Robotic Exploration Algorithms in Simulated Environments with Python

Aniruddha Mysore and Sudarshan TSB

Department of Computer Science and Engineering
PES University, Bangalore, India
aniruddhamysore@pesu.pes.edu

Abstract. Swarm Robotics is inspired by the biological swarms of social insects such as ants and bees, where individuals performing basic tasks give rise to complex behavior. It utilizes a team of cooperating robots to perform tasks more efficiently than possible by isolated robots. In this research, we study the exploration of unknown indoor areas using robots that coordinate with each other. In particular, we implement the Reverse Nearest Neighbor coordination algorithm with certain modifications to account for real-world constraints. The library developed as part of this work provides scripts to help with robotic tasks for exploration and robotic arm control that can be used to set up simulation tools like VREP, without much prior experience thereby lowering the barrier for entry and making the robotics projects more accessible.

Keywords: multi-robot exploration, pedagogical robotics software

1 Introduction

The call for robot exploration arises from the need to mitigate the risks posed when accessing unexplored terrains. Human exploration of difficult terrains without proper analysis and mapping can result in accidents leading to loss of life. In fact, regions of exploratory interests can often prove to be hostile to humans, mandating robotic exploration. Some examples of such scenarios are underground exploration for mining, archaeology, and exploration of collapsed structures and others. Robots that are deployed in such terrains must be adequately equipped to address unforeseen environment constraints and effectively coordinate with each other.

Although methods for robot-driven spatial navigation and mapping are well researched, real-world implementation often proves to be a challenge due to the dynamic nature of real-world environments. Addressing this challenge, we posit a novel method for spatial navigation and mapping of unexplored terrains using swarm robotics. Swarm robotics is a paradigm that capitalizes on the advantages of multiple robots while doing away with the complexity involved in individual specialization. A group of robots working collectively enables efficient coverage of the area by exploiting the properties of a swarm.

Our research focuses on closely examining some of the common limitations in the practical applications of state-of-the-art swarm exploration algorithms and propounds effective solutions for handling them. In this work, we develop and test several real-time scenarios for swarm exploration using a 3D robotics simulator. This solution employs the Reverse Nearest Neighbor (RNN) algorithm [1] and identifies the constraints introduced when the same is applied to simulations that closely imitate real-world scenarios. We implement RNN to coordinate the direction and movement of each robot in the swarm. This algorithm is coupled with a path-finding algorithm to generate a map of the explored terrain. The A-Star algorithm is used for finding the most efficient path to navigate through the landscape.

The mapping of an area can be broadly viewed as 3 sub-tasks - mapping, path-planning, and coordination. Each robot must be able to survey its immediate environment to construct a local map (mapping) using the current position as a frame of reference. In indoor environments there is the additional responsibility of *localization* - determining the robot's current location relative to a global frame. Localization ensures that while coalescing the local maps to form the global structure a single frame of reference is used. Path-planning involves identifying the route to the next point to explore. The potency and robustness of swarms are made possible due to the emergent behavior that arises when multiple robots act in coordination. Hence the algorithm used to coordinate the swarm is critical to the exploration task.

The setup for robotics simulation projects has traditionally been a cumbersome step. The TRS (Teaching/Learning Robotics with Simulator) package [2] published by Detry et. al. was a step forward that vastly simplified creating robotics projects by providing scaffolding in the form of simulator scene files and control scripts. TRS lowers the barriers for entry to robotics and greatly assists in pedagogical activities. However the original TRS package required the use of MATLAB to run the robot control scripts. In this work, we present a tool written in Python that provides the same functionality of TRS, but does not require paid software. In summary:

- In this work, we study the exploration of a new area by a robot swarm using a 3D robotics simulator. In particular, we investigate the Reverse Nearest Neighbor algorithm and identify constraints involved when applying the same in a physics-world simulation.
- We propose a post-processing step to the A* path-finding algorithm in order to fulfill these constraints.
- Finally, we introduce a Python package that allows using the TRS - Teaching Robotics With A Simulator tool that allows the ease of setup and use the original package in an open-source, more widely supported language.

Based on this, the paper is organized into three parts. Note that in this paper we use the terms CoppeliaSim and VREP interchangeably - both refer to the robot simulator that was previously named VREP and the latest version has been renamed to CoppeliaSim.

2 Related Work

Distributed or collective systems are an important subset of the field of robotics. Teams of robots have been observed to perform many standard tasks [3] better than single robot models. Swarm robotics is a paradigm of multi-robot systems that is heavily inspired by nature, especially from the behavior observed among social insects [4] such as ants and bees. The task of exploration benefits greatly from the usage of robot swarms due to its inherent nature.

The exploration and mapping of an area by an autonomous agent is a well-studied domain. In their seminal work, Thrun et. al [5] discuss several algorithms, map representation techniques, and challenges associated with this problem. Yamauchi et. al [6] proposed the original frontier-based exploration technique which is found in many later works. Rajesh et. al[7] use a probabilistic model for the selection of frontier-cells. Particle Swarm Optimisation(PSO) [8] forms the basis for many swarm coordination algorithms. Several variants of PSO have been proposed that outperform the original - Random Drift PSO (RDPSO) has been found to be highly efficient in a simulated swarm consisting of 14 robots [9].

Previous works represent the map or exploration in areas in several ways, with a square grid being the most widely picked. Hexagonal grid representations have been found to reduce revisiting of already-seen areas of the map [1]. Wurm et al. [10] proposed a method of coordinated exploration where the target area is represented contiguously and is partitioned into segments to minimize overall mission time. Another approach used by the bacteria chemotaxis algorithm [11] involves dividing a continuous environment into Voronoi cells. The representation of the target area is also affected by the communication method used by robots since the individuals in a swarm need to exchange this information. Sheng et. al have proposed an algorithm for exploring areas when under the constraint of limited-range communication [12], which allows for lightweight map-synchronization using a number based representation of the explored map. Coordination approaches either assume a completely decentralized swarm model or allow for the existence of one or more "leaders" which are responsible for the communication of commands to the other individuals in the swarm [13]. The presence of explicit leaders also allows control by human operators [14] when a manual override is needed. The relative positioning of these leaders has been found to influence the swarm's behavior [15]. The weighted Reverse Nearest Neighbor(RNN) algorithm has been shown to outperform other PSO techniques [1]. Hence we have chosen to use this procedure in this work.

