each robot runs the above function for every other robot apart from itself to decide which robots are there to communicate.

In the Webots implementation, however, the robots must determine each other's presence using an emitter and receiver sensor. Every robot has an emitter and receiver on a broadcast channel, allowing them to communicate with each other when they are within close proximity

(2m). Every robot has assigned the unique id the WeBots. As outlined in Algorithm 1, each robot broadcasts a WAIT message at each time step to signal its presence. The WAIT message takes the form of $WAIT\_ < robot\_id > \_ < robot\_mode >$. In this project, the RobotsCanCommunicate method was replaced with the WAIT message. As indicated in Algorithm 1, each robot reads the message from its receiver at each time step. Once a robot receives the WAIT message, it knows that another robot is nearby and can identify its ID and mode. This information is critical in determining whether to communicate with the other robot or not.

Once a robot has identified the robots near it, the next task is to determine with whom it should communicate. The goal is to prevent frequent communication with the same robot within a short period. Robots should communicate once, exchange useful information if possible, and then wait for a certain period before communicating again with the same bot. The most recent communication is given a more negative priority, and vice versa. If a robot encounters another robot for the first time, it is given the highest priority. In the base algorithm, this was archived with priorities. Every bot assigns communication priority to other bots. If multiple robots are near each other, the robot decides whom to communicate with based on priority.

To determine whether to communicate with a given robot, a similar method as the base algorithm is used in this project. A commHistory table is maintained that includes the ID of the other robot and the timestamp of the last communication. Any two robots are not allowed to communicate with each other, for a specific amount of time after a successful path exchange. When a robot receives the WAIT message, it checks whether it should communicate with that particular robot or not. However, unlike the base algorithm, the robot does not search for the best robot to communicate with if there are multiple options. This is because processing too many WAIT messages is time-consuming, especially when too many bots are near each other and broadcasting WAIT messages. Since any two bots cannot communicate for a specific period of time after a successful path exchange, this simulates a priority mechanism. As a result, each time a robot receives a WAIT message and finds another robot nearby, it will prioritize communicating with the new robot over previously communicated ones. The process is implemented in Algorithm 5. After every successful communication, the commHistory table will be updated.

Robots are now capable of detecting when another robot is in close proximity and can decide whether to engage in communication. The base algorithm did not include a communication protocol, and paths were updated by programmatically accessing the object of the partner bot. In a WeBots simulation, however, a communication protocol is essential for ensuring robust communication between robots. During the project development, several communica-

Algorithm 6 EstablishCommuniction()

```
1: if Receiver queue is not empty then
2:     Discard all messages until "WAIT" message or queue is empty
3:     message ← get message
4:     Get partnerId and partnerMode from message and assign them to variables
5:     if CanRobotCommunicate(partnerId, mode) == true then
6:         Stop robot
7:         isPrimary ← (this robot's ID > partnerId)
8:         if isPrimary then
9:             Send "ACCEPT" message every 1 second, until receiving a response from the
   partner or reaching the THRESHOLD number of attempts
10:            if "REJECT" message received then
11:                HandleRejection()
12:            else if "ACCEPTACK" message received then
13:                Compute common channel ID as the concatenation of the smaller and larger
   robot IDs
14:                Tune emitter and receiver on the common channel ID
15:            end if
16:        else
17:            Wait for partner's message or for THRESHOLD seconds, whichever is first
18:            if "REJECT" message received then
19:                HandleRejection()
20:            else if "ACCEPT" message received then
21:                Send "ACCEPTACK" message
22:                Compute common channel ID as the concatenation of the smaller and larger
   robot IDs
23:                Tune emitter and receiver on the common channel ID
24:            end if
25:        end if
26:    else
27:        Send "REJECT" message
28:    end if
29: end if
```

tion issues were encountered, such as robots waiting for signals due to assumptions about the partner bot having received the signal, while it was not the case. And unnecessary messages from other robots while data exchanging caused delays. These challenges led to increased communication times and negatively impacted overall transportation efficiency.

