Summary

Autonomous swarm exploration is an important problem, having applications in space exploration & search and rescue, among others. Being able to coordinate groups of robots to explore unknown areas with unknown layouts and obstacles in an efficient manner is difficult, and requires novel solutions in both algorithm and communication design. Many algorithms using various strategies, including information gain, distance based, spiraling and backtracking, and animal heuristics have been attempted. We attempt a new direction, using human heuristics, that being the intuitive and natural way a human might chose to explore an area.

Similarly to Spiraling and Selected Backtracking (SSB), one of our algorithms, named Minotaur, spirals around the edge of rooms, and then continues to explore that same room until every part of it has been seen. As it explored around the edges of a room, it marks the doorways it finds, or what constitutes for doorways in environments that are not entirely room-like, and notes these doorways as being unexplored. On finishing the exploration of the room it is in, the robot will then move to the nearest doorway that can be accessed from within its current room, if available. If there are no unexplored doorways in the room, it will go to the nearest unexplored doorway anywhere on the map, and continue its exploration efforts from there.

Thus, it will explore the map room by room, going to the nearest unexplored area whenever available. This reduces redundant movement by ensuring that all rooms are finished before the robot leaves them, avoiding the problem of having to go back to previous sections to finish exploration of smaller areas that were missed the first time around. The simplicity of the spiral movement, and this reduction of redundancy, are the primary drivers of the speed increase that is seen over previous exploration algorithms.

All of this is the case for just a single robot exploring an environment, and is fully capable of exploring complex environments. When multiple robots are used to explore environments collaboratively, the robots move in much the same way, spiraling around the edges of the rooms and marking doorways. As they do this, they regularly share their maps with each other, allowing for more efficient exploration, as robots can walk side by side, one exploring along the wall, the others exploring further toward the center of the room.

This behavior is emergent from the nature of the algorithm, and has not been specifically designed or made to happen. Further, as doorways are discovered, an auction begins, where robots in the room send their distance to the new doorway, and half of the available robots leave to explore the adjacent room, leaving the other half to finish the current exploration. This causes the robots to split up further, and earlier, allowing for more parallel exploration.

All of this has been implemented in the MAES simulator, running on Unity, and experimenting extensively with various configuration of settings, on a large variety of maps of different sizes. For comparison, the same experiments have been run with The Next Frontier (TNF) algorithm, an information gain frontier focused algorithm, as well as another of our algorithms, a greedy algorithm with map sharing capabilities called Greed. The results show a large improvement over TNF on all but the smallest maps, where TNF moves quite efficiently to unseen areas in direct reach of it. The speed increase on all other maps is large, with Minotaur being upwards of 2 to 4 times faster in nearly all cases.

The Greed algorithm, on the other hand, is in many cases just as fast or even slightly faster than Minotaur. The complexity of maps, with a generous exploration requirement to finish allows the sim-

plicity of Greed, especially with the emergent behavior created by communication, to be very fast. In some maps, particularly large and more open maps, and with line of sight communication, Minotaur is faster, owing to the speed of the wall following and the efficiency of the central spiraling and the lack of redundancy since corners are covered more consistently than Greed can manage.

With these positive results, the Minotaur algorithm has proven itself useful in exploration efforts, in accordance to the simulations. While more testing and experimentation is always useful, especially on real robots in real environments, we believe that the Minotaur algorithm is functional and applicable for use in real world scenarios, as a fast and efficient method of autonomously exploring areas using multiple robots, with the Greed algorithm providing a simplified alternative with very similar speed when using few robots.

The code is released open-source on https://github.com/Molitany/MAES with the experiments and implementation of other algorithms. A demo can be seen here https://youtu.be/gjnVm46Ezd0.

Spiralling Human Heuristics Exploration with doorway detection

The Minotaur Exploration Algorithm

Rasmus Borrisholt Schmidt, Andreas Sebastian Sørensen, Thor Beregaard

Software/Datalogi, cs-24-ds-10-04, 2024

AALBORG UNIVERSITY
STUDENT REPORT

Software
Aalborg University
https://www.aau.dk/

Title:

Spiraling Human Heuristics Exploration with doorway detection - The Minotaur Exploration Algorithm

Project:

Master thesis

Project period:

January 2024 - June 2024

Project group:

cs-24-ds-10-04

Participants:

Rasmus Borrisholt Schmidt
Andreas Sebastian Sørensen
Thor Beregaard

Supervisor:

Michele Albano

Page Numbers: 16

Date of Completion:

May 31, 2024

Abstract

Exploration of unknown environments is important for the versatility of autonomous robot swarm systems. The faster they can fully explore an area, the faster a coordinated plan can be made, or points of interest found. This has applications in fields like space exploration and search and rescue.
Previous algorithms have often focused on frontier based, or nature-inspired heuristics. We present a human-heuristics based exploration algorithm, Minotaur, that enables simple and efficient exploration of static buildings.
The algorithm follows walls to discover rooms and doorways, after which it follows the previously explored area as marked on a SLAM map. On finishing exploration of a room, robots continue through the nearest unexplored doorway. Multiple robots share these maps and doorways, and can communicate to split up during their exploration efforts to reduce redundant exploration, while enabling fast exploration through both designed and emergent behavior.
Comparative experiments have been run in the MAES simulator, comparing with both TNF and a greedy approach. These show an improvement over the previous algorithms, particularly when the algorithms are limited in their communication.
Minotaur spends about 1/2 to 1/4 of the time to explore a map compared to TNF, while being slightly faster than Greed in some cases. There is a clear trend for more agents to further improve the speed of Minotaur in comparison to Greed, showing the applicability of Minotaur in systems using a large amount of robots.

Contents

I. Introduction

Robot exploration is a common problem in swarm robotics, with applications in search & rescue, fire-fighting, emergency response, and space- and undersea exploration. [1] Much work has been proposed regarding various exploration algorithms to solve this issue, some inspired by animals [2] [3] [4] [5], some using frontier exploration [6] [7] [8], and some using various mathematical concepts [9] [10]. Many of these rely on unrealistic expectations like global communication or a fully discrete environment [11]. Furthermore, some over-complicate the issue, and introduce systems that can result in occasional deadlocks because of issues with the way decisions are made on the map, and a lack of fall-back options.

In a bid to remedy this issue, we propose the Minotaur exploration algorithm, a human heuristic based algorithm. Inspired by common ideas for exploring mazes, such as keeping to the right wall, it further expands to groups of robots with an efficient communication scheme that keep the robots apart to avoid redundancy.

The robot explores along the nearest wall, at a little less than their maximum vision range, detecting doorways on the way. If there are multiple robots, half of the robots in the room will enter the discovered doorway, and explore from there, spreading throughout the map. When an explored area is seen, robots follow the explored area similarly to the initial following of the walls. The robots continuously communicate their maps and marked doorways, to limit redundant movements. When a robot finishes the exploration of a room, it moves to the nearest unexplored doorway that can be reached without passing another doorway, and thereby entering a new room. If there is no such doorway, the robot moves through the nearest unexplored doorway, even if it has to move through a previously explored doorway.

The code is released open-source, and is available at https://github.com/Molitany/MAES

The paper is organized as follows: section II describe the related works, section III details the methods used and the specifics of the algorithm and pseudocode. section IV shows the design as written in the simulator and implementation details. In section V the experiments are described, and their data is presented and discussed in section VI, while section VII will conclude on the paper and section VIII provides suggestions and ideas for future work.

II. Related Work

A. MAES

Multi Agent Exploration Simulator (MAES) [12] is a swarm robot simulator similar to Gazebo [13] and ARGoS [14], given that they can be used for exploration algorithms and have ROS2 [15] support. The benefit of using MAES is that it does not require a lot of the overhead that the others have. Without having to focus on details, the algorithms can take full focus.

Further, MAES has the support for creating transmission models dependent on distance and walls passed through, while also supporting attenuation based communication. MAES utilizes the Unity engine for visualization and generation of the maps, as well as handling the robots and creating templates for the simulation.

B. CME

One of the backbones of robot exploration is Coordinated multi-robot exploration (CME), often referred to as Near-Frontier Exploration (NFE). [6] It is a simple frontier-based algorithm that sets values to the occupancy grid, with the cost to each cell being the distance of the robot to the cell. It also uses a utility factor, where the robot will

consider the destination of other robots and choose its destination away from others destination. CME is the basis point for MinPos [7] that in turn is the comparison of The Next Frontier.

C. TNF

The Next Frontier (TNF) [8], is an exploration algorithm designed to use information gain and distance cost as a basis for autonomous swarm exploration. It is inspired by MinPos. [7] TNF utilizes two parameters, an alpha and beta value, where alpha prioritizes frontiers further away and beta prioritizes frontiers closer to the agent. The alpha and beta values ratio decides which frontier to prioritize, this being the distance factor.

Furthermore, the information factor creates a window around cells and uses the occupancy grid to see if there are new places to explore, assigning a value to each tile of the occupancy grid. Lastly, there is the coordination factor which creates a negative wavefront from nearby robots, and lowers the possible information gain of frontiers near them, causing the robots to split up further. These three factors get combined into the utility factor for each tile. The tile with the highest utility factor is the one the robot moves to.