Exploration also forms a core sub-task in domains such as search and rescue operations that use robots. [16]. Junior et. al have proposed the use of wave algorithms[17] to manage the execution of a sequence of sub-tasks for the collective navigation of swarms. Batch bayesian search procedure [18] has also been proposed for use in decentralized models for robot search and rescue. Recently there has also been a rise in the use of reinforcement learning techniques to control robots [19].

3 Analysis of Exploration Algorithm

3.1 Constraints

This work uses a modified version of the RNN exploration algorithm. The latter follows a two-rule process where a robot visits its immediate neighbors if possible, and the remaining frontier cells are considered only if all immediate neighbors have already been explored. This approach is well suited for a simplified grid world, however, there are other factors to be considered while making a real-world based simulation.

In a grid world, a robot only sees its immediate neighbors, however a vision sensor, such as the laser scanner allows a robot to see a large, non-uniform area, and except in the very beginning of the exploration process, all immediate neighbors would usually be marked as explored. Hence we propose skipping this step completely, and only consider the nearest frontiers while deciding where to move a robot next. A* algorithm is used to compute each robot's path to the closest frontier. This algorithm also posed some problems during simulation, which is discussed in the following sections.

3.2 Path-finding with A*

$A*$ is a search algorithm [20] that computes the the shortest path between an initial state (source) and a final state (destination or goal). The following parameters are used to weigh candidate paths while making a decision:

- g : The cost of moving from the initial cell to the current cell. It is the sum of all the cells that have been visited since leaving the first cell.
- h : The heuristic value. It is the estimated cost of moving from the current cell to the target cell. The actual cost cannot be calculated until the target cell is reached - hence, h is the estimated cost.
- f : the sum of g and h. $f = g + h$

A* makes decisions by taking the f-value into account. The algorithm selects the smallest f-valued cell and moves to that cell. This process is iterated until the target cell is reached.

3.3 Extracting Way-points from A* Path

One of the important characteristics of the path formed by A* is that it is end-to-end. In a grid, since paths are composed of a sequence of cells, each defined by an index, using the output from A* directly entails asking the robot to move to cell 1 and then cell 2 and so on, according to the path.

A problem that arises from this, is that while the robots represent the map as a grid internally, the simulation environment and the real-world are modeled with real-valued Cartesian coordinates. In such a world, the vanilla A* path that directs a robot to go to the equivalent cell centers will not be the most efficient

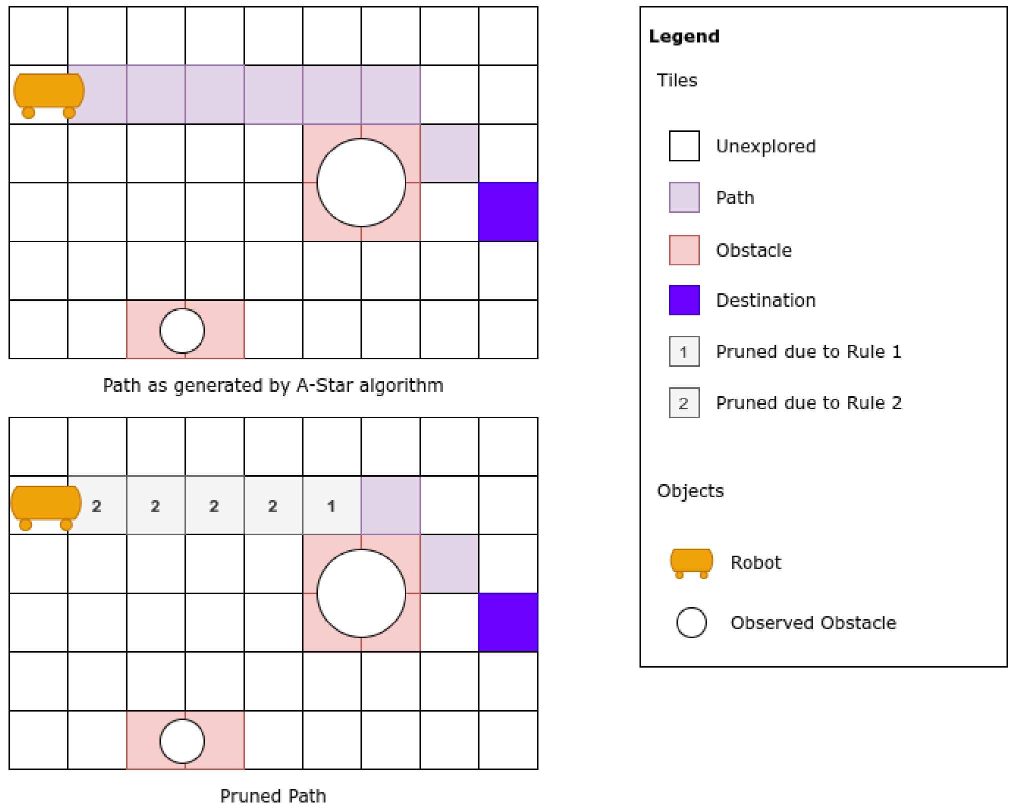


Fig. 1. Pruning

and cause delays. It would be much more effective to reduce the number of points in the path and preserve only the necessary way-points to guide the robot.

The robot control model equations stated later in the Actuation section are used to drive the robot to the way-points. These equations (Section 4.4) describe the actuation values required to move a robot on a straight-line path between two points in a Cartesian system. They cannot be used in isolation without the path-finding algorithm since they require no obstacles to be present between the two points. In the system we have implemented, A* is used to build the shortest path to the goal and then is decomposed into way-points where each point is in line-of-sight with its two adjacent points.

In order to prune the A* path and extract the minimal number of way-points, we propose the Pruning Algorithm described in Alg. 1. Our approach formulates two rules to flag points which can be safely removed from the path:

– **Rule 1:** If two consecutive points in the original path have the same X or Y coordinate, flag the first point

– **Rule 2:** If a point in the original path has no obstacles as a neighbor, flag the point.