In designing the communication protocol, the aforementioned issues were considered. The protocol is divided into two parts. First, both robots must agree to communicate. Second, once they agree, they will tune their emitters and receivers to a unique channel accessible only to those two robots. The primary-secondary communication protocol is employed, wherein one robot acts as the primary (major) and the other as the secondary (minor). When a robot receives a wait message, it also obtains the ID and mode of its partner. Based on the [algorithm], the robot decides whether to communicate or not. If either robot does not wish to communicate, it will send a reject message in the form $REJECT\_ < robot\_id > \_ < partner\_id >$. Upon receiving the REJECT message, the other robot will continue without waiting.

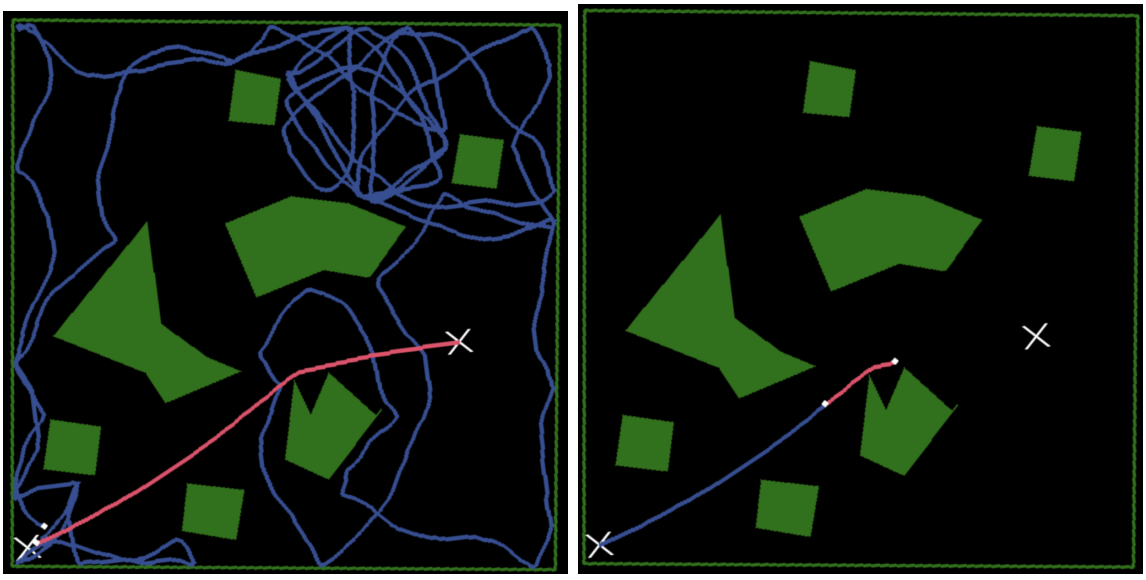
If a robot wishes to communicate, it will determine whether it is primary or secondary (the robot with the higher ID will be primary). Both robots will have each other's IDs, and their roles will be mutually decided. If a robot is a primary, it will send an ACCEPT message in the form $ACCEPT\_ < robot\_id > \_ < partner\_id >$. The secondary robot will wait for the message from its partner, verifying the message using robot_id and partner_id in the message. The primary robot will continue to send ACCEPT messages at a specific frequency until it receives an ACCEPTACK from the secondary robot or until a defined number of attempts have been made. Once the secondary robot receives the ACCEPT message, it will send an ACCEPTACK. The secondary robot will also wait for the ACCEPT message for a predetermined amount of time before moving on, assuming that, due to certain conditions, the other robot was unable to transmit the data. When both robots complete this process, they are mutually agreeing to communicate with each other.

