# Cooperative Map-Building for Autonomous Mobile Robots

by

Rajish Jaywant Wagh

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
May 6, 2017

Keywords: robotics, SLAM, mobile, mapping,
localization, simulation, cooperative

## Abstract

A novel algorithm for multi-robot simultaneous localization and mapping is investigated through simulation. Although cooperative-SLAM has been studied by other researchers, the algorithm presented here offers a simpler, less computationally-intensive solution to map merging. The novel way to create local maps is identifying standard landmarks appearing naturally in the given area. The simulation results show the effects of three parameters: (1) number of robots, (2) map overlap matching threshold, and (3) map overlap window size. The success metric is the number of simulation steps required to fully explore a given map. This in turn validates the efficiency and accuracy of the map merging technique used.

The map overlap matching threshold establishes the degree to which compared parts of maps from two different robots must agree for the algorithm to join the two maps. The map overlap window size establishes the number of map cells that are considered when comparing maps from two different robots.

The results show that, within the range of parameters studied, increasing the number of robots significantly decreases the number of steps required to fully explore and map the given area. It is also shown that the number of steps required to fully explore the map increases when the map overlap matching threshold is increased. In the latter case, the simulation time also increases with increasing threshold.

## Acknowledgments

I would first like to thank my thesis advisor Dr. Thaddeus Roppel. The door to Dr. Roppel’s office was always open whenever I ran into a trouble spot or had a question about my research or writing. He constantly steered me in the right direction whenever he thought I needed it. I would also like to thank Dr. John Hung and Dr. Shiwen Mao for their valuable time and support as members of my thesis committee. I would also like to thank all the faculty and staff of the department of Electrical and Electronics Engineering, Samuel Ginn College of Engineering, Auburn University for their continuous support and well wishes. I would also like to thank Daniel Schreck and Felix Hummel for discussing this project with me during the inception of the ideas and preliminary coding experiments. I would like to thank Saleel Shetye for helping me with his python expertise several times whenever I got stuck with the code. Finally, I must express my very heartfelt gratitude to my parents, my family and my friends for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Table of Contents

## List of Tables

List of Figures

## List of Abbreviations

| | |
|---|---|
| SLAM | Simultaneous Localization and Mapping |
| C-SLAM | Co-operative Simultaneous Localization and Mapping |
| IDE | Integrated Development Environment |
| WS | Window Size |
| AP | Acceptance Percentage |
| YF | Yellow Field |
| NASA | National Aeronautics and Space Administration |
| AI | Artificial Intelligence |
| IEEE | Institute of Electrical and Electronics Engineers |
| EKF | Extended Kalman Filter |
| RBPF | Rao Blackwellized Particle Filter |
| MTM | Map Transformation Matrix |
| PSM | Polar Scan Matching |
| VSL | Virtual Supporting Lines |
| GIS | Geographic Information System |
| RRT | Rapidly Exploring Random Tree |
| LPA* | Lifelong Planning A* |
| PyPI | Python Package Index |
| SDL | Simple DirectMedia Layer |

| | |
|---|---|
| RAM | Random Access Memory |
| CPU | Central Processing Memory |
| GB | Giga byte |
| MB | Mega byte |
| HD | High Definition |
| AMD | Advanced Micro Devices |
| APU | Accelerated Processing Unit |

# Chapter 1

# INTRODUCTION

From the early explorers of the Phoenicians to the Mars rovers of NASA, looking for newer turfs and mapping them is something humans have consistently strived for. We have mapped and plotted almost all parts of the visible and traversable Earth. The ones that can be seen from up above have been mapped thanks to the satellites. But some areas like the underwater terrain, deep underground caves, etc. cannot be visited by humans. Exploring other planets and distant celestial bodies by humans also looks like a distant dream because of the safety issues involved, among other things.

Robots used for extra-terrestrial exploration have pre-programmed codes that enable them to make decisions on their own in an unknown, unseen and uncharted environment. One of their primary jobs is to map the surrounding areas to the best of their abilities. With increased number of robots in these operations, not only does the exploration take less time but the maps also become more reliable due to multiple robots' data. Such robots implement Simultaneous Localization and Mapping techniques (SLAM). Starting from an arbitrary initial point, a mobile robot should be able to autonomously explore an environment with its onboard sensors, gain knowledge about it, interpret the scene, build an appropriate map, and localize itself relative to this map [1].