Algorithm 1 Path Pruning Algorithm

Inputs: Path formed by A* algorithm consisting of multiple points
Outputs: Pruned Path

```
 1: procedure PRUNE-PATH(path)
 2:     pruned_path ← [ ]
 3:     for i = 0 to path.length − 1; step = 1 do
 4:         x_i, y_i ← path[i]
 5:         x_{i+1}, y_{i+1} ← path[i + 1]
 6:         if x_i ≠ x_{i+1} and y_i ≠ y_{i+1} then
 7:             flag ← False
 8:         for n ∈ neighbors(path[i]) do
 9:             if n =obstacle then
10:                 flag ← False
11:         if not flag then
12:             pruned_path.append(path[i])
13:     last ← path.length − 1
14:     pruned_path.append(path[last])
15:     return pruned_path
```

An example of this algorithm at work is depicted in Figure 1. Both the original and pruned paths are shown in purple, and the grey cells are cells that were a part of the original path but have been removed by the algorithm.

Note that the method proposed here always preserves the final point in the original path, so the smallest possible path length is 1 - where the way-point is the goal itself. This scenario occurs (and with high frequency!) when there are no obstacles between the goal and the starting point.

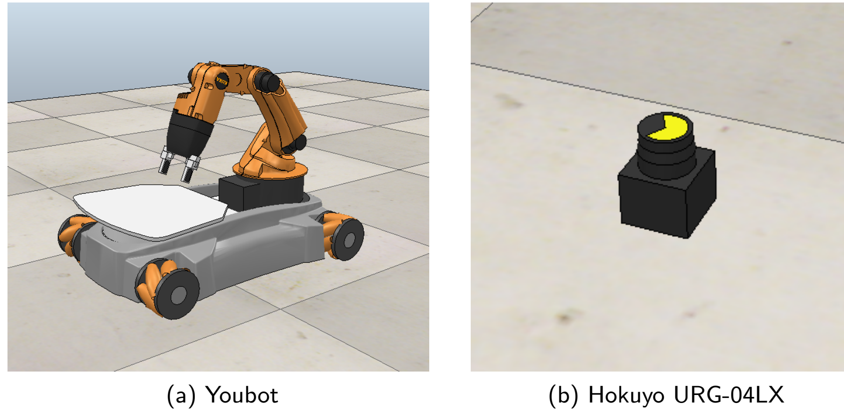
4 Adapting RNN

4.1 System Design Considerations

This section discusses the decisions made while designing how the swarm exploration will function.

Assumptions. All systems have a set of boundaries and conditions within which they are designed to perform their regular operation. The assumptions of the exploration method we discuss are:

– The floor or the surface of the environment upon which the robot moves is flat, with no undulations or slopes.



(a) Youbot (b) Hokuyo URG-04LX

Fig. 2. Simulation hardware

- All robots are able to communicate with each other via WiFi.
- The environment does not include powerful light sources or other objects that can confuse the laser scanner and cause it to report incorrect values.

Exploration Environment. The second design decision made relates to the area that is to be explored and how the same is modeled internally by the robot. We use an enclosed area created using the simulator that is bounded by walls and has the floor plan of an average home. There are obstacles present for the robot to detect, avoid, and place in the map which the swarm is creating for this space.

Each robot sees the exploration area as a square grid of fixed size. This square grid is further partitioned into a number of square regions called subareas. Each cell in the grid can be one or more of several types - unexplored, visited, obstacle, frontier clearance, or in-field-of-view.

4.2 Simulation Hardware

Robot. The robot used in the simulations is the KUKA youBot - a mobile platform with a robotic arm. The chassis has four Mecanum wheels with rolls mounted around the circumference at a 45° angle to the wheel's plane, enabling the youBot to combine translational and rotational velocities so as to make omni-directional motion possible, including sideways and diagonal motion.

Hokuyo Laser Scanner. A laser scanning sensor is mounted on the robot to be used for collision detection and mapping of the area. In particular, we use the Hokuyo URG-04LX Scanning Laser Rangefinder for this simulation. This sensor is tailored for small robotic applications and is commonly used in many tasks. It can obtain measurement data up to a distance of 5 metres with millimetre resolution in a 240º field.

4.3 Architecture

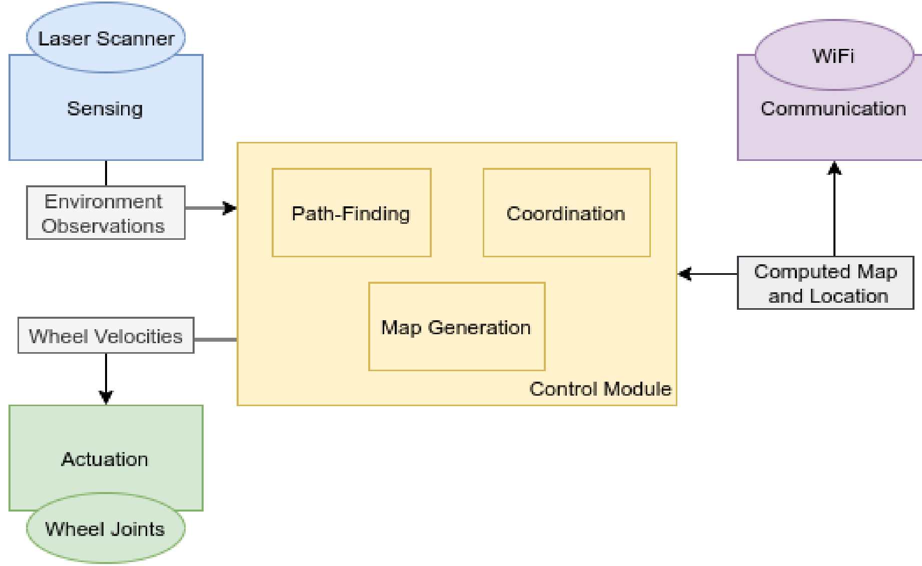


Fig. 3. System Architecture Model

Each robot in the swarm performs various operations and these have been structured into the four modules represented in the architecture diagram depicted in Figure 3. The modules depicted are Sensing, Actuation, Communication, and Control modules. The Control module can be logically divided into the Map generation, Coordination, and Path Finding modules. The following subsection discusses the functionality of each module in brief.