Upon agreeing to communicate, they will generate a channel ID for communication. The channel ID will be a concatenation of the secondary robot's ID and the primary robot's ID. Since both robots have each other's IDs, they will generate identical channel IDs. They will then tune their emitters and receivers to that channel ID, ensuring no other robots can interfere while they exchange data. Both robots will exchange paths and, finally, send an END message to each other. Once a robot has sent its path (if it has) and received the END message, it will close the connection by tuning its emitter and receiver to broadcast mode and proceed. To avoid the infinite waiting time, each robot has a timer for communication. Once the timer will be zero, the robot will close the connection and will continue its journey. The use of a dedicated communication channel allows robots to communicate efficiently

even when surrounded by multiple robots. The acknowledgements enable robots to initiate communication robustly.

Algorithm 7 Path Sharing Algorithm

```
1: if both robots have a path to the destination then
2:     intersections ← all points where robot paths crossed
3:     for each section of robot paths separated by an intersection do
4:         find which robot has the shorter section
5:         replace the other robot's longer section with the shorter section
6:     end for
7: else if only one robot has a path to the destination then
8:     helper ← robot with path to destination
9:     helpee ← robot without path to destination
10:    helpee receives the path from helper
11:    find the appropriate path index based on Helpee's current location
12:    move toward the source
13: end if
```



(a) Both bots came in the contact

(b) Bot with blue path adopted the path from the bot with pink path

Figure 7: Bot in explore mode adopting path from another bot

In this project, the same algorithm 3.2 as the base algorithm is used for path exchanging. Once the robots have completed path exchanging, they will adapt their paths accordingly. If a robot was in explore mode and received a path, it will update its current path index based on its location. The current path index indicates the position in the path list where the robot is presently located. With the updated path index, the robot will begin moving toward the source to retrieve the item using the received path. This is shown in figure 7.

If a robot already had a path, it will compare its path with the received one. The robot will create a list of intersection coordinates between the two paths. For each intersection, the robot will calculate the distance between $intersection_i$ and $intersection_(i + 1)$ in both paths, using location coordinates and Euclidean distance, and adopt the shorter path segment between those two intersections. In this manner, the robot will generate an updated, more efficient path.

4 Experiment

In order to evaluate the transportation efficiency with Swarm Robotics, experiments were conducted using the WeBots platform. The primary goal of these experiments was to determine the time required for 15 robots to transfer 100 items between two locations in a 50 m x 50 m environment. The environment setup is shown in the image, with the bottom left cross representing the source and the right middle cross as the destination. The robots have a communication range of 2 meters and a path-smoothing vision radius of 6 meters. They store path locations every 1 meter. The simulation was run in three different communication modes to assess the impact of communication on transportation efficiency. The simulation was stopped once the combined total trips from source to destination and vice versa reached 100.

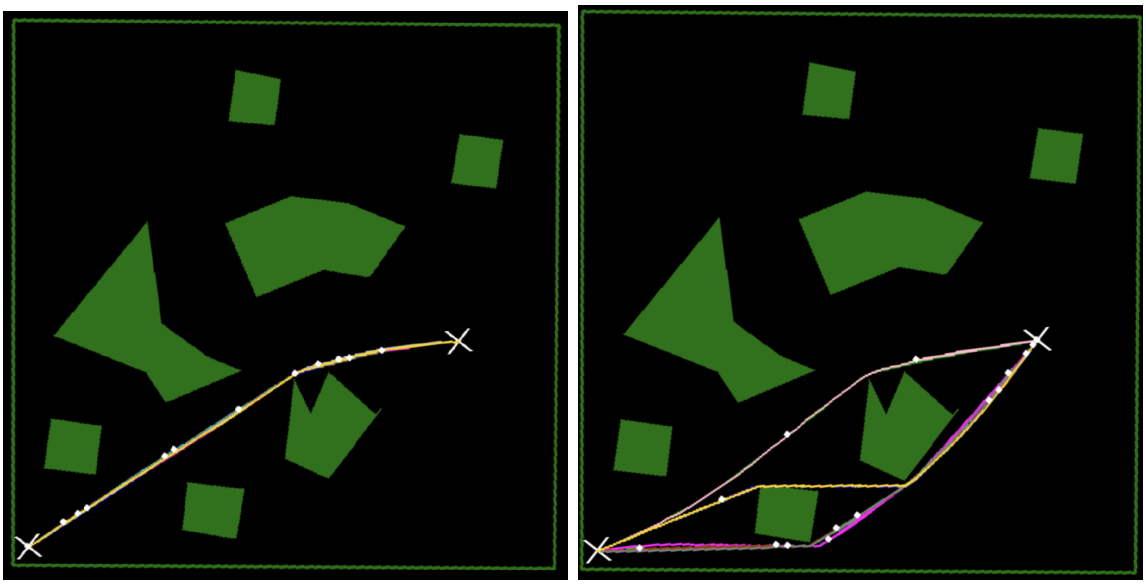
Initially, robots were permitted to communicate any number of times but were restricted from doing so during the first 15 minutes, allowing each robot a fair chance to explore the environment. After this period, they could communicate with any number of robots they desired. However, two robots could not communicate with each other again for the next 10 minutes after completing a communication.