SLAM is a major field of study in robotics. The need to automatically and accurately understand a previously unknown environment is finding more and more traction as man ventures more into space exploration. One type of consumer robotic vacuum cleaner is programmed to perform SLAM and create an image of the part of the house it is cleaning. It perceives a part of the house at any given time and plans a local path based on what it sees. Then it moves on to clean other parts of the house.

Autonomous mobile cooperative robotics is what the industry is looking forward to right now. The ability of a robot to move in a confined space, communicate and coordinate with other robots like it and at the same time perform its own duties is highly desirable. Some big industries have implemented such robots in their shop floors and they use it to move light and heavy equipment/materials automatically. In cooperative SLAM, each robot generates its own map of a part of the area which is then compared to the maps developed by other robots. Thus, an important problem in this is to decide when, where and how to merge the different maps of the individual robots. Cooperative SLAM comes from two old ideas: teamwork and curiosity. Entities, human and non-human, working together have been found to complete tasks faster than when they attempt to do them all alone.

One of the inherent challenges in cooperation is constructing a global map from the individual maps generated by each cooperating entity. This thesis presents a solution to the problem of map merging that is simpler to implement than traditional techniques.

In Chapter 2, an overview of the contemporary literature is provided. This is followed in Chapter 3 by the description of the software and hardware used for the simulation. In Chapter 4, the different algorithms used for the code are discussed in detail. Results obtained after the simulations are explained in Chapter 5, and conclusions and suggestions for future work are compiled in Chapter 6.

# Chapter 2

# LITERATURE SURVEY

This chapter introduces a few important technologies and concepts pertaining to co-operative robotics and other related fields. Also, past research related to the fields of multi-robot SLAM, path planning and navigation, map matching and merging techniques is presented.

## 2.1 Simultaneous Localization and Mapping (SLAM)

The concept of Simultaneous Localization and Mapping (SLAM) tries to enable a mobile robot to build a consistent map of its environment while simultaneously determining its own location on this map, while assuming that the robot does not know its own place at the beginning [2]. Several attempts to solve this problem have been made since the 1980s. In the 1986 IEEE Robotics and Automation Conference probabilistic methods were introduced into both robotics and artificial intelligence (AI). Over the next few years, various papers contributed to analyzing and determining the complete problem presented by SLAM. Amongst them, notable contribution came from Smith et.al. which proved to be a landmark in SLAM research [3]. It showed that, as the mobile robot moves through the unknown environment taking relative observations of landmarks, the estimates of these landmarks are all necessarily correlated with each other because of the common error in estimated vehicle pose [14].

The first effort that clearly defined the structure of the SLAM problem, the convergence result and the coining of the term “SLAM”, was a survey paper presented at the International Symposium on Robotics Research [4]. Thus, various estimation theories used to deal with the uncertainty in robot perception and motion have been introduced and worked upon in the recent past. Two major strategies that come forward are (extended) Kalman filter (EKF) and Rao-Blackwellized particle filter (RBPF). The extended Kalman filter successfully works on Gaussian mixture distributions. The RBPF, however, has been proven to work on any distribution non-parametrically [5]. This is commonly known as FastSLAM. The FastSLAM algorithm was first introduced by Montemerlo et al. [6].

### 2.1.1 C-SLAM

The SLAM problem has been seen to be better tackled using co-operation among multiple robots. This has been termed Co-operative SLAM or C-SLAM [14]. C-SLAM (Cooperative-SLAM) is an efficient framework to solve the problem of SLAM, which is based on the cooperation of multiple robots to estimate the robot poses and build a map of the environments [7]. It entails performing individual mapping by all the robots, matching and merging these maps at the right places to create an accurate representation of the environment in 2D or 3D maps. Prompt communication of information, map comparison techniques, map merging techniques and uniform decision making are a few additional challenges that are introduced with the introduction of C-SLAM.

#### 2.1.1.1 Communication Architectures:

There have been two obvious approaches in C-SLAM techniques used for data transmission: Centralized and Decentralized communication architectures. Notable papers like [8], [9], [10], [11] all provide examples of multi-vehicle SLAM where all vehicles send their sensor data to a central Kalman filter. All of these involve the communication of raw sensor data (distance, vision, etc.) from each vehicle to a central source and thus requiring a large amount of communications bandwidth. In [12] and [13] the algorithm decentralizes the communication architecture by equipping all the units with decision making abilities. The test of these techniques is that the decisions taken by these individual robots have to be uniform.