Project Organization. The sensing module is responsible for initiating the laser scanner mounted on the robot and reading the data returned from the scanner. The sensing module also parses the raw sensor data and converts it into a set of coordinates. This also involves transforming coordinates from robot-relative to world coordinates.

The control module represents the main computation cycle where all other robot operations are invoked. It runs the initialization steps and then runs in an infinite loop until the exploration is complete. We have logically divided the control module into 3 sub-modules that handle the Map Generation, Path Finding, and Coordination respectively. Map Generation is responsible for combining processed sensor inputs of the current cycle with the previously known environment information to generate an up-to-date map. The widely used A* algorithm forms the core of the path-finding sub-module. There are several modifications

made to the original A* algorithm to account for constraints as discussed in later sections.

Exploring cooperatively with other individuals in the swarm requires the use of a coordination algorithm, and we implement a modified version of the Reverse Nearest Neighbor approach. Coordinating requires that robots signal each other via a communication medium. The communication module represents the controls every robot uses to send and receive information to each other. The communication medium used is WiFi, which supports the general requirements of this project.

The actuation module is responsible for solving the robot-dynamics equations to compute the correct actuation values to send to the wheel motor controllers.

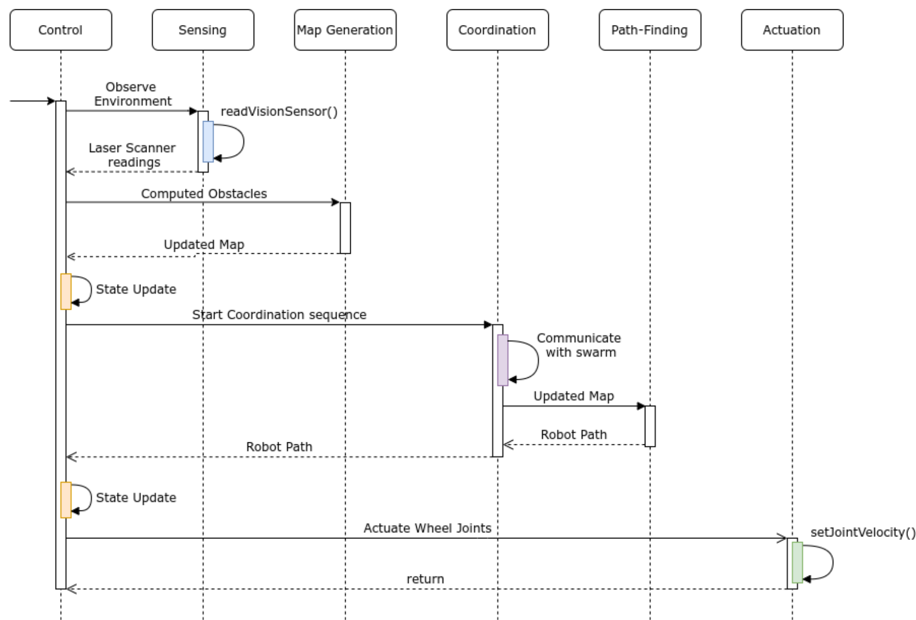


Fig. 4. Sequence Diagram

Sequence Diagram. Figure 4 visualizes the flow of logic within the system. The sequence described here is executed from beginning to end by each robot in a loop until the swarm has finished exploration of the area.

Each execution cycle begins with the robot observing its surroundings with the laser scanner. This is handled by the sensing module which makes a $readVisionSensor()$ call to the simulator which then computes and returns the simulated sensor readings. The parsed sensor data is used by the map generation sub-module to update the robot's map with the obstacles detected and areas explored.

At this stage, the robot checks its state and updates it if needed according to the Reverse Nearest Neighbor algorithm. The state updation procedure is explained in the next chapter. Once the map is updated, the robot broadcasts its updated map to all other robots in the swarm over WiFi. Similarly, all other robots in swarm also broadcast their current map. The map is used by the path-finding algorithm to calculate the robot's trajectory to the closest frontier point to explore next. Once the path is generated, another state check and update takes place. Finally, the robot's actuation module calculates the wheel velocities and sends it to the simulator with the $setJointVelocity$ command. This updates the speed and direction of the robot's wheels in the simulation causing the robot's momentum to change.

4.4 Implementation

Choice of Simulator: CoppeliaSim/VREP. All simulations in this project have been developed and demonstrated using CoppeliaSim. The simulator supports different physics engines and the configuration for this project uses the Bullet Physics engine. CoppeliaSim offers two main ways to control objects - via an embedded script or with a Remote API client. We implement the robot coordination and mapping logic using Python and we use the RemoteAPI to connect to the simulator.

TRS. TRS (Teaching/Learning Robotics with Simulator) [9] is an open-source recipe that allows quickly setting up an environment to experiment with different robotics concepts. The environment that TRS provides consists of a set of Matlab scripts, and a V-REP file that models a mobile robot and a building floor. TRS provides a software skeleton and code examples for control, navigation, vision, and manipulation algorithms.

We have used the building floor scene provided by TRS as the main exploration area for the project. It is a realistic area consisting of common household obstacles. The Matlab scripts included with this scene initialize the robot, its wheel joints, arm joints, and the laser scanner. They also provide functions that ensure actuation values do not exceed safety limits. We have ported these scripts to Python to fully utilize the convenience provided by TRS. We have also released the ported codebase as an open-source repository to benefit other researchers.

Map Abstractions. The simulator uses a real-valued Cartesian coordinate system. It is not feasible for the robots in the swarm to represent the map of the exploration area in this way - it would increase both complexity and computational load. Hence, internally the robots construct an abstraction of the map in the form of a square-grid where each cell represents a 0.2×0.2 metre tile of the exploration area. Internally, the grid is stored as a 2-dimensional Numpy array.