In the second mode, robots were only allowed to communicate once. If a robot did not have a path, its partner would share a path if available. Once a robot received or found a path, it would only provide its path to others and would not receive any more paths. In this mode, communication also began after 15 minutes, with a 10-minute restriction between communications.

Finally, robots were not allowed to communicate at all. Each robot had to find the destination independently, optimize its path, and transport items.

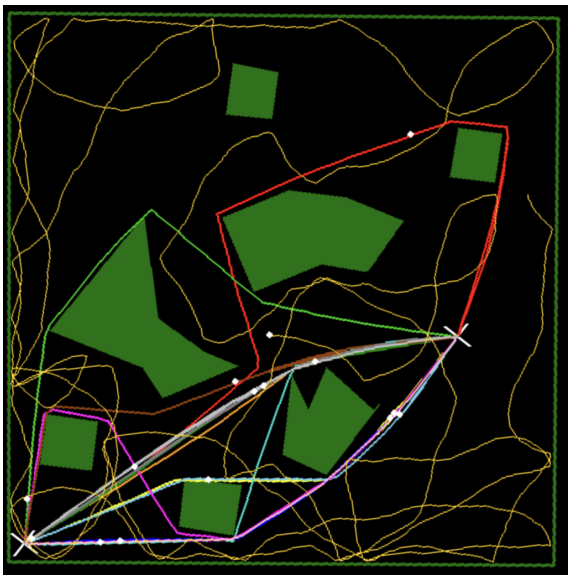
For each mode, the following metrics were recorded: the total number of robots that found their own path, the total number of robots that received a path from others, the distribution of items transferred by each robot, the average path length of robots, and the average time taken by each robot to transfer one item from source to destination.