**2.1.1.2 Map Matching and Merging**:

Map matching and merging is employed to obtain a global map by merging the individual maps of the different robots using the Map Transformation Matrix (MTM) [4]. The general formation of the MTM is as follows:

$$MTM(\Delta_x, \Delta_y, \Delta_\Theta) = \begin{bmatrix} cos\Delta_\theta & -sin\Delta_\theta & \Delta_x \\ sin\Delta_\theta & cos\Delta_\theta & \Delta_y \\ 0 & 0 & 1 \end{bmatrix}$$

where $\Delta_\Theta$, $\Delta_x$ *and* $\Delta_y$ are relative angle and x-y translation amounts, respectively. In [7], Lee et al. have described two basic categories of map merging techniques: direct map merging techniques and indirect map merging techniques. Konolige et al. introduces a two-step map framework where the MTM is obtained using robot to robot measurements and then this hypothesis is confirmed by moving the robots to so they meet at a particular position on the hypothesized map [15]. If they fail to meet each other at the estimated location, the hypothesis is rejected. If they succeed their maps are merged. Another example of direct map merging is found in [16], where the MTM was computed by overlapped regions detected by ceiling-vision sensors. Image patches are collected around observed landmarks and analyzed to find common regions overlapped by the robots. Both the above techniques compute the MTM using direct data from visual and range sensors. A few other techniques involve finding and matching the common part of the discovered maps and merging them if they look similar. As in the case of [17] where they have used the nearest neighbor test as a point feature matching algorithm. Tungadi et al. have used the polar scan matching (PSM) algorithm to compute the MTM [18]. Spectral information on maps can also be utilized to develop a map merging algorithm as proposed by Carpin [19]. Lee et al. have proposed a technique where spectral information is extracted by virtual supporting lines (VSLs). VSLs are obtained by joining the features using lines around a selected central feature [20].

In this project, the Indirect Map Merging has been deployed where the maps are merged by comparing the field values of the already existing maps. Field values are decided according to the number and position of the neighbors, which act as intrinsic landmarks.

## 2.2 **Path Planning**:

Path planning is one of the essential tasks in the automation process of a system that moves in the environment while avoiding obstacles and respecting various constraints [21]. Unlike SLAM this is a much older field and has been much more successfully solved. Apart from robotics, path planning finds importance in other related fields like video games and automation. Although the various algorithms and hypothesis have been very effective in controlled environments, the ability to generate an efficient path from a given initial point to a final destination in real-time conditions is still one of the biggest challenges.

Environment modeling is a fundamental step that determines the selection of the path planning algorithm. Regular grids, Irregular grids, Navigation mesh, Visibility graph, and Voronoi diagram are some examples. Hart and Nilson in 1969 first introduced the Visibility graph method [22]. All the possible obstacle vertices are linked and this data can be used to find the shortest possible path from point A to point B in the environment. It has been proved, however, that this it remains a realizable solution only in 2D space modeling environment and with an off-line resolution. Irregular grids were introduced by Finkel and Bentley in which they proposed the use of “Quadtree” [23]. This is used in a lot of areas like GIS, image processing or 2D collision detection. In its 3D form it is called “Octree” [24], [25].

E. W. Dijkstra [26] in 1959 proposed the way to find the path of minimum length between two nodes on a graph. Even now, “Dijkstra’s algorithm” is one of the most popular graph traversal algorithms. Path traversal algorithms have been classified into three types: The A-star (A*), the Rapidly Exploring Random Trees (RRTs) and the Potential Fields algorithm [27]. Hart and Nilson in 1969 introduced the well-known A* algorithm [22]. This algorithm is used in many applications even today. The A* has received many modifications over the years and these are named differently according to their characteristics. The D* has the same basic construction of the A* and has an additional property of being dynamic. The idea was proposed by A. Stentz in 1994 and meant reacting quickly to any change on the studied map [28]. Lifelong Planning A* (LPA*) reduces the number of nodes needed to be examined in future runs by using previous data [29]. Notable among these is Field D* which was used by NASA for the Mars rovers. One of the major drawbacks of the A* algorithms is their execution time. The Potential Fields algorithm, introduced in 1986, is considered one of the fastest. It was first introduced by O. Khatib in [30]. In a given space, the RRTs strategy is to expand a tree by randomly sampling the space and growing from a starting point until the tree is sufficiently close to a known target. In bi-directional RRT, two trees are launched, one from the start node and the other from the target [31]. RRT* is an incremental sampling based algorithm which finds an initial path as the basic RRT and later enhances the performance using the triangle inequality [32].