D. SSB

Spiraling and Selective Backtracking (SSB) [16], is a modified coverage algorithm of BSA-CM [17] [18], using spiraling patterns. With expectations of global communication and discrete grid-based movement, SSB spirals through known areas and updates other robots of the covered space and the nearby uncovered tiles it reserves for backtracking.

The backtracking is used to find new uncovered areas to spiral out from, and uses an auction mechanism to decide which robot should go to each backtracking point. If no backtracking point is available, or if the robot did not win any auction for a backtracking point, an auction is instead done for the nearest unvisited point. As the robot spirals and creates backtacking points, it reserves the points of the spiral and backtacking, and broadcasts them to other robots, limiting their movement in an attempt to reduce redundant movement.

E. Human Heuristics

These previous algorithms are generally based on a specific idea of exploration, or finding the greatest gain. These are very mathematically focused, but another approach is to instead attempt a more intuitive approach, and explore in a way that makes sense to humans, with a simple idea of the priorities that should be used, and then following those. This is the idea of human heuristics, using the ideas of humans to aid in exploration. Human heuristics are used in various tasks in robotics [19] [20].

An example of human heuristics is An Efficient Robot Exploration Method Based on Heuristics Biased Sampling [21]. They utilize computer vision with human heuristics to detect where doors are to separate rooms apart. Furthermore, it prioritizes finishing exploring the current room before proceeding to the next.

In our case, the exploration heuristic means exploring the environment one room at a time, noting down the exits for later, and then simply going to the closest unexplored exit when the current room is fully explored. Since no knowledge of future rooms is known, simply going to the closest one is the fastest known choice to explore more.

III. Methods

Here we will describe the main contributions of this paper: The Minotaur algorithm, with detailed explanations and pseudocode, and the derived Greed algorithm.

A. Minotaur Algorithm

To explain the algorithm, we will initially go over a constrained version for a system with only a single robot. This will be expanded to show the full multi robot behavior, with smaller black-box functionality around doorways that are then described in their own sections, all together creating the full algorithm. These sections are described as follows:

Single Robot Behaviour Handles the base behavior and complete exploration capabilities of a robot exploring alone

Multi Robot Behaviour Handles base behavior, communication and cooperation to allow for multiple robots to explore areas together

Doorway Detection A subsystem used by both Single Robot Behavior and Multi Robot Behavior to detect and mark doorways, to enable efficient movement between rooms, and locality prioritization

Doorway auction Subsystem used by the multi robot exploration that handles the communication about newly discovered doorways and the auction that happens to negotiate sending robots through the doorway to explore multiple rooms at once.

1) Single robot

The single robot algorithm fully explores areas without using communication or sending other robots through doorways. It explores along the wall nearest to the insertion point when there is one, otherwise it will move until it finds a wall, following it along bends and turns, marking doorways that fit the parameters.

After covering all walls in a room, it will explore the center of the room in a spiral pattern, until everything is explored. If it is, then it will enter the nearest unexplored doorway in the room, and continue the exploration from there, returning to unexplored doorways in previous rooms if none can be found in the new room.

Optimizing the order of doorways further beyond taking the closest is not possible, as nothing more is known about the map than what the robot has seen. The pseudocode for the single robot exploration can be seen in Algorithm 1.

Algorithm 1 Minotaur Algorithm (Single)

```
Input
    Visual range: Radius of robot sensors
Initialize the Minotaur
Check for nearby walls or explored areas in visual range
while No walls or explored areas are found do
    Move forward
end while
while Map is not explored do
    while Unexplored area is reachable without passing doorways do
        if Doorway found then
            Store the doorways, stay in the room and continue exploring along next wall
        end if
        if Wall or explored area in visual range && unexplored area ahead then
            Follow wall or explored area counterclockwise at max visual range
        else
            Move to the nearest unexplored area without passing doorways
        end if
    end while
    if Unexplored doorway accessible without passing other doorways then
        Move through the nearest unexplored doorway within room
    else
        Move through the nearest unexplored doorway
    end if
end while
```

2) Multiple robots

The methodology for multiple robots is very similar to the single robot, though with more steps required to make efficient use of all the robots. Following Algorithm 2 half of the robots (rounded down) ex-

Algorithm 2 Minotaur Algorithm (Multi)

```
Input
    Visual range: Radius of robot sensors
Half move to local right, rest to local left
while No walls are found do
    Continue moving in given direction
    Broadcast map, including walls, explored area, doorways and robot locations
end while
Half of group go clockwise, half counter-clockwise
while Map not explored do
    while Unexplored area is reachable without passing doorways do
        Broadcast map, including walls, explored area, doorways and robot locations
        if Doorway found (Algorithm 3) then
            Store and handle doorway (Algorithm 4)
            Half of doorway group go clockwise, half counter-clockwise
        end if
        if Wall or explored area in visual range && unexplored area ahead then
            Follow wall or explored area counter-clockwise at max visual range
        else
            Go to the nearest unexplored area without passing a doorway
        end if
    end while
    if Unexplored doorways exists then
        Cross nearest unexplored doorway
    else
        Go to the nearest unexplored area
    end if
end while
```

plore around the rooms in one direction, the others in the other direction, to quicker find doorways.
As they do this, and while they do anything, they broadcast their slam-map and any found doorways for other robots to add to their own maps. On finding a doorway, the robot marks it, and starts an auction with all the robots it can communicate with. They respond with a bid for entering the doorway, equal to the distance of their path to that doorway, so long as they can get there without passing another doorway. This ensures that no robot spends time moving to a doorway far away, and that all room that have started being explored also get finished. This is further described in Algorithm 4.
Some of this is inspired by SSB [16], particularly the spiraling movement, and the auction mechanism to decide which robot goes to important locations.

3) Doorway detection

When the robots see a wall, they will check if it is possible that it contains a door. This depends on whether there is a single wall or multiple walls. If there is a single wall, then when the robot finds a gap in the wall, it will continue along the direction of the wall as far as the door width parameter.
If there are multiple walls, then the robot will create intersection points between the walls as infinite lines. If these intersection points have been seen, are non-solid, and wide enough to make a doorway, then it is a door.

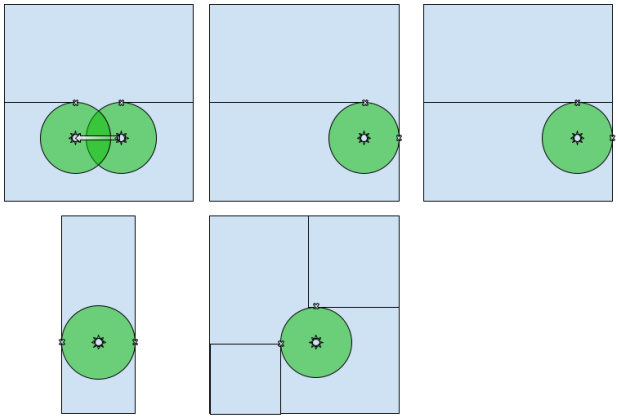

Figure 1: Base cases for the doorways

The different possibilities for potential doorways can be seen on Figure 1. Where the first case is a single wall, so the robot continues across the wall until it sees it continue. In the second case there is a door in the corner separating two rooms, where in the third case there is not. The fourth case is a hallway where the walls will never intersect.

Algorithm 3 Doorway detection

```
Input
    Visual range:  Radius of robot sensors
    Doorway width:  How large a door is
Output
    Doorway  Collection of doorways found or nothing
if Wall ends/stops then
    move forward doorway width
    if Wall continues after doorway width then
        Mark doorway and direction seen from
    end if
end if
if Two walls in vision area then
    Store the wall points
    Continue vision range forward
    Store the new wall points
    Create lines along wall direction between the points
    if Lines intersect && distance between wall points greater than robot size then
        Find intersection point of lines
        Move towards intersection point until it is explored
        Create line segments between wall points and intersection point
        if Robot size of line segments combined length is open then
            Mark doorway and direction doorways seen from
        end if
    end if
end if
if any doorways "seen" from both sides || doorway passed through then
    Mark doorway as explored
end if
```

Lastly, there is a doorway at the fifth case where two separate intersection points will be created, depending on the approached direction. Algorithm 3 covers all of these cases.

Doorway detection is likely to be highly dependent on the implementation environment, but similar issues should appear in most map representations.

4) Doorway auction

When there are multiple robots that are able to communicate with each other, they should be smart about how they spread out. When a robot, Alice, finds a door that has two robots, Bob and Charlie, in their communication range, Alice should send out a "doorway found" message, and if Bob and Charlie receive the message, and are able to reach that door without passing through another doorway, then they respond with their distance to that door. Afterward, Alice tells the half of the respondents who have the lowest bid to proceed through that door with Alice. If only Bob answers the doorway found message, then Alice will go through the door alone, leaving Bob to finalize the previous room. However, if neither Bob nor Charlie sends an acknowledgment, that means that Alice is alone in the room and Alice will then finalize the room by themselves. This is formalized in Algorithm 4.

Algorithm 4 Doorway auction

```
Communicate doorway location
if Receiving robot can reach the doorway on the same side without passing any doorway then
    Receiving robots give ACK and distance
end if
if Half of Available > 1 then
    Command other robots to follow through doorway
else if Half of Available == 1 then
    Move through doorway
else
    Pass Doorway and continue exploring room
end if
```

Algorithm 4 is also visually explained on Figure 2, where green areas are unexplored, the colored dots are robots, and the dashed lines are communication lines, with their color signifying whether they respond with an ACK (green) or NACK (red). On Figure 2 Pink finds a door and communicates to the other robots in range. Cyan will be able to go to the room.