In order to convert Cartesian coordinates into array indices and vice-versa, we use the following rule.

$$index = (cartesian_coordinate + map_size)/tile_size$$

- **Cartesian coordinate**: A pair of real-valued coordinates (x,y) representing a point in the exploration area.
- **Index**: A pair (i,j) corresponding to the index positions of the square tile in which the Cartesian coordinate is located.
- **Map Size**: The size of the entire simulation area. The scene provided by TRS has a side-length of 7.6 metres.
- **Tile Size**: The size of an area represented by a single cell. We consider the tile-size to be 0.2 metres.

Neighborhood Scanning. The Hokuyo laser scanner has a 240-degree field-of-view with a range of 5 metres. In order to scan the robot's neighborhood, remote API calls need to be made to the two 120 degree sub-sensors. Each sub-sensor returns a raw data packet that needs to be parsed to separate the sensor reading from the metadata. The sensor readings are reshaped into a 4xN matrix consisting of $(x, y, z, distance\ to\ sensor)$ values for each of the N points scanned.

Next, we compute an obstacle mask flagging readings for which the distance-to-sensor component is lesser than 5 metres. The range of the scanner is 5 metres, so a point which is closer than 5 metres will be an obstacle. The *Points matrix* is calculated twice, once for each 120-degree sub-sensor. These are then combined into a single matrix by concatenating horizontally. Similarly, the associated obstacle masks for the two Points matrices are also combined by concatenating horizontally. The parsed coordinates up to this point have been expressed as sensor-relative coordinates. We use the following steps to express these in world coordinates.

- The world coordinates of the robot are used to calculate translation matrix $transl_R$, and the world-relative orientation of the robot is used to calculate rotation matrices $rotx_R$, $roty_R$, $rotz_R$. The dot product of these matrices is the robot-world transformation $trf_{robot-world}$.

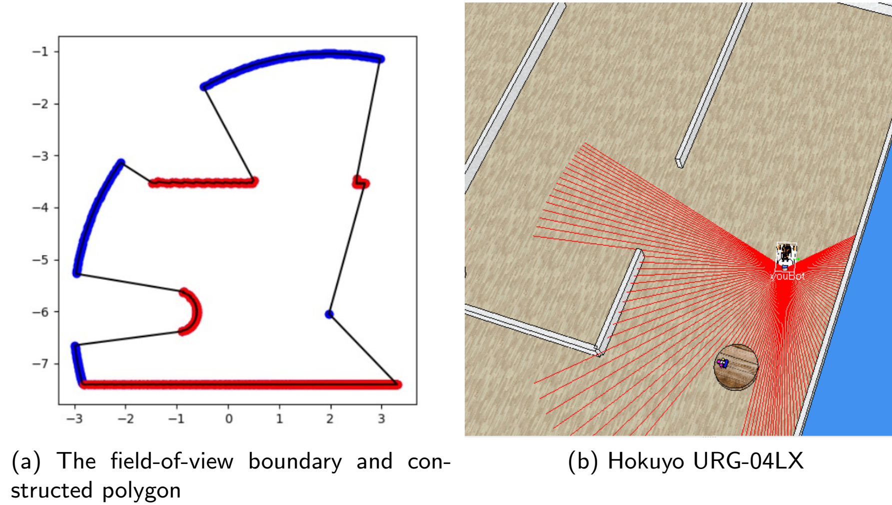
$$trf_{robot-world} = transl_R \cdot rotx_R \cdot roty_R \cdot rotz_R$$

- The robot-world transformation is used to calculate the world-coordinates of the laser scanner.
- The robot-relative position and orientation of the laser scanner are used to calculate the matrices $transl_{LS}$, $rotx_{LS}$, $roty_{LS}$, $rotz_{LS}$. The dot product of these matrices is the sensor-robot transformation $trf_{sensor-robot}$.

$$trf_{sensor-robot} = transl_{LS} \cdot rotx_{LS} \cdot roty_{LS} \cdot rotz_{LS}$$

- The points detected by the laser scanner readings are expressed relative to the sensor itself. They are first transformed to robot-relative coordinates using $trf_{sensor-robot}$ and then to world coordinates using the world location of the laser scanner, which is computed using $trf_{robot-world}$.

Figure 5b shows red lines emerging from the robot - this is the 240-degree field-of-view of the Hokuyo laser scanner.



(a) The field-of-view boundary and constructed polygon

(b) Hokuyo URG-04LX

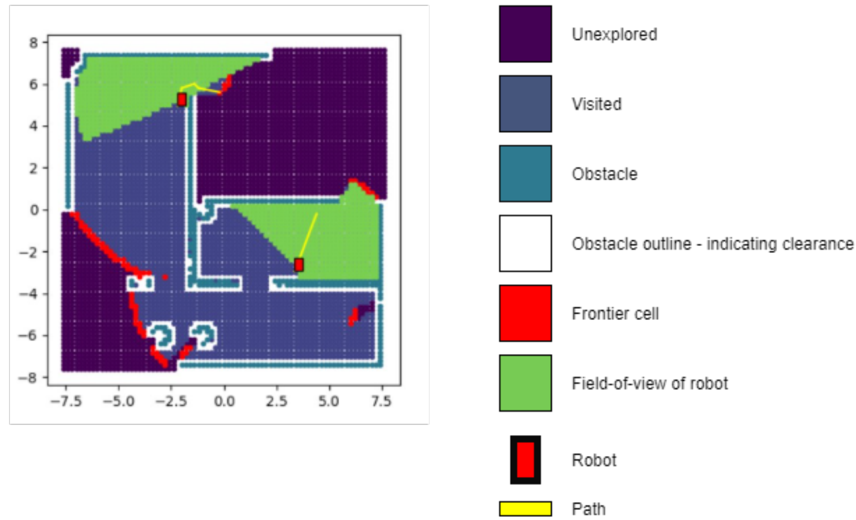
Fig. 5.

Map Generation and Plotting. The scanning process results in a set of coordinate points which represent the boundary of the field-of-view of the robot. Some of these points are obstacle points, identified by the obstacle mask. Robots in the swarm use a grid internally to represent the map. Using the boundary points of the field-of-view, we construct a polygon and all grid points within this polygon are marked as visited. The Shapely library offers a vectorized implementation of the *contains* function which efficiently determines which points in a large set of points lie within a polygon.