5 Results



(a) Full Communication Mode

(b) Partial Communication Mode



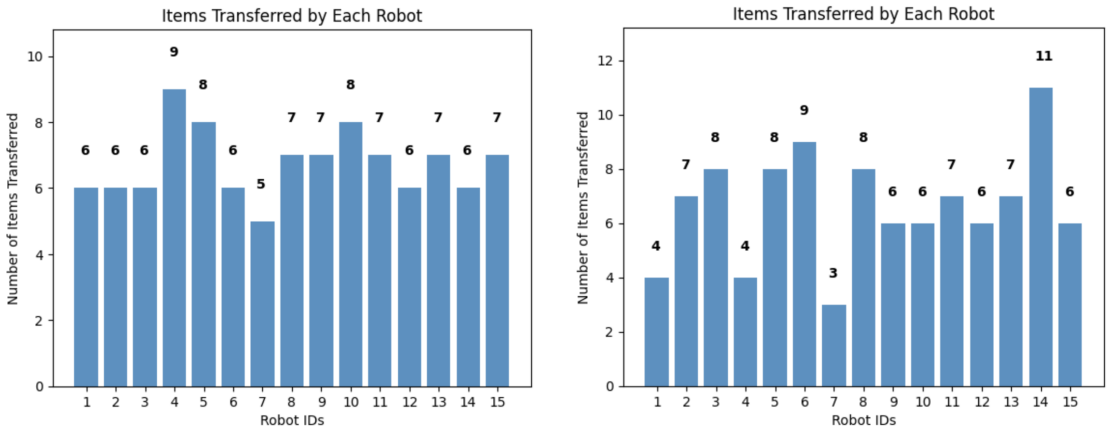
(c) No Communication Mode

Figure 8: Final result of different modes

The image 8 displays the paths of the bots at the end of the simulation. In the full communication mode, it is evident that every bot follows the same path, which is the shortest path, by the end of the simulation. In the partial communication mode, there are three major paths visible. Out of the 15 bots, 9 bots found their own paths, and the remaining 6 bots borrowed paths from those 9 bots. Nevertheless, they all optimized their paths, resulting in what appears to be only three distinct paths. In the no communication mode, the bot with

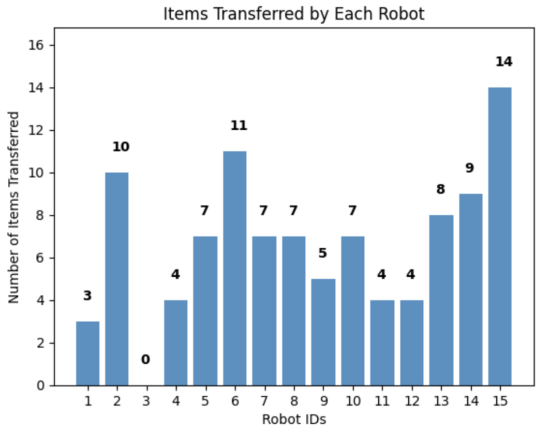
the yellow path did not find the path until the end, while the other 14 bots successfully found their paths. In the third sub-image, there are seven major paths observable. The number of paths is higher in this mode because each robot found its own path, resulting in a larger variety of different paths compared to the other modes.

5.1 Work distribution



(a) Full Communication Mode

(b) Partial Communication Mode



(c) No Communication Mode

Figure 9: Items transferred by each bots

In this section, we discuss the impact of communication mode on workload distribution among the robots. Figure 1 illustrates the number of items carried by each robot under different communication modes.

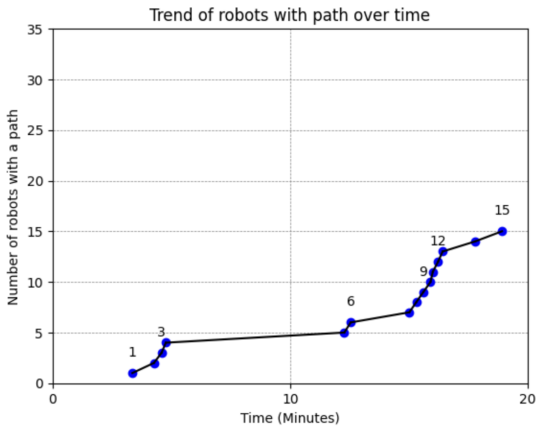
In the full communication mode, it is evident that each robot carried a nearly equal number of items. This is because robots without a path could borrow paths from other robots, enabling them to acquire a path early in the simulation. As a result, all robots spent approximately

the same amount of time transporting items.