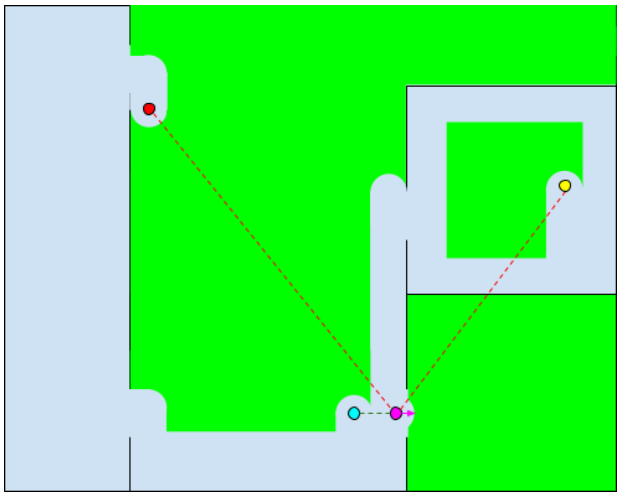

Figure 2: Communication scenario for Algorithm 4

Yellow cannot get to the door Pink is communicating about without passing another doorway, and therefore replies with a NACK.
While Red technically can get to the door, it does not know this, as the area between is unexplored. From its perspective, it would have to backtrack through multiple doorways to reach Pink, and it therefore also replies with a NACK.
This ensures that the robots split up, but still prioritize fully exploring rooms. If the robots can always expect previous rooms to have been explored, they can leave rooms and mark the doorways explored without issue. They can then continue exploring along the unknown path, reducing redundancy and improving parallel exploration capabilities.

B. Greed algorithm

When the Minotaur has no nearby unseen areas, or unexplored doorways to go through, it will move to the nearest unexplored tile anywhere on the map and continue exploration from there.
This feature has been extracted into its own algorithm, creating a greedy algorithm that always moves to the nearest unexplored tile, and still communicates and shares its map with other robots. This is similar to CME, but instead of being frontier-based like CME [6], and only sharing locations, the greedy algorithm works on the nearest unseen tile, and shares the whole map, allowing more parallel exploration, while still being very simple.
Due to its greedy nature, we have simply called it Greed. Being a greedy algorithm, it is expected to perform quite well, and perhaps be faster than Minotaur in some cases, especially with few robots since it does not have the limitations on its baseline movement that is imposed on Minotaur to make it follow walls. The difference is likely to be minor, since Minotaur should get an advantage by avoiding some redundancy, and be faster with more robots because of the auction system of multiple robots. Experimental results showing this can be seen in section VI.

IV. Design & Implementation

A. Spiraling implementation

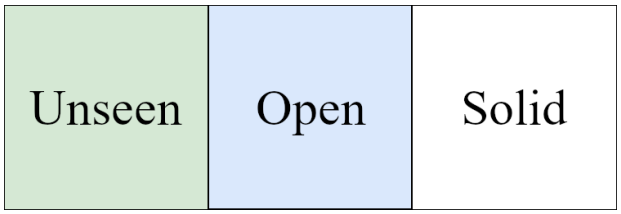

Figure 3: Representations of the different tile statuses

The implementation of spiraling in the MAES simulator relies on the overlaid grids on the SLAM map that the robot creates during runtime and uses to navigate. Each tile on this grid can have one of three states, Open, Unseen and Solid, as can be seen in Figure 3.
While following the wall on the initial exploration of a room, it looks for solid tiles within its vision range. The robot then creates lines along these solid tiles that are continuing in the same direction, e.g. there can be a line that continues in front of and behind the robot. It decides which wall to follow by

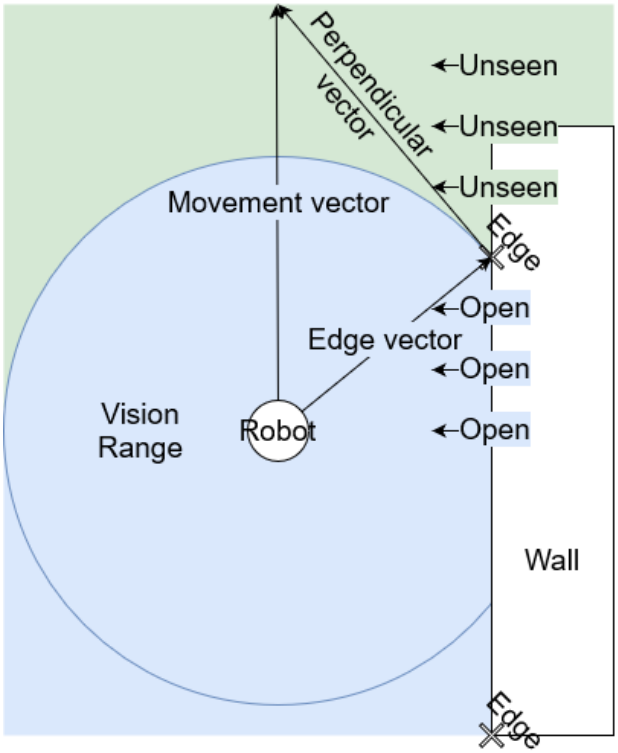

Figure 4: Figure showing a valid wall due to having an unseen where there presumably is a wall with edges and movement vector based upon those edges.

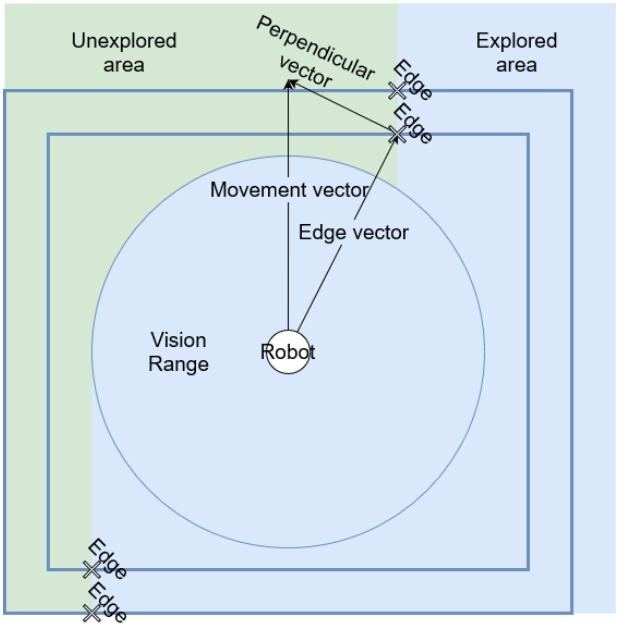

Figure 5: Figure showing a robot following an explored area by utilizing an edge point and creating a perpendicular vector from it to describe the movement vector.

checking if there are any unseen tiles in the line's direction where the wall might continue. If there are none, it has already explored that wall.

When the robot has multiple candidate walls, it will prioritize moving in the same direction that it is already moving. The robot will get the furthest points it can see on the walls and make a perpendicular vector in a forwards direction, pointing where it has to move to next. This has been illustrated on Figure 4.

If the wall stops, or turns away from the robot without having a doorway, the robot will initially lose track of it, but then see the turn and create a new line, before following that around the bend.

When reaching the end of a confined area, like a dead-end in a hallway, the robot will return to the last seen area with unexplored tiles and continue the spiraling pattern from there, allowing smooth spiraling of complex layouts.

On completing the first lap around the edge of the room, the robot will then start to spiral around the edge of the previously explored area. This is tricky, since as opposed to walls, edges between explored and unexplored areas disappear when the unexplored area in question is explored. Therefore, the robot must use the fact that the explored area is already in its SLAM map, to look ahead beyond its own vision range and follow the edge there. A grid 2 tiles larger than its vision range in all directions is used for this, following the edge similarly to how the walls are followed. This is illustrated on Figure 5.