The algorithm used in this project is a type of RRT that initiates a tree for every direction it can go, and expands all over the space until it finds a target. Then it finalizes the optimum path towards the target and takes the first step on this new path and re calculates the new path to its target from the new position. There are two reasons to do this: firstly, due to the presence of other robots, the target node can change at any iteration and secondly, even the known map keeps changing at every step of every robot, so there could be a new, better path available to reach the same node that the algorithm previously didn’t know about.

# Chapter 3

# METHODS AND EQUIPMENT USED

## 3.1 Software

Python is one of the most widely used general purpose programming languages. This chapter introduces Python 2.7 that was used for the simulation. It also describes the important properties of Python that are pertinent to the code.

### 3.1.1 Python:

Python is a very popular high-level programming language created by Guido van Rossum in 1991. The philosophy behind creation of such a language was that the code must be easily readable to most programmers and more compact than other contemporary popular languages like C++ or Java. All its features from automatic memory management to object oriented approach and the fact that the code is cleaner (owing to the use of indentation) serve to justify this philosophy.

The Python development process is very transparent and has a strong community backing. Python’s versions, 2.6x and 2.7x series and 3.0 are the most popular. Its high-level built in data structures, combined with dynamic typing and dynamic binding are important from the point of view of an Application Developer. It supports modules and packages and the code can be reused. This is a great feature for use in research simulations. It provides an increased productivity since the edit-test-debug cycle is very fast due to the absence of the compilations step. To handle compiler errors, Python uses exceptions. In turn, these exceptions can be made use of by the programmers themselves to avoid several errors. Due to all of this, often the quickest way to debug a program is to add a few print statements to the code. The core philosophy of the language is summarized by the document The Zen of Python (PEP 20), which includes aphorisms such as:

i. Beautiful is better than ugly
ii. Explicit is better than implicit
iii. Simple is better than complex
iv. Complex is better than complicated
v. Readability counts [33]

As of 20 August 2016, the community of Python developers has also contributed over 86,000 software modules to the Python Package Index (PyPI), the official repository of third-party libraries for Python. [34]

One such library, *pygame*, is a free and open source Python programming language library for making multimedia applications like games built on top of the excellent SDL library. It is highly portable and runs on nearly every platform and operating system [35]. It was used in this thesis work for producing a graphical game-like representation of the code in real-time.

The elegant syntax, ease of use and understanding, large standard library, versatility, vast community support and other such factors make it one of the most powerful languages in robotics programming. It enables even an intermediate programmer to produce the desired code in a short period of time.

## 3.2 Hardware

For completeness, the computers that were used for the simulation are detailed here. One of the computers (desktop), used an Intel Xeon CPU E3-1225 v3 running at 3.20 GHz. It had an installed memory (RAM) approximately 16.0 GB and used a 64-bit

Windows 10 Enterprise Operating System. The video graphics card in that machine was an Intel HD Graphics P4600/P4700 that had an approximate memory of 2160 MB. Another computer used was a laptop. It utilized an AMD A10-5750M APU with an installed memory of around 8.00 GB. The video graphics card it had installed was an AMD Radeon HD 8650G + HD 8750M Dual. The approximate total memory was 6219 MB. This was also a Windows 10 machine that used the 64-bit version of the operating software.

Most Python external libraries are very flexible with respect to the hardware on which Python runs. ‘Pygame’, a library used in this project to display the progress of the simulation in real-time, does the same for this project. It runs equally well for both the different machines without requiring any change in the code specific to the device it is used on.

# Chapter 4

# SIMULATION

The aim of the work described in this thesis was to simulate multiple robots collaboratively exploring and mapping an unknown area. Towards this end, the following algorithm was designed.