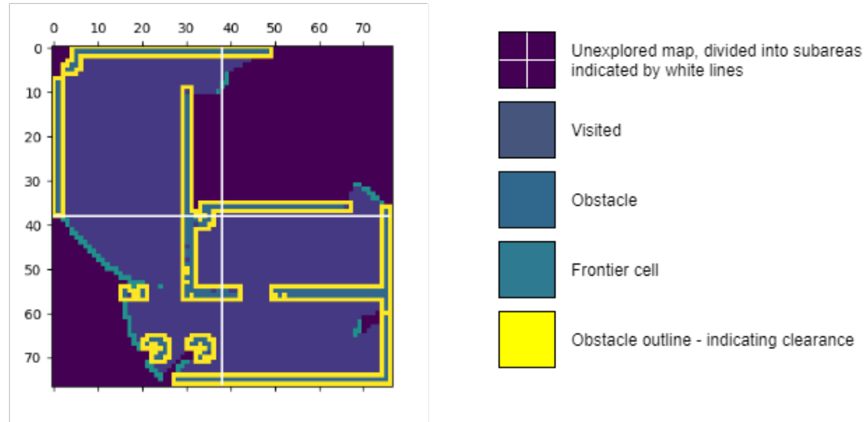
The obstacle mask is used to mark the obstacles in the grid, and all neighboring cells of an obstacle are marked as a special *clearance* cell. The path planning algorithms discussed in the next section ignore these cells. We use this approach to reduce collisions of the robots with obstacles. The field-of-view boundary as well as the constructed polygon for the robot in Figure 5b is depicted in Figure 5a. The red and blue dots form the boundary, where the red indicates obstacle points. The black outline is the polygon calculated using the boundary points.

The map generation procedure identifies frontiers as boundary points of a robot's field-of-view which (1) are not obstacles and (2) have not been explored before. The frontiers are the points that are considered by the path-finding algorithm when deciding the future trajectory of the robot. Figure 6 shows a plotted map.

Figure 6a shows the individual robots and their immediate vision. Figure 6b depicts the computed map at the same scene including the 4 subareas which are visible as white line partitions.



(a) Robot Locations and Field of View



(b) Computed Map

Fig. 6. Detailed scene-map with robot positions, view, and trajectory, Constructed map, with subareas

Coordinating with the Swarm. We make use of the same coordination logic used by RNN. At any given instance after initialization, each robot is either in the *Exploration* state or the *Moving* state. In the former, the robot continues exploring the sub-area it is currently in, and in the latter, the robot moves to an unexplored sub-area.

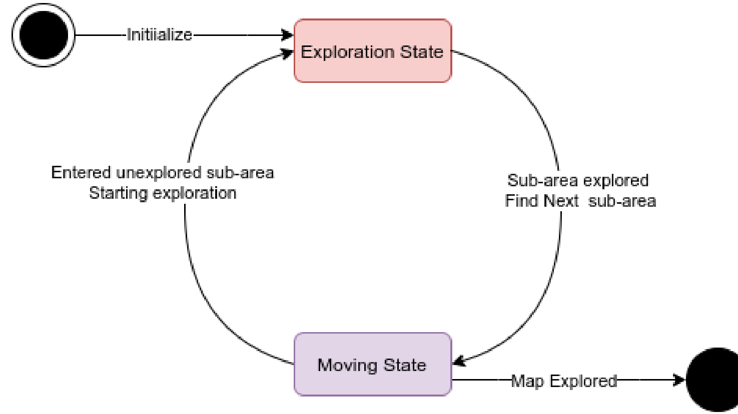


Fig. 7. Robot coordination states

In the beginning, the robot is in the *Exploration* state. It explores the sub-area that it is present in until there are no accessible frontiers left. Then, the robot switches to the $Moving$ state and moves to another sub-area. The next sub-area is chosen such that it has a lesser number of robots currently approaching it to reduce the number of times a region is re-explored. This is achieved by the addition of a weight term w to the path-finding algorithm while calculating the shortest path, where

$$w = \alpha * num_robots_approaching$$

Once the robot is done moving, it switches back to exploring. The two states, as well as their updation logic, is visualized in the state diagram in Figure 7.

Communication Logic. Coordination between robots requires that each robot knows at all times where its peers are, and where they are going next. In the simulation, all robots run a server process that listens to incoming signals from other individuals in the swarm. Each robot broadcasts its current position and trajectory on each cycle. The broadcast-and-listen procedure works in the following way.

- The robot R_i sends $N-1$ HTTP requests to the IP addresses of each of the other $R_{.}.. R_{N-1}$ robots at a predetermined TCP port.
- Each robot R_j has a background HTTP server that is listening on the TCP port. When an HTTP request from another robot R_i arrives, this background process parses the request and updates the *swarm_data* file with R_i's latest position and trajectory.
- When any robot R_k enters the $Moving$ state and needs the trajectory information of the rest of the robots in the swarm to calculate the weight term w, it uses the data present in the *swarm_data* file.

We assume that all robots can communicate with others.

Actuation. The Youbot's Mecanum wheels are an example of a four-wheeled Holonomic drive. In order to drive the robot, three-component velocities are specified.

- v_x - Forward/Backward velocity
- v_y - Left/Right velocity
- ω - Rotational velocity

These component velocities are calculated using the equations described below and are communicated to the simulator.

- **Robot Position**
$$position = (x_p,\ y_p)$$
- **Robot Orientation**
$$orientation = (\alpha,\ \beta,\ \gamma)$$
- **Goal Position**
$$goal = (x_g,\ y_g)$$
- **Forward/Backward velocity**
$$v_x = (x_g - x_p)\ Sin(\gamma)\ + (y_g - y_p)\ Cos(\gamma)$$
- **Left/Right velocity**
$$v_y = (x_g - x_p)\ Cos(\gamma)\ + (y_g - y_p)\ Sin(\gamma)$$
- **Rotation Velocity**
$$\omega = nwp + \delta\gamma\ *\ (\ |\delta\gamma| > 0.04\ )$$
$$nwp = -\left(\gamma + 3\frac{\pi}{2}\right)\ +\ tan^{-1}\left(\frac{y_g - y_p}{x_g - x_p}\right)$$
where nwp is Next Waypoint Direction.