If the robot is no longer surrounded by any unseen tiles, it will move greedily towards the nearest unseen tile in the map, found by searching the map in a spiral pattern around the robot using a flood fill algorithm. This allows the robot to fill in all holes in the map automatically at the end of the spiral, if

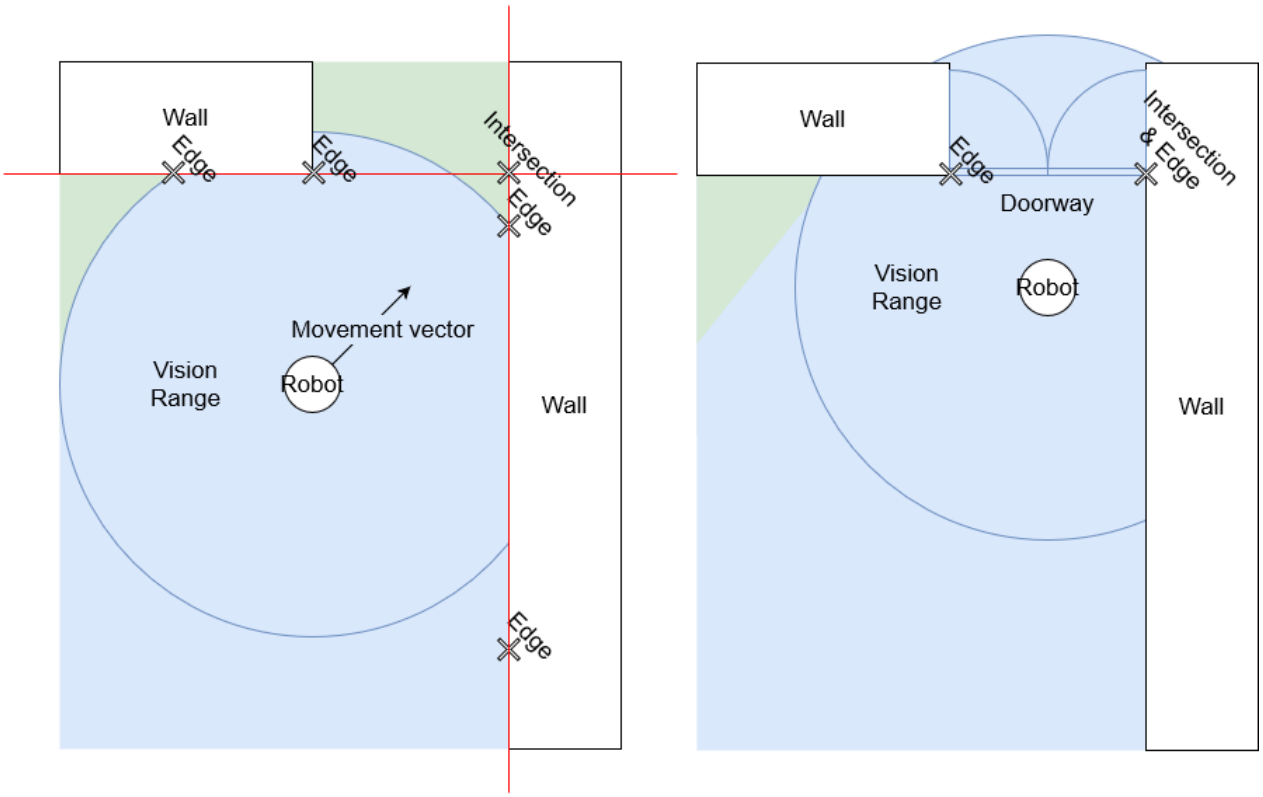

Figure 6: Doorway detection when seeing multiple walls. The left shows the state on detection of a potential door, and the right shows the state after moving to check intersection point.

any such holes were to occur.

Having finished the exploration of a room, the robot is now ready to enter the nearest unexplored doorway and continue the exploration.

B. Doorway implementation

While the robot is moving to any destination, it checks for doorways, unless it is moving towards an intersection point or to a doorway that is already found. These exceptions make sure that it does not keep computing the same intersection point.

For a doorway with a single wall, as seen on Figure 7, the robot sees that the next tile of the wall is an open tile. When this happens, the robot will add a distance equal to the doorway width parameter to its next movement. The robot will then be standing at the end of the movement vector and will be able to see if the wall continues, and if so, creates an intersection point between the walls. Then it checks if that is a valid door, according to the set parameters.

If there are two or more walls, the robot looks for an intersection point between those, as can be seen on Figure 6. When the intersection point is in the unseen, as is likely to happen, the robot will go to explore the intersection point. After the robot has moved to see the point, it takes the closest points to the intersection from the two walls that created it, as can be seen on the two edge points on the right state of Figure 6. With those two points, the robot checks if they create a valid doorway, by once again creating a central point and checking if the distance corresponds with the doorway width parameter.

When the robot finds a valid doorway, it will check

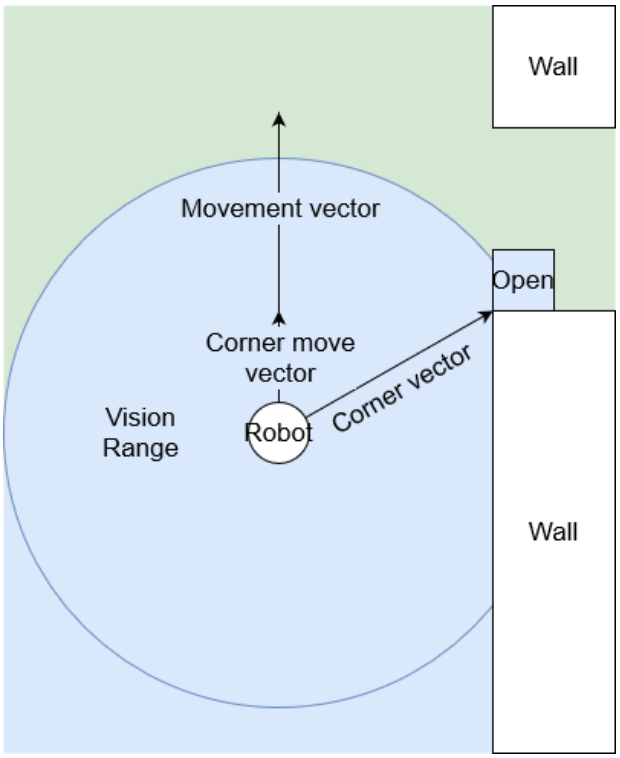

Figure 7: Doorway detection when seeing a single wall.

if it already had the doorway in memory. It does this by taking every tile the doorway is created from and checking along that line, in the direction from the robot to the door, tile by tile for the whole wall width. Should any of the tiles coincide with another doorway's tile, then it is the same doorway and the doorway is marked as explored, as it has been seen from both sides.

C. Deadlock prevention

We have added deadlock prevention as a failsafe for the robots. This is to make sure that no matter what, the robots will always complete their simulations. The deadlock prevention is triggered after 25 ticks (2.5 seconds) of the robot not moving. After deadlock prevention is triggered, the robot will store it's current waypoint and try to greed out of its location by going to the nearest unexplored tile in the room. If there is no unexplored tile in the room, then the nearest unexplored doorway, and if that is not available either, then the nearest unexplored tile anywhere on the map. Should the waypoint be the same, it will go to a nearby open tile and try again until it becomes unstuck.

V. Experiments

To determine the performance of Minotaur, we conduct a series of experiments to compare Minotaur with TNF, as well as Greed, as defined in subsection III-B. Each of the experiment are run on maps of sizes 50 by 50, 75 by 75, and 100 by 100, with 100 of each map size within each experiment case. Each algorithm is tested with three different types of communication enabled, namely, global communication, material communication, and line of sight (LOS) only communication. Furthermore, the experiments also consist of varying amounts of robots, namely 1, 3, 5, 7, and 9, running the same algorithm. The robots are also tested with spawning at a random point together, as well as spawning at separate random points. Each combination of these parameters are tested.[1]

These parameters are chosen to see the effect of an increasing amount of robots on the exploration speed on various maps sizes. On top of this, the different forms of communication are chosen to see if there is a noticeable impact on the different algorithm with global communication when compared to the more realistic communication through materials or line of sight only communication. Finally, the different spawn positions are chosen to see the potential impact of a single insertion point, and comparing it to situations where multiple insertion points are available.

With the use of Unity scenes for every experiment type, all experiments can be easily reproduced by building the project with the individual scene that is intended to be tested. Since the experiment can

[1]Experiment code can be found at https://github.com/Molitany/MAES/tree/experiments

Parameter	Value(s)
Timeout in Ticks	36000 (1 hour)
Vision range	7
Communication	Global, Material, and line of sight
Amount of robots	1,3,5,7, and 9
Map sizes	50x50, 75x75, and 100x100
Maps per size	100
Algorithms	Minotaur, Greed, and TNF
Door width	2
TNF parameters	alpha: 1, beta: 10
Spawn positions	Randomly together, and randomly separate

Table I: Table describing the various experiment parameters.

take a long time, checks have also been put in place that allow for execution to be interrupted during the experiments, and restarted, with the seed returning to the same position it was previously, without having to rerun previously finished experiments.
Hardware specification of the experiment running machines are not mentioned, as they are fully independent of the experiment results. The only measurement taken is the amount of ticks taken to complete an exploration, and the only impact on this that the performance of the CPU running it has, is the computation time. All experiments are run on Windows machines.

VI. Results

As can be seen in the graphs on Figure 10 and Figure 11, the Minotaur algorithm is much faster than TNF in all but a very small set of cases, and comparable, if a little slower than Greed when not using line of sight communication, or with few robots. One-on-one Greed is faster than Minotaur, but this is not too surprising, as Minotaur cannot use its auctions to split up and gain an advantage if alone, an aspect that is further examined in subsection VI-A.

With global- and material-based communication, the robots can communicate enough that they all split into what can amount to a series of one-on-ones between the algorithms, where again, Greed wins.
With LOS communication, Minotaur can use its auctioning more efficiently since it is already limited to the room, and Greed does not gain an advantage from being able to communicate constantly, allowing the Minotaur to pull ahead. This is good for the Minotaur's prospects as an algorithm, since LOS is the worst-case communication scheme.
Greed was expected to be very fast, with Minotaur hopefully coming out on top in a few cases. Complex maps would benefit from the earlier splitting that the auction system in Minotaur allows for, and from the backtracking avoidance Minotaur has by fully exploring rooms.
Greed, however, appears to inherit many of the emergent attributes of Minotaur, like exploring rooms in lines along each other. Greed generally moves quickly through the environment without "wasting" time on some details that end up taking a large amount of time for the Minotaur, as described in subsection VI-A. This causes Greed to edge out the Minotaur on some communication schemes or with few robots, though only very slightly, and both are still much faster than TNF.
Further results about the comparison with other algorithms can be inferred from previous experiments run by other authors, as TNF was compared favorably with MinPos [7], which was in turn compared favorably with CME [6]. This indicates that both Minotaur and the Greed version are faster than all these previous algorithms by a large margin.