## 4.1 Assumptions

The simulation utilizes very minimal assumptions and depends on the strength of the algorithm to completely explore and accurately create the map of an unknown area. The list of these assumptions is as follows:

1. The robots traverse in a 4-neighbor world. The map must be a square.
2. In case of multiple robots, all of them start with different poses (orientations).
3. At any stage of the program, two or more robots can be present in the same field position.
4. Each robot knows its orientation at the start. They always face north in the beginning.
5. Communication between any two robots is guaranteed. So, there is no provision for communication failure.
6. The map consists of alleys only. This means, there can’t be a place where all four fields of a 2x2 space are traversable.
7. The simulation is for a 2-dimensional map. For simulation purposes only the floor plan of the map is considered.

Apart from these, a few things that demonstrate the strength of the algorithm are as follows:

1. All robots are independent. There is no central robot or a leading robot.
2. None of the robots know the starting floorplan.
3. The map size need not be pre-determined. The program will work with any map that is a square.
4. None of the robots know their starting positions with respect to any point on the real map. All the localization that takes place is relative to the local map at the robot's memory.

## 4.2 **Behaviors**:

Each robot action can be called a behavior. These behaviors are mostly independent of each other and represented in code by functions or classes. As stated earlier, they can be considered as the states in which the robots are at any given time. There are seven different major behaviors for each robot. Most of these behaviors logically follow each other while some are activated intermittently depending on the stage at which the whole system is currently. The behaviors are: Robot initialization, Path planning, Path following, Map building and Localization, Map matching, Map merging, Communication. As soon as the map is built for the initial position, all the other states are activated sequentially and continue to do so till the map is thoroughly explored. Each of these series of behaviors can be considered as a *step*. Each step consists of path planning, path following, map building and localization, map matching, map merging, communication. At the end of each step, a new local map is ready with new fields to traverse to. Thus, the next step is taken considering this new local map.

### 4.2.1 Robot Initialization

Robot initialization is an important step in the program since it is the first time a local map is made. Even though the map is a 4-neighbor environment, the first map is a 3x3 matrix of all the eight neighbors of the robot with the robot being at the center of this matrix. Since all maps are considered to be matrices, it becomes easy to initialize such a map. As stated in the assumptions, the initial orientation of the robot is recorded to be facing north.

### 4.2.2 Path Planning

The Path planning algorithm works in a similar fashion to the Rapidly Exploring Random Trees (RRT) algorithm [27]. Since the robot only knows the map it has already covered, it tries to look at places it has marked as traversable but hasn't yet been visited (known as the Yellow Field or YF). This is very useful to determine if the map is completely explored. At the end of each step, the code makes a check for any remaining unexplored fields (YF), If any of such YFs exist in the local map, the robot decides that the map hasn't yet been completely explored and decides to move towards the nearest YF. At this point, there can be two scenarios: a YF can exist as one of its current neighbors or none of its neighbors is a YF. Both these cases are explained in the following sections in detail.

These two conditions can be described as follows:

**a.** Target is a Neighbor: This is the easier one of the two. In this case, one or more of the four neighbors is a YF. The robot only needs to determine which one is the target (according to condition pre-set by the programmer like 'always prefer left').

**b.** Dead-end situation: A situation where the none of the neighbors is a YF. Path planning in this case is a computationally heavy process that requires the calculation of all the possible paths from the given point towards the nearest unexplored area on its local map. In this process, all the different paths leading from the current position of the robot are traced and registered in a matrix until a field that hasn't been traversed yet (YF) is reached. Once such a YF is found, the path that leads to it is extracted from the matrix. The first element in this path is the next best position for the robot to move towards the nearest new location. This algorithm finds the target location based on the distance from current position. Thus, it accomplishes two different tasks at once. This is one of the biggest advantages of this algorithm. The newly determined next position is relayed back to the main body of the code in terms of direction to move.

Another important feature of the code is that even though the robot calculates the entire path to its target, it is recalculated at every step. This is because the map is designed to be updating at every step. At every step, each robot has certain new information to share with other robots. This information is updated at each step so that the path planning becomes more efficient.