4.5 Result Analysis

Run-time Comparison. Figure 8 depicts the different simulation configurations that were used to analyze the final result. We now examine the number of iterations of the sense-compute-actuate loop that are required to finish exploring the map changes. We study the number of iteration-steps rather than the absolute time to negate the influence of the current load on the system.

The aggregated result of the simulation runs has been depicted in the graph in Figure 9.

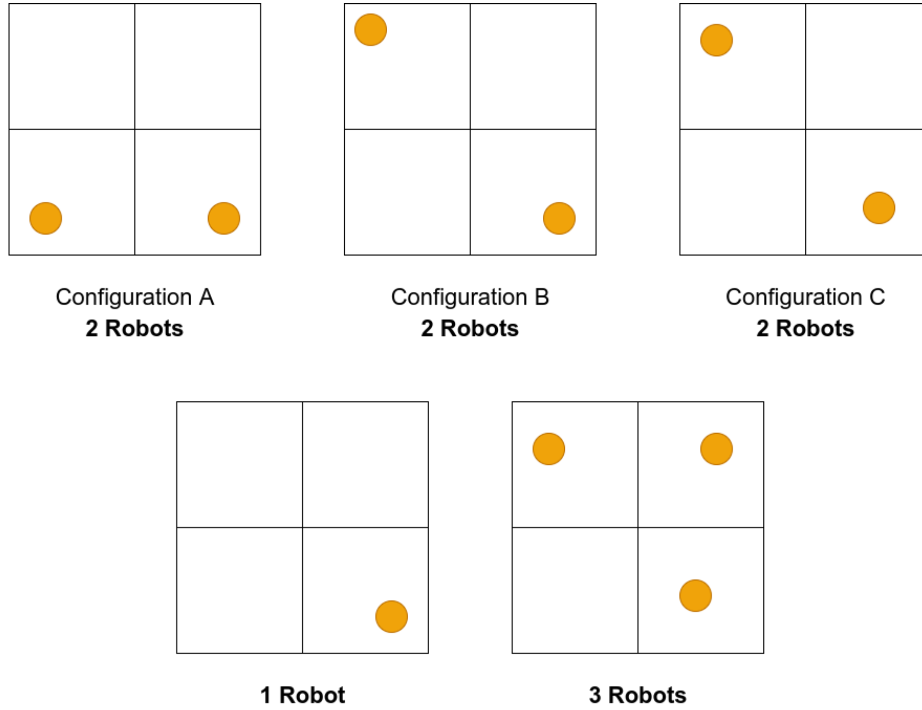


Fig. 8. Configurations Used for Result Analysis

In general, the number of iterations taken to explore the map decreases with an increase in the number of robots. It is clear that when the same number of robots used in different positions such as in the A, B, and C configurations, the iterations needed to explore also vary widely. Hence the starting positions of the individual robots are an important factor in exploration.

Between configurations B and C, there is only one small change - one of the robots has been moved a few metres towards the center of the map, which is an area with many obstacles and walls on 3 sides. Yet the number of iterations required to explore changes - this also shows how the distribution of obstacles has a significant influence on the exploration time.

From the analysis of these results we find that the number of iterations taken to explore the map, and hence the efficiency of the algorithms we propose, is heavily dependent on several factors:

- The size of the exploration area.
- The number of obstacles present and their distribution.
- The number of robots used for exploration.
- The starting positions of the robots.
- The hardware used to run the simulation.

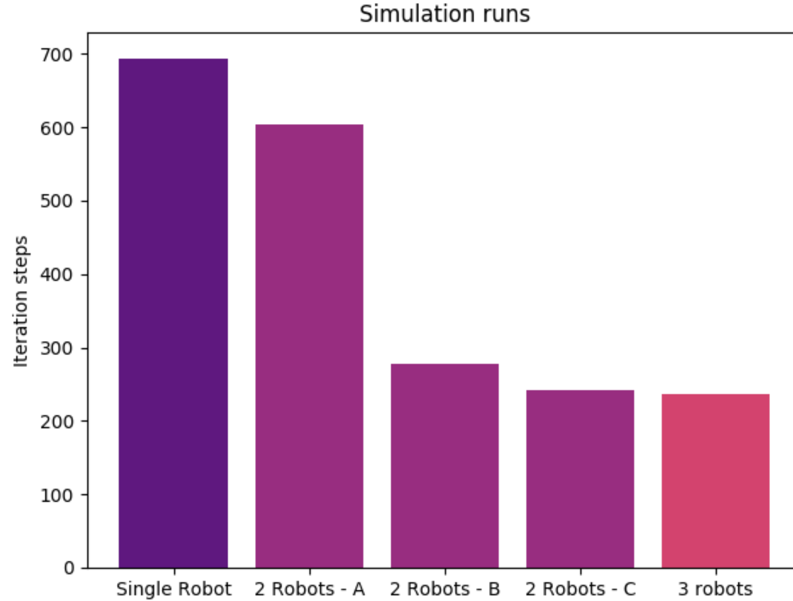


Fig. 9. Results Graph

Notes on Scalability. The methods discussed in this work have been tested on swarms consisting of one, two, and three robots in the robot simulator. To test exploration approach with a very large number of robots, we would also need to greatly increase the size of the exploration area, as well as the number of obstacles in the map. This will require an infeasible amount of manual setup-time and compute power, which can be explored in the future.

The scalability of the original Reverse Nearest Neighbor algorithm has been tested on swarms of hundreds of robots in a simple grid-world environment. The addition of pruning algorithm reduces number of iteration steps required to complete exploration in the simulator. The average time for two robot configuration is 417.12 iteration cycles with pruning and nearly 1.5 times without the pruning step in original RNN algorithm - 642 iteration cycles.

Limitations.

- Laser scanners are effective when the surface is two-dimensional, so slopes and undulations on the floor surface may lead to errors.
- When there are a large number of unexplored frontiers, computing the path-finding procedure for each frontier might lead to a heavy load on the system.