A. Minotaur Slowdowns

On Figure 8 the behavior of Minotaur and Greed when moving into a corner can be seen. Greed simply moves along the corner, finding the closest area, and moving there. Minotaur instead comes

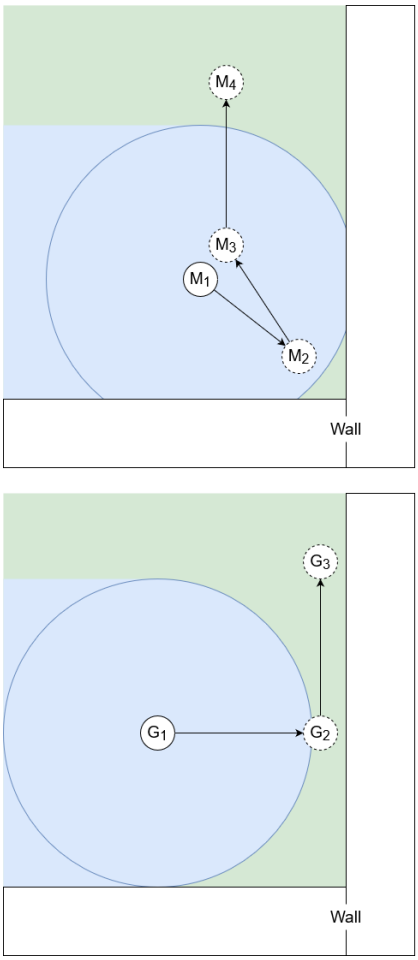

Figure 8: Minotaur exploring slower due to attempted doorway detection in a corner

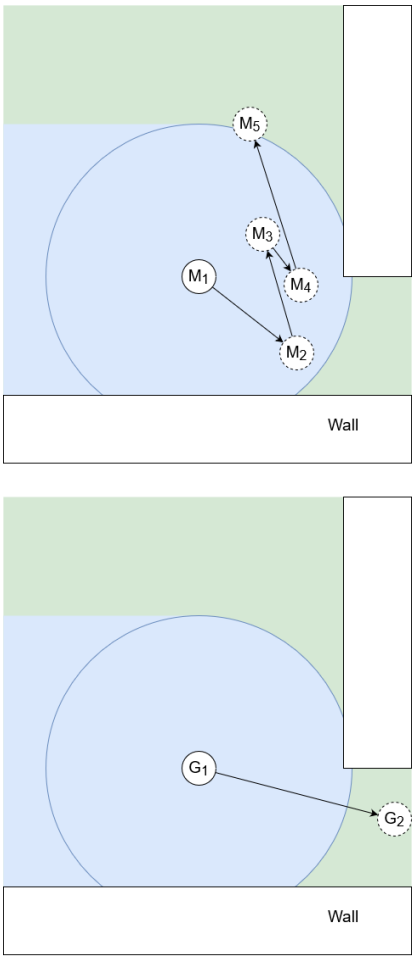

Figure 9: Minotaur exploring slower due to multiple doorway detection

upon the corner, and when it sees the opposing wall of the corner, needs to check if there is a doorway. This causes it to move to M_2, much further toward the wall than Greed ever does, before returning and continuing its following of the wall. Over a large map, with many corners, this causes a slowdown.

For Figure 9 the cases are similar, except there is actually a doorway. The Greed algorithm once again moves simply, and just enters the doorway, exploring new space all the while. The Minotaur once again sees the opposing wall, and moves to M_2 to see if there is a doorway. When the doorway is detected, it moves away to M_3 to continue along the wall, marking the doorway. It then sees the wall from a different angle, and creates a separate intersection point, and moves to explore it, moving to M_4. The robot discovers a doorway it has already seen, dismisses it, and only then continues exploring the room. These wasted movements all cause slowdown, providing Greed an advantage in direct exploration speed.

As can be seen on the Table II the Minotaur algorithm is the fastest algorithm when utilizing 5, 7, and 9 robots when the communication style is line of sight. Furthermore, on Table III with the same communication but spawning together results in

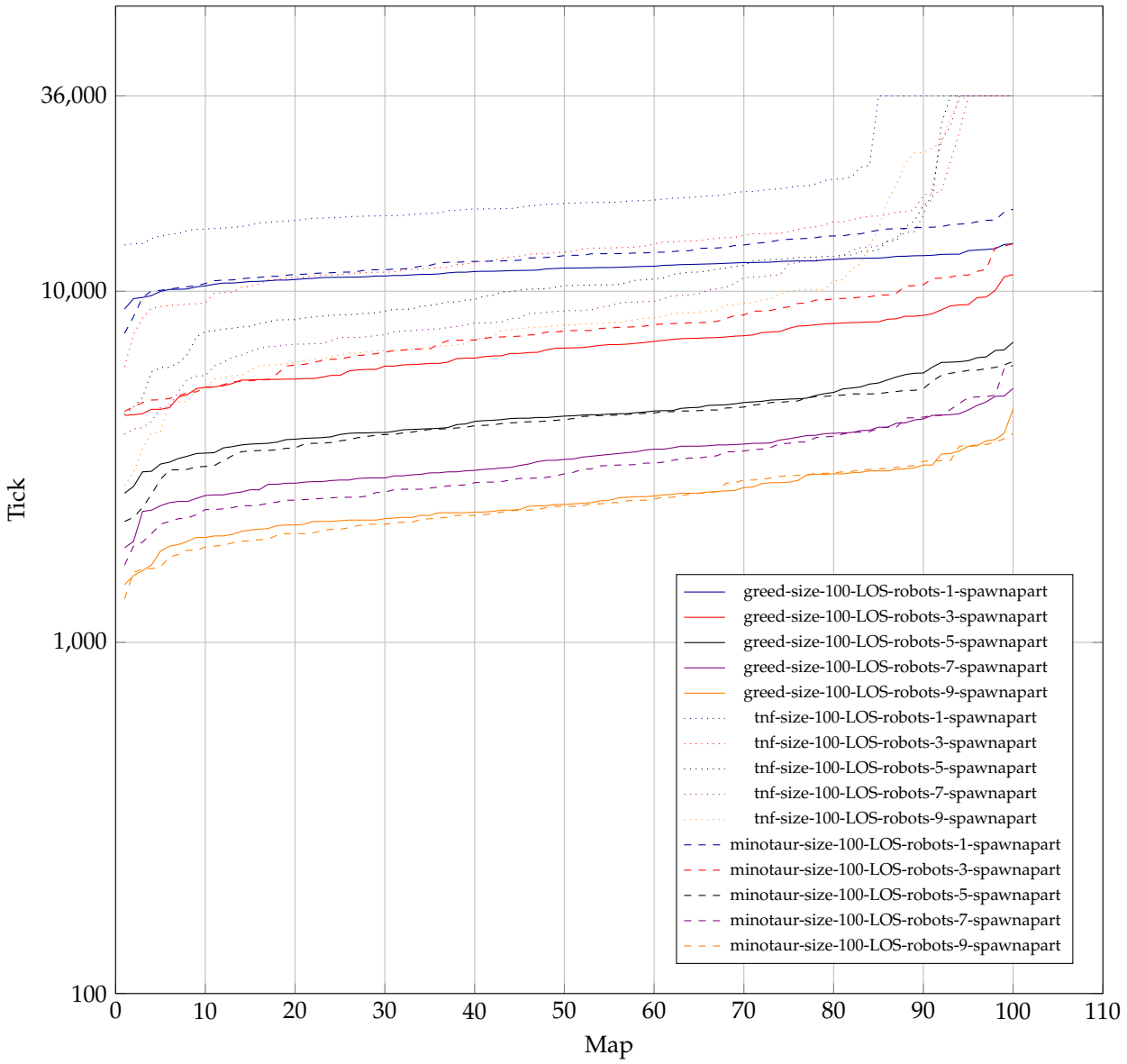

Figure 10: Cactus plot over the various strategies with the configuration that uses a map size of 100, using LOS for communication, and spawning apart.