### 4.2.3 Path Following:

Path following in a simulation environment means to update the values of all the variables responsible for maintaining the current position, predicted next position and orientation of the robot. In real life scenario, this corresponds to movement of the robot immediately after the determination of the path. Once the variables for next position are updated, technically, the robot has followed the set path one step at a time. The robot performs all other behaviors before it takes the next step. This next step will again be determined by the previous function called the Path Planning Function.

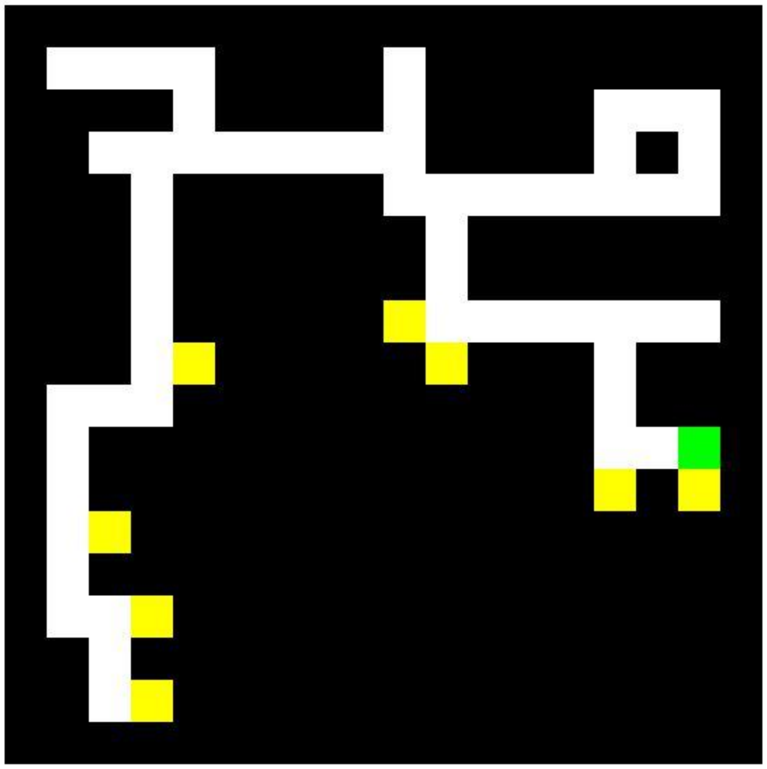


Figure 4.1: An Incomplete map with confirmed but unexplored fields marked with yellow color that give it that name: ‘Yellow Fields (YF)’

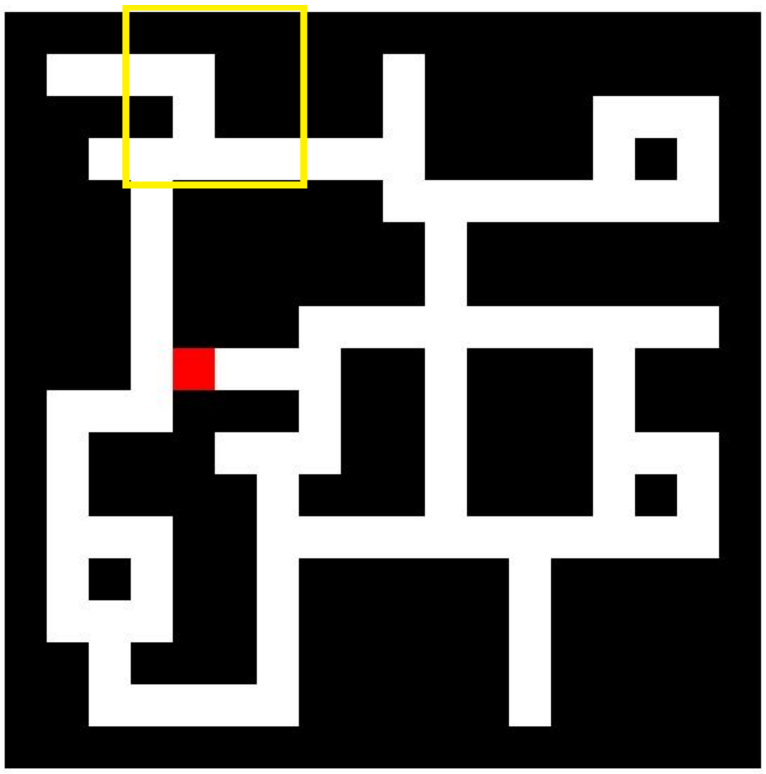
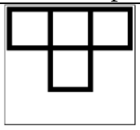
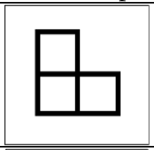
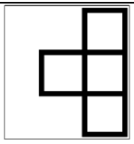
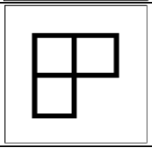
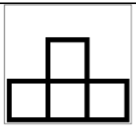
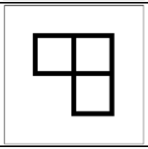
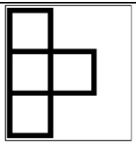
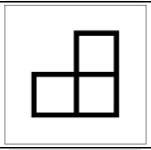
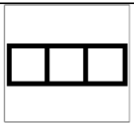
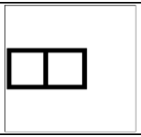
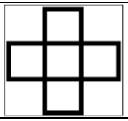