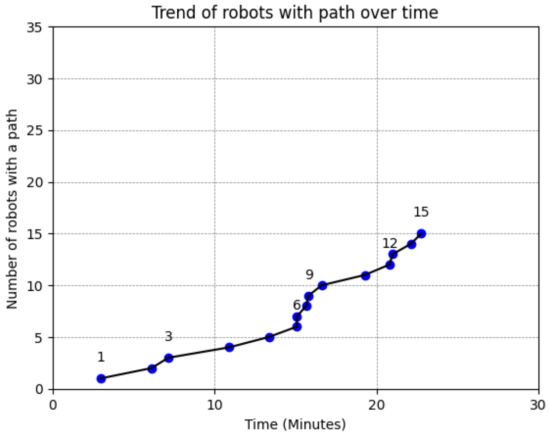
In the partial communication mode, the distribution of workload is somewhat uneven. Some robots only transported 3 or 4 items, while robot 14 carried 11 items. This can be attributed to the fact that some robots obtained lengthy paths since they only communicated once, making it challenging for them to acquire efficient paths.

In the no-communication mode, the workload distribution was significantly imbalanced. Robot 3 was unable to find a path, while robot 15 had to transport 14 items. This is because robots in the no-communication mode spent a considerable amount of time searching for the destination, which affected their ability to evenly distribute the workload.

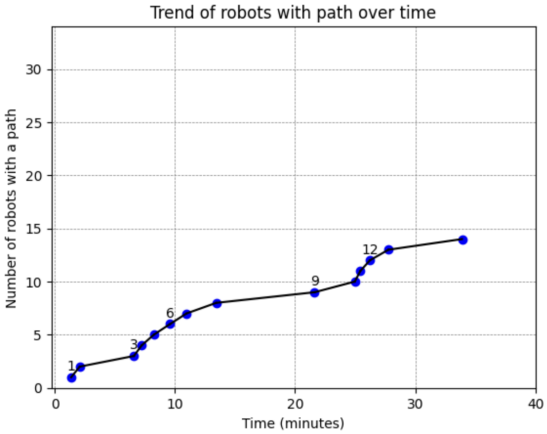
5.2 Trend of robots with a path over time



(a) Full Communication Mode



(b) Partial Communication Mode



(c) No Communication Mode

Figure 10: Trend of robots with a path over time

In the full communication mode, as seen in figure 10, approximately six robots reached the

destination before communication started at the 15-minute mark. Once communication was enabled, in just three minutes, seven more robots acquired the path to the destination. This rapid progress is due to the robots' ability to communicate an unlimited number of times. By around 19 minutes, all 15 robots had obtained a path.

In the partial communication mode, about six robots found their path before communication was enabled. After that point, the remaining robots acquired paths at a slower pace compared to the full communication mode.

In the no communication mode, it took around 34 minutes for 14 robots to find their path, with one robot still unable to locate its path. Thus, we can observe that path exchange provides a significant advantage for the robots, allowing them to begin transferring items more quickly.

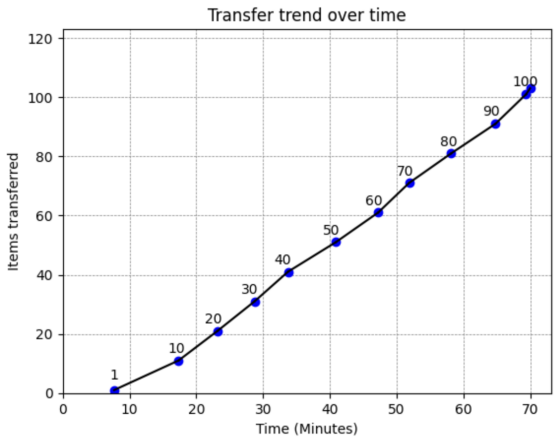
5.3 Time to transfer all items

Figure 11 illustrates the item transfer trend over time. One can observe that in the full communication mode, items are transferred at a steady rate. In contrast, in both the partial communication mode and the no communication mode, the initial item transfer rate is low. However, after a certain amount of time, the transfer rate becomes faster. This is because, in these two modes, the robots take some time to obtain their destination paths. Here, both the no-communication and partial-communication modes took around 51 minutes to transfer 100 items, while the full-communication mode took approximately 70 minutes to complete the transportation.