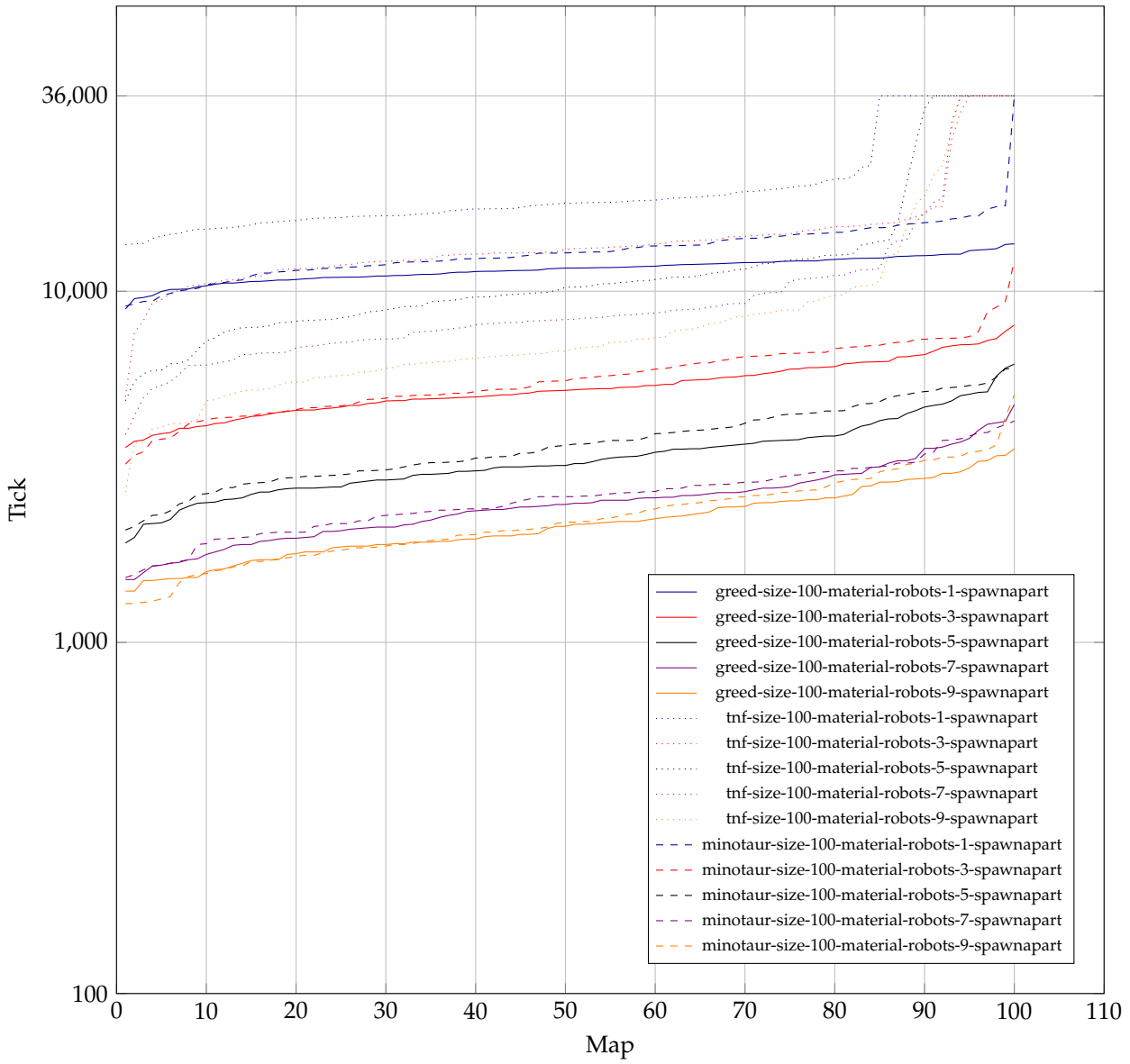

Figure 11: Cactus plot over the various strategies with the configuration that uses a map size of 100, using material for communication, and spawning apart.

Minotaur being the fastest with 7 and 9 robots. As mentioned, this is likely because the Greed robots are faster individually, but the more robots using the Minotaur algorithm, the greater the advantage they get from the doorway detection and auction mechanisms. This is further shown on Figure 10.
A notable result is that if the robots spawn apart from each other, such as in a situation with multiple possible insertion points during search and rescue operations, the exploration gets significantly faster. This effect is increased with the amount of robots, as they will immediately be able to effectively spread out and explore a larger area.

VII. Conclusion

We have created an efficient exploration algorithm based on human heuristics, that can fully explore unknown areas very quickly. The algorithm uses simple ideas of wall following, following explored areas, fully exploring rooms before leaving, and exploring doorways.
To that, it adds an auction mechanism that causes robots to split up when new doorways are discovered to reduce redundant exploration as much as possible, while exploring more in parallel, to avoid many robots spending time down deep dead ends. When no nearby doorways or unexplored areas are available, it falls back to a single greedy movement, going to the nearest unexplored tile. This aspect has been extracted and used to create the Greed algorithm, that retains the map-sharing of the Minotaur.
Comparisons have been made with The Next Frontier (TNF), and the extracted Greed algorithm. Minotaur is 2 to 4 times faster than TNF while being comparable with Greed. Particularly when using line of sight communication, Minotaur outperforms Greed as the amount of robots increase. This is especially visible when 9 robots spawns together, in which case Minotaur is 5.6% faster than Greed.
In general, the more robots are added, the better Minotaur is compared to Greed, but both are always far better than TNF. With detailed pseudocode and simple concepts, Minotaur should be implementable in a real world system. Minotaur is useful for exploration of environments in the cases where communication is limited to line of sight or where many robots are used. Greed can be used with small robot groups, or when better communication is available, to great effect.

VIII. Future Work

A. Testing on cave maps

This implementation of the Minotaur algorithm has only been tested on building maps, as the algorithm was designed around common building features. It would also be valuable to perform an experiment to see if the algorithm performs well on cave maps.

B. Real Robots

While the algorithm shows promising results in the MAES simulator, implementing the system on real robots, and testing them on areas created in the real world would likely present unique challenges and show the applicability in the real world. With realistic SLAM maps, sensors and communication, the algorithm may perform differently.

C. Testing against more algorithms

We have tested against TNF and Greed, and have good results compared with them. Testing with other algorithms, and making more comparisons with different methodologies, would provide a clearer idea as to the Minotaur algorithm's efficiency.

D. Extend to 3 Dimensions

While 2D ground based exploration is useful, with many robots still being on the ground with wheels

or legs, expanding to 3 dimensions would allow the algorithm to be used with aerial drones, mapping out far more complex spaces. This would likely require a different understanding of the SLAM map, and possibly of the notion of exploration, and the grazing of the robots would be much more important to ensure fast exploration, especially if the space is very large compared with the visual range of the robots.

E. Minotaur in heterogeneous swarms

Many current exploration efforts are focused on exploration with multiple types of robots, so-called heterogeneous swarms, like airborne- and ground-based robots. [22] Mixing the capabilities of multiple drones, and testing environment exploration with different kinds of movement and obstacle avoidance abilities would be an excellent proof of viability for real life exploration efforts.

Bibliography

[1] J. P. Queralta, J. Taipalmaa, B. Can Pullinen, *et al.*, "Collaborative multi-robot search and rescue: Planning, coordination, perception, and active vision," eng, *IEEE access*, vol. 8, pp. 191 617–191 643, 2020, ISSN: 2169-3536.

[2] G Nicola, N Pedrocchi, S Mutti, P Magnoni, and M Beschi, "Optimal task positioning in multi-robot cells, using nested meta-heuristic swarm algorithms," *Procedia CIRP*, vol. 72, pp. 386–391, 2018.

[3] F. Gul, I. Mir, W. Rahiman, and T. U. Islam, "Novel implementation of multi-robot space exploration utilizing coordinated multi-robot exploration and frequency modified whale optimization algorithm," *IEEE Access*, vol. 9, pp. 22 774–22 787, 2021. DOI: 10.1109/ACCESS.2021.3055852.

[4] F. Gul, I. Mir, S. Mir, and L. Abualigah, "Multi-agent robotics system with whale optimizer as a multi-objective problem," *Journal of Ambient Intelligence and Humanized Computing*, vol. 14, no. 7, pp. 9637–9649, 2023.

[5] F. Gul, I. Mir, and S. Mir, "Aquila optimizer with parallel computing strategy for efficient environment exploration," *Journal of Ambient Intelligence and Humanized Computing*, vol. 14, no. 4, pp. 4175–4190, 2023.

[6] W. Burgard, M. Moors, C. Stachniss, and F. Schneider, "Coordinated multi-robot exploration," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 376–386, 2005. DOI: 10.1109/TRO.2004.839232.

[7] A. Bautin, O. Simonin, and F. Charpillet, "Minpos: A novel frontier allocation algorithm for multi-robot exploration," in *Intelligent Robotics and Applications: 5th International Conference, ICIRA 2012, Montreal, Canada, October 3-5, 2012, Proceedings, Part II 5*, Springer, 2012, pp. 496–508.

[8] R. G. Colares and L. Chaimowicz, "The next frontier: Combining information gain and distance cost for decentralized multi-robot exploration," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing, Pisa, Italy, April 4-8, 2016*, S. Ossowski, Ed., ACM, 2016, pp. 268–274. DOI: 10.1145/2851613.2851706. [Online]. Available: https://doi.org/10.1145/2851613.2851706.

[9] S. Mirjalili, "Sca: A sine cosine algorithm for solving optimization problems," *Knowledge-based systems*, vol. 96, pp. 120–133, 2016.

[10] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, 1995, 1942–1948 vol.4. DOI: 10.1109/ICNN.1995.488968.

[11] E. Beck, B.-S. Shin, S. Wang, T. Wiedemann, D. Shutin, and A. Dekorsy, "Swarm exploration

and communications: A first step towards mutually-aware integration by probabilistic learning," *Electronics*, vol. 12, no. 8, 2023, ISSN: 2079-9292. DOI: 10.3390/electronics12081908. [Online]. Available: https://www.mdpi.com/2079-9292/12/8/1908.