5 Package for Python

5.1 TRS

It is hard to bootstrap simulation robotics projects for beginners. There are many available simulators like Gazebo and VREP which offer powerful capabilites, but the knowledge of integrating 3D scenes in these with a programming language from scratch is siloed which results in a high barrier for entry into modern robotics tools for beginners.

TRS is an open-source package that provides a quick-setup for conducting robotics studies. It provides simulator scene files that include models of robots and an indoor environment as well as Matlab scripts that allow various robot operations. TRS provides the scaffolding and code examples for control, navigation, vision, and manipulation algorithms.

We have used the building floor scene provided by TRS as the main exploration area for the project. It is a realistic area consisting of common household obstacles. The Matlab scripts included with this scene initialize the robot, its wheel joints, arm joints, and the laser scanner. They also provide functions that ensure actuation values do not exceed safety limits.

5.2 TRS Python

A drawback to TRS is the fact that all associated scripts are based on Matlab. We have ported these TRS scripts to Python. Python is a free and open-sourced programming language, while Matlab requires a license to use. Python is more widely-used than Matlab and has been the most popular language in recent years for Machine Learning and AI applications. We hope that the Python version will allow more people to conduct research into the field of robotics. We have also released the ported code-base as an open-source repository to benefit other researchers.

6 Conclusion

During accidents, disasters, or other cases where humans cannot access a certain area, technology can aid in the mapping of these unknown regions. The generated maps can be used by other robots and human personnel for a variety of different applications. This is the main motivation for studying the exploration and mapping of unknown areas using swarm robotics. In this work, existing exploration algorithms are compared and an efficient swarm exploration method based on the RNN technique is developed. A system is modeled and implemented that performs the task of exploring an area using the above method. The approach is simulated using the CoppeliaSim robot simulator. Finally, we present TRS-Python, an open-source tool that can provide the organization and scaffolding required to quickly bootstrap an advanced robotics project using a simulator.

7 Acknowledgments

The authors acknowledge Kaushik M Reddy, Supreeth Subramaniam and Sindhu Gouda for their assistance in the literature survey phase of this project. We would also like to thank Poulami Sarkar for lending support to this work.

References

1. A. Datta, R. Tallamraju, and K. Karlapalem, “Multiple drones driven hexagonally partitioned area exploration: Simulation and evaluation,” in *Proceedings of the 2019 Summer Simulation Conference*, ser. SummerSim ’19. San Diego, CA, USA: Society for Computer Simulation International, 2019.
2. R. Detry, “Trs: An open-source recipe for teaching/learning robotics with a simulator.”
3. L. Bayındır, “A review of swarm robotics tasks,” *Neurocomputing*, vol. 172, pp. 292 – 321, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231215010486
4. I. Navarro and F. Matía, “An introduction to swarm robotics,” *International Scholarly Research Notices*, vol. 2013, pp. 1–10, 2013.
5. S. Thrun, “Robotic mapping: A survey,” in *Exploring Artificial Intelligence in the New Millenium*, G. Lakemeyer and B. Nebel, Eds. Morgan Kaufmann, 2002, to appear.
6. B. Yamauchi, “A frontier-based approach for autonomous exploration,” in *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA’97. ’Towards New Computational Principles for Robotics and Automation’*, 1997, pp. 146–151.
7. Rajesh M., G. R. Jose, and Sudarshan T.S.B., “Multi robot exploration and mapping using frontier cell concept,” in *2014 Annual IEEE India Conference (INDICON)*, 2014, pp. 1–6.
8. J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Proceedings of ICNN’95 - International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948 vol.4.
9. M. S. Couceiro, P. A. Vargas, R. P. Rocha, and N. M. Ferreira, “Benchmark of swarm robotics distributed techniques in a search task,” *Robotics and Autonomous Systems*, vol. 62, no. 2, pp. 200 – 213, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S092188901300208X
10. K. M. Wurm, C. Stachniss, and W. Burgard, “Coordinated multi-robot exploration using a segmentation of the environment,” in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 1160–1165.
11. B. Yang, Y. Ding, Y. Jin, and K. Hao, “Self-organized swarm robot for target search and trapping inspired by bacterial chemotaxis,” *Robotics and Autonomous Systems*, vol. 72, pp. 83 – 92, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889015000937
12. W. Sheng, Q. Yang, J. Tan, and N. Xi, “Distributed multi-robot coordination in area exploration,” *Robot. Auton. Syst.*, vol. 54, no. 12, p. 945–955, Dec. 2006. [Online]. Available: https://doi.org/10.1016/j.robot.2006.06.003
13. S. A. Amraii, P. Walker, M. Lewis, N. Chakraborty, and K. Sycara, “Explicit vs. tacit leadership in influencing the behavior of swarms,” in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 2209–2214.

14. P. Walker, S. Amirpour Amraii, N. Chakraborty, M. Lewis, and K. Sycara, "Human control of robot swarms with dynamic leaders," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 1108–1113.
15. R. Tiwari, P. Jain, S. Butail, S. P. Baliyarasimhuni, and M. A. Goodrich, "Effect of leader placement on robotic swarm control," in *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, ser. AAMAS '17. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2017, p. 1387–1394.
16. E. Magsino, F. Beltran, H. Cruzat, and G. Sagun, "Simulation of search-and-rescue and target surrounding algorithm techniques using kilobots," in *2016 2nd International Conference on Control, Automation and Robotics (ICCAR)*, 04 2016, pp. 70–74.
17. L. S. Junior and N. Nedjah, "Efficient strategy for collective navigation control in swarm robotics," *Procedia Computer Science*, vol. 80, pp. 814 – 823, 2016, international Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050916308468
18. P. Ghassemi and S. Chowdhury, "Decentralized informative path planning with exploration-exploitation balance for swarm robotic search," 2019.
19. J. Yang, X. You, G. Wu, M. M. Hassan, A. Almogren, and J. Guna, "Application of reinforcement learning in uav cluster task scheduling," *Future Generation Computer Systems*, vol. 95, pp. 140 – 148, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X18325299
20. P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.