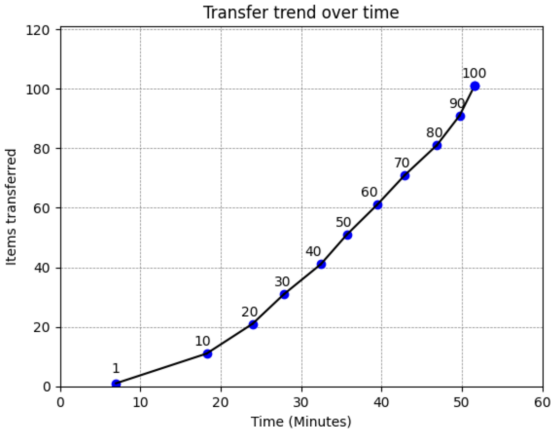
Given the time taken for each robot to find a path and the number of items transferred by each robot, one might easily assume that the full communication mode would be the most efficient. Surprisingly, the results show the exact opposite. The reason behind this unexpected outcome will be explained in the following subsection.

5.4 Reason behind low efficiency of full-communication mode

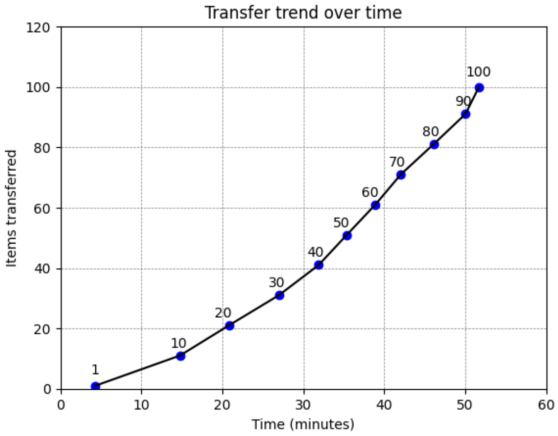
Discussing The average time for transporting a single item, in the no-communication and partial-communication modes, it's approximately 5 minutes and 15 seconds. In contrast, the time taken in the full communication mode is 7 minutes and 14 seconds for transferring a single item. This time accounts for the entire round trip, going from the source to the destination and back. While the images show that the robots have the shortest path in the full-communication mode, the question remains: why is this happening? There are primarily two reasons behind this this.



(a) Full Communication Mode



(b) Partial Communication Mode



(c) No Communication Mode

Figure 11: Items transfer trend over time

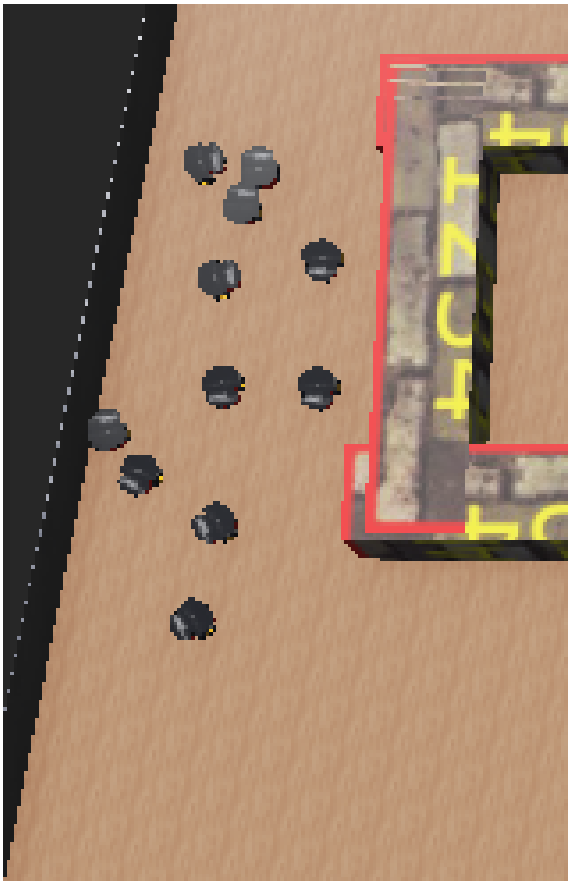


Figure 12: Traffic issue in full-communication mode

1.) Traffic: In the full-communication mode, after some time, all the robots will obtain the most efficient path and move along the same route. As a result, every robot faces traffic issues. The bots take a significant amount of time to navigate through the traffic congestion. Figure 12 illustrates the traffic conditions.