[12] M. Z. Andreasen, P. I. Holler, M. K. Jensen, and M. Albano, "MAES: a ROS 2-compatible simulation tool for exploration and coverage algorithms," *Artif. Life Robotics*, vol. 28, no. 4, pp. 757–770, 2023. DOI: 10.1007/S10015-023-00895-7. [Online]. Available: https://doi.org/10.1007/s10015-023-00895-7.

[13] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, 2149–2154 vol.3. DOI: 10.1109/IROS.2004.1389727.

[14] C. Pinciroli, V. Trianni, R. O'Grady, *et al.*, "ARGoS: A modular, parallel, multi-engine simulator for multi-robot systems," *Swarm Intelligence*, vol. 6, no. 4, pp. 271–295, 2012.

[15] S. Macenski, T. Foote, B. P. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Sci. Robotics*, vol. 7, no. 66, 2022. DOI: 10.1126/SCIROBOTICS.ABM6074. [Online]. Available: https://doi.org/10.1126/scirobotics.abm6074.

[16] A. Gautam, A. Richhariya, V. S. Shekhawat, and S. Mohan, "Experimental evaluation of multi-robot online terrain coverage approach," in *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2018, pp. 1183–1189. DOI: 10.1109/ROBIO.2018.8665196.

[17] E. Gerlein and E. González, "Multirobot cooperative model applied to coverage of unknown regions," *Multi-Robot Systems, Trends and Development*, pp. 109–130, 2011.

[18] E. Gonzalez, O. Alvarez, Y. Diaz, C. Parra, and C. Bustacara, "Bsa: A complete coverage algorithm," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, 2005, pp. 2040–2044. DOI: 10.1109/ROBOT.2005.1570413.

[19] G. Bekey, H. Liu, R. Tomovic, and W. Karplus, "Knowledge-based control of grasping in robot hands using heuristics from human motor skills," *IEEE Transactions on Robotics and Automation*, vol. 9, no. 6, pp. 709–722, 1993. DOI: 10.1109/70.265915.

[20] C. Tijus, E. Zibetti, V. Besson, *et al.*, "Human heuristics for a team of mobile robots," in *2007 IEEE International Conference on Research, Innovation and Vision for the Future*, 2007, pp. 122–129. DOI: 10.1109/RIVF.2007.369145.

[21] J. Liu, Y. Lv, Y. Yuan, W. Chi, G. Chen, and L. Sun, "An efficient robot exploration method based on heuristics biased sampling," *IEEE Transactions on Industrial Electronics*, vol. 70, no. 7, pp. 7102–7112, 2023. DOI: 10.1109/TIE.2022.3203762.

[22] J. V.-V. Gerwen, K. Geebelen, J. Wan, W. Joseph, J. Hoebeke, and E. De Poorter, "Indoor drone positioning: Accuracy and cost trade-off for sensor fusion," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 1, pp. 961–974, 2022. DOI: 10.1109/TVT.2021.3129917.

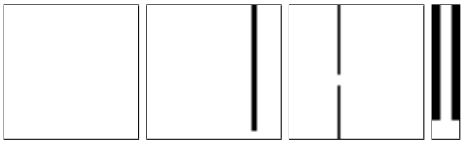

Figure 12: The movement and doorway testing maps

Appendix A
Testing

For the testing we have implemented a testing strategy, where we do continuous manual testing of both individual new features and the system as a whole at every iteration, to ensure changes are not breaking any system. This is supported by Unit testing where applicable.
We used the Unity testing suite, as well as tests on custom maps, to ensure that behavior like wall and door movement is consistent across different development stages. This strategy was necessary because of the specific design of MAES limiting the applicability of common unit and integration tests, as well as the novel nature of the algorithm and surrounding systems requiring very abstract testing.
Furthermore, a small change in the settings will produce a completely different result because of the complexity of the system. The effect of this was reduced to some extent by the testing procedures, but the chaotic nature of it necessitated a large amount of manual, structured testing.

A. Unit tests

MAES already had some unit tests already defined, those being:

- ROS TCP connection
- Movement tests: rotation and forward movement
- General communication tests: including distance travelled, wall distance travelled, and transmission success rate

We have previously introduced more unit tests for the material communication. These encapsulate both the general communication tests and testing the attenuation result of various cases to ensure that the attenuation is calculated correctly. Furthermore, we have created doorway detection tests for the Minotaur exploration algorithm, where a Minotaur will go through some bitmaps created specifically for the tests. The maps are:

- A blank map with a border.
- A map with a door near the border separating two rooms.
- A map with a door in the middle separating two rooms.
- A map with a hallway separated by a door near the starting location.

They are also visualized in Figure 12. While these maps were being run, the execution of the algorithm can be checked to ensure that it follows walls, seen areas, cover corners, and greed to the correct tile. In all of MAES there is a total of 99 unit tests with 15.6% code coverage and 13% method coverage. This low percentage is in part due to the packages imported into MAES not having code coverage, and partly due to the algorithms being hard to unit test, as they are incredibly complex systems where a

single change to a variable can completely change the execution. This means unit tests with set goals are problematic. However, the base movement parts, moving forward and rotating, are covered as well as the communication part using *TestAlgorithms*, which are very basic algorithms.

Appendix B
Development Process

We agreed initially to very regular meetings about the current development process, and the process of the project, especially taking previous experience into account. At the same time, we agreed to meet for work every day, rather than focus on a work-from-home style, as we had also had previous bad experience with that. All design work was done collaboratively, often with one or two members having the base idea, and workshopping it among everyone to work out the problems, often ending up with a mix of various ideas from everyone, ensuring even contributions.
We agreed to do pair programming when developing most features, since bugs were a large problem in the MAES system, and having more eyes on things was useful, though full extreme programming was deemed unnecessary for the process, with the limited scope and team size. We had weekly meetings with our supervisor, giving invaluable direction and advice for the project, creating pseudo-sprints to work from based on the feedback from these meetings.

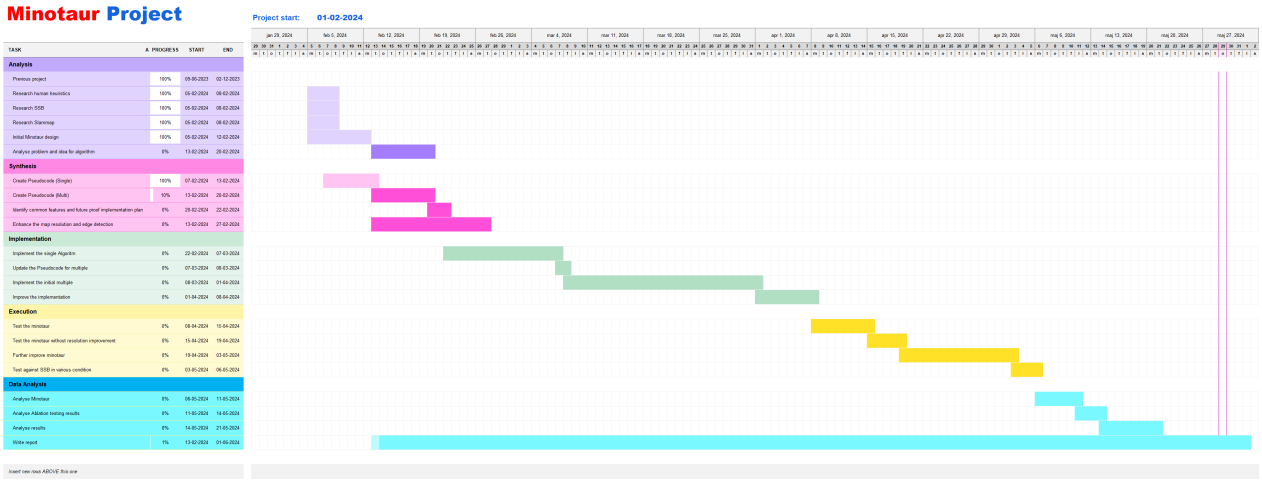

Figure 13: Initial Gantt chart of the project

Early on, we agreed on important features, like testing strategy and expectations, and created an early expected timeline in the form of a Gantt chart. This plan was followed somewhat closely in the early phases of the project, though, as expected, the actual process ended up somewhat different.
As the final stretch of the project approached, we created another timeline for the final 8 weeks, with weekly objectives, to ensure we stayed within scope and that the work would be done in a timely manner to allow for possible corrections. This timeline was followed almost exactly all the way until the hand-in. The only changes were finishing some aspects unexpectedly quickly, allowing more time for polishing other aspects.