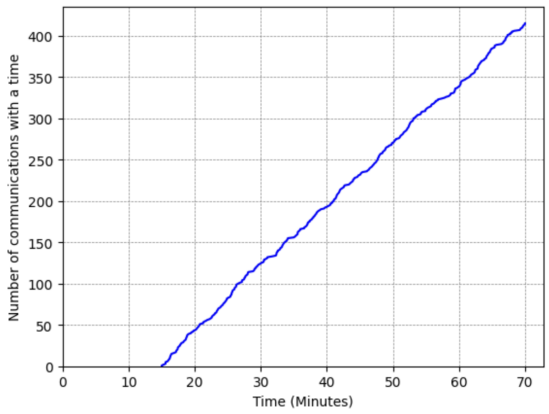


Figure 13: Number of communications over time

2.) Communication: The average communication time for the robots is around 1.7 seconds (including failed communication attempts). As shown in the figure 13, there were approximately 410 communications during the simulation. Thus, the robots also spend a considerable amount of time in communication.

In conclusion, the efficiency of the full-communication mode is reduced due to the robots spending a significant amount of time in traffic and communication.

6 Future Work

The successful implementation of the base algorithm in the WeBots environment has opened up new avenues for further exploration and improvement. This section will discuss some of the challenges that remain to be addressed in future work.

One challenge lies in the use of sonar distance sensors, which have inherent limitations, such as the issue of sonar rays not returning when outside the reflection cone of aperture 45 degrees. This can result in the robot mistakenly assuming that no obstacles are present. A potential solution involves exploring alternative distance sensing technologies, such as infrared (IR) or LIDAR sensors, to enhance obstacle detection accuracy.

Another area for future work is the optimization of critical parameters that affect transportation efficiency. While the present project has not extensively tested and evaluated various

parameter settings, a systematic approach to discovering the most effective parameter configurations could yield further improvements in transportation efficiency.

Lastly, the current algorithm in full communication mode may lead to traffic congestion issues, as most robots rapidly adopt the most efficient path found during exploration. To address this, future work could focus on developing a more robust and effective traffic management algorithm that minimizes congestion and optimizes robot movement. This would ultimately contribute to enhancing the overall performance of swarm robotics in transportation tasks.

7 Conclusion

In this project, we have successfully implemented the base algorithm for path finding, improving, and sharing in WeBots with appropriate modifications. We have evaluated the efficiency of transportation in the environment using three communication modes. The results indicate that while path exchange through communication provides an early head start, a higher number of communications leads to traffic issues and communication delays. These problems can be addressed in two ways: either by limiting the number of communications or by implementing more efficient traffic management or communication algorithms. By exploring these solutions, we can potentially enhance the overall transportation efficiency in swarm robotics systems.

References

[1] Cyberbotics: Robotics simulation with Webots. Available at: https://cyberbotics.com/.

[2] Webots: Compass. Available at: https://cyberbotics.com/doc/reference/compass.

[3] Webots: Distance sensors. Available at: https://cyberbotics.com/doc/reference/distancesensor/.

[4] Webots: Emitter. Available at: https://www.cyberbotics.com/doc/reference/emitter.

[5] Webots: Gps. Available at: https://cyberbotics.com/doc/reference/gps/.

[6] Webots: Pioneer 3-dx robot. Available at: https://cyberbotics.com/doc/guide/pioneer-3dx/.

[7] Webots: Receiver. Available at: https://cyberbotics.com/doc/reference/receiver.

[8] Lukas McClelland. Pathfinding with swarm robotics, April 2020.