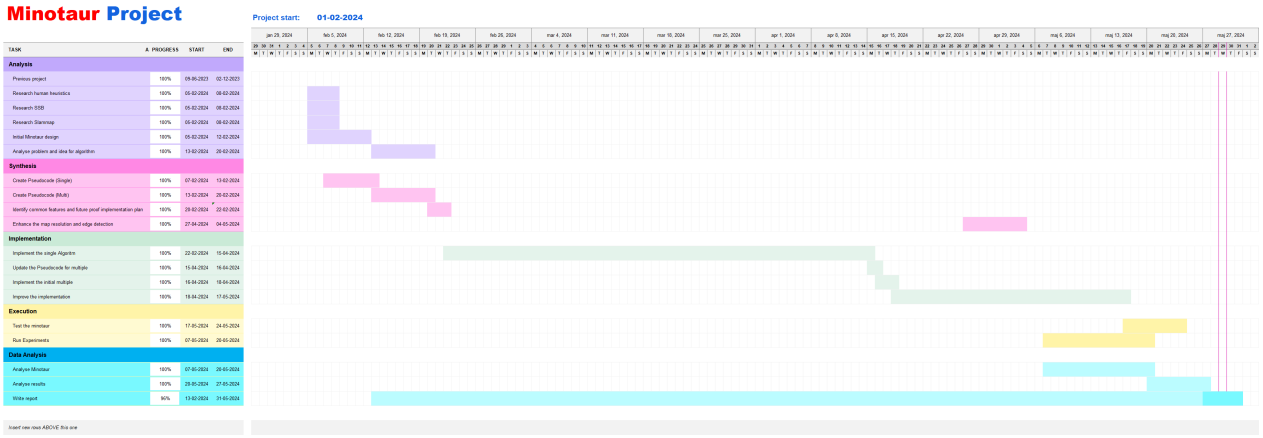

Figure 14: Final Gantt chart of the project

Before the final 8 weeks, we used a Kanban board to keep ourselves organized, with tasks that had their own sub-tasks as checkboxes. The Kanban board had the standard columns of: Backlog, sprint, in progress, in review, done, and dropped. This was done to keep track of the current progress as well as planning for the future with what was more essential than others. Tasks were added at semiregular meetings for that purpose, and assigned based on similarity to other currently assigned tasks, with occasional swaps to ensure everyone had a full understanding of the whole project.

When we started the project, we were annoyed by the state of MAES and how there seemingly was no global standard for the code and structure. This led us to follow the Common C# code conventions as our coding style, which can be found at https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions. We did not refactor all of MAES to follow the standard, as it was deemed a time-consuming task with few results for the project, though all features that we changed, added or refactored were written within the coding standards.

As opposed to previous projects, this one was much smoother and nicer to work on, largely because of our insistence that we meet up at the university every day instead of working from home. This massively increased motivation and enjoyment from the process, and allowed us to work with more coherence, and communicate more. This led to a better adherence to schedules and plans, better designs and ideas, and better implementation understood by all, as communication was constant and spirits were high.

Looking at the original Gantt Chart and the updated one, we can clearly see that we massively underestimated the amount of work the implementation would be. Working with already existing software presents a large amount of issues to work through when creating one's own implementation that does not quite adhere to the methods supplied by said software.

Beyond that, quite a few fields in the original plan never ended up being implemented, like the variable resolution or support for ablation testing of the algorithm. With a smaller idea of scope initially, we might have been able to better plan out the project, though the iterative nature of development means that course corrections like this are nearly inevitable. One can also see that we were able to complete more work in parallel towards the end of the project than we initially assumed, something that would also be useful to remember in the future.

Appendix C
Experimental Data

Strategy	Average Ticks	Successes	Timeouts	Success Rate	Fastest Success (Ticks)	Slowest Success (Ticks)
greed-size-100-LOS-robots-1-spawnapart	11,559.60	100.0	0.0	1.0	8,900	13,650.00
greed-size-100-LOS-robots-3-spawnapart	6,931.50	100.0	0.0	1.0	4,430	11,160.00
greed-size-100-LOS-robots-5-spawnapart	4,536.80	100.0	0.0	1.0	2,660	7,160.00
greed-size-100-LOS-robots-7-spawnapart	3,398.77	100.0	0.0	1.0	1,860	5,287.00
greed-size-100-LOS-robots-9-spawnapart	2,573.77	100.0	0.0	1.0	1,460	4,620.00
tnf-size-100-LOS-robots-1-spawnapart	20,321.60	84.0	16.0	0.84	13,560	22,820.00
tnf-size-100-LOS-robots-3-spawnapart	14,408.50	94.0	6.0	0.94	6,100	28,210.00
tnf-size-100-LOS-robots-5-spawnapart	12,391.40	92.0	8.0	0.92	4,470	29,820.00
tnf-size-100-LOS-robots-7-spawnapart	11,281.20	93.0	7.0	0.93	3,920	29,520.00
tnf-size-100-LOS-robots-9-spawnapart	10,927.70	94.0	6.0	0.94	2,800	34,950.00
minotaur-size-100-LOS-robots-1-spawnapart	12,705.70	100.0	0.0	1.0	7,590	17,130.00
minotaur-size-100-LOS-robots-3-spawnapart	7,854.40	100.0	0.0	1.0	4,550	13,650.00
minotaur-size-100-LOS-robots-5-spawnapart	4,334.00	100.0	0.0	1.0	2,210	6,310.00
minotaur-size-100-LOS-robots-7-spawnapart	3,229.70	100.0	0.0	1.0	1,660	6,150.00
minotaur-size-100-LOS-robots-9-spawnapart	2,530.30	100.0	0.0	1.0	1,330	3,930.00

Table II: Table over the various strategies with the configuration that uses a map size of 100, using LOS for communication, and spawning apart.

Strategy	Average Ticks	Successes	Timeouts	Success Rate	Fastest Success (Ticks)	Slowest Success (Ticks)
greed-size-100-LOS-robots-1-spawntogether	11,415.00	100.0	0.0	1.0	8,910	14,060
greed-size-100-LOS-robots-3-spawntogether	7,307.80	100.0	0.0	1.0	4,920	10,750
greed-size-100-LOS-robots-5-spawntogether	4,676.20	100.0	0.0	1.0	2,840	6,560
greed-size-100-LOS-robots-7-spawntogether	3,898.80	100.0	0.0	1.0	2,600	6,190
greed-size-100-LOS-robots-9-spawntogether	3,454.60	100.0	0.0	1.0	1,940	5,810
tnf-size-100-LOS-robots-1-spawntogether	18,950.10	94.0	6.0	0.94	13,560	32,910
tnf-size-100-LOS-robots-3-spawntogether	14,613.10	96.0	4.0	0.96	7,440	24,540
tnf-size-100-LOS-robots-5-spawntogether	12,358.20	98.0	2.0	0.98	6,660	26,950
tnf-size-100-LOS-robots-7-spawntogether	11,142.80	100.0	0.0	1.0	4,950	33,010
tnf-size-100-LOS-robots-9-spawntogether	13,242.10	94.0	6.0	0.94	5,050	29,330
minotaur-size-100-LOS-robots-1-spawntogether	12,892.50	100.0	0.0	1.0	8,420	17,510
minotaur-size-100-LOS-robots-3-spawntogether	7,617.80	100.0	0.0	1.0	4,250	13,530
minotaur-size-100-LOS-robots-5-spawntogether	4,738.80	100.0	0.0	1.0	3,150	6,910
minotaur-size-100-LOS-robots-7-spawntogether	3,770.30	100.0	0.0	1.0	2,440	6,040
minotaur-size-100-LOS-robots-9-spawntogether	3,261.20	100.0	0.0	1.0	1,930	4,770

Table III: Table over the various strategies with the configuration that uses a map size of 100, using LOS for communication, and spawning together.

Strategy	Average Ticks	Successes	Timeouts	Success Rate	Fastest Success (Ticks)	Slowest Success (Ticks)
greed-size-100-material-robots-1-spawnapart	11,559.60	100.0	0.0	1.0	8,900	13,650.00
greed-size-100-material-robots-3-spawnapart	5,357.30	100.0	0.0	1.0	3,590	8,010.00
greed-size-100-material-robots-5-spawnapart	3,417.70	100.0	0.0	1.0	1,920	6,200.00
greed-size-100-material-robots-7-spawnapart	2,552.37	100.0	0.0	1.0	1,510	4,760.00
greed-size-100-material-robots-9-spawnapart	2,217.30	100.0	0.0	1.0	1,400	3,559.00
tnf-size-100-material-robots-1-spawnapart	20,321.60	84.0	16.0	0.84	13,560	22,820.00
tnf-size-100-material-robots-3-spawnapart	14,633.50	94.0	6.0	0.94	5,010	32,340.00
tnf-size-100-material-robots-5-spawnapart	12,921.50	90.0	10.0	0.9	4,880	33,200.00
tnf-size-100-material-robots-7-spawnapart	10,790.30	93.0	7.0	0.93	3,910	30,090.00
tnf-size-100-material-robots-9-spawnapart	9,743.70	94.0	6.0	0.94	2,680	35,750.00
minotaur-size-100-material-robots-1-spawnapart	13,231.90	99.0	1.0	0.99	9,070	17,520.00
minotaur-size-100-material-robots-3-spawnapart	5,769.60	100.0	0.0	1.0	3,220	12,230.00
minotaur-size-100-material-robots-5-spawnapart	3,762.80	100.0	0.0	1.0	2,090	6,020.00
minotaur-size-100-material-robots-7-spawnapart	2,632.70	100.0	0.0	1.0	1,530	4,270.00
minotaur-size-100-material-robots-9-spawnapart	2,325.95	100.0	0.0	1.0	1,290	5,075.00

Table IV: Table over the various strategies with the configuration that uses a map size of 100, using material for communication, and spawning apart.