Figure 4.2: Final map with all fields explored. Also shows the common area pertinent to section 4.2.7

### 4.2.4 Map-building and Localization:

The map that each robot builds according to the surrounding it encounters is called its ‘local map’. This algorithm uses a very novel idea to build the local map around the robot. It starts at the initialization phase and is expanded in this function. For easy traversing and computational purposes, the local map is designed to be a 2-dimensional matrix that has different field values for different field types (landmarks). This helps in the process of map matching in which comparison between the field values is made to identify similar looking regions. The field types are defined according to how they appear from the top of the map. The field value depends on its traversable neighbors and their positions. For example, as in Figure 4.3(a), if the field has traversable neighbors to its east, west and south, it gets the value: ‘1’. At the same time of a particular field has traversable neighbors towards north, west and south, the field is given a value: ‘2’ (Figure 4.3(b)). There 10 such identified landmarks that occur everywhere in the real world. The table 4.1 gives a complete list of such landmarks.

Table 4.1: Field Values attributed to various landmarks

| Landmark Appearance from Top | Field Value | Landmark Appearance from Top | Field Value |
|---|---|---|---|
|  | 1 |  | 5 |
|  | 2 |  | 6 |
|  | 3 |  | 7 |
|  | 4 |  | 8 |
|  | 0 |  | 0 |
|  | 9 | | |

Since the map is a 4-neighbor world, the robot can have information only about its four current neighbors at any point in time. Therefore, the robot can calculate the field value of only its current position. Also, at any given point it can determine which of its four neighbors is/are traversable. If it finds a traversable neighbor, the robot updates its local map with a temporary field value of ‘B’ assigned to that field. Instead if the robot finds that the neighbor is a wall/obstruction, the field value attributed is ‘A’. The value ‘B’, corresponds to the Yellow Fields (YF), discussed in the last topic. A map that is not yet completely explored looks like the one in Figure 4.1. If there is an unknown field in the local map, the field value assigned is ‘C’. After the robot moves to the nearest ‘B’ or YF, it now has enough information to identify the permanent field value of this field. So, it looks up into the table and renames this field from ‘B’ to the appropriate field value. In this way, the local map is continuously updated until all the ‘B’s are converted into appropriate permanent field values. The robots work together to obtain a map that looks like in Figure 4.2.

As mentioned in chapter 2, two methods for map merging have been traditionally used to build the global map. Firstly, when the robots meet face to face, a common reference point is achieved. In the second method, an overlapping region in found in the two maps. This new map is confirmed by making the robots physically meet in some place [37]. However, due to the new way of making the local map and identifying the landmarks, the two maps can be merged without the need of such a confirmation.

As the robot moves around, the local map keeps increasing. Extra rows and columns are added as per required at every step. Along with the building/updating of the local map, localization is achieved by updating the variable that maintains the current position of the robot in that map. Each robot is responsible for updating its own map and localizing itself in that map. By making sure all the robots do this consistently, overall localization is achieved.

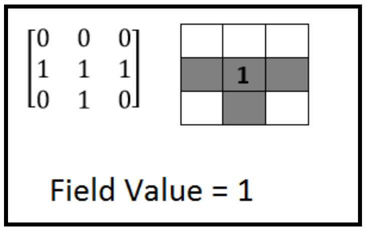


Figure 4.3(a): Landmark T1 with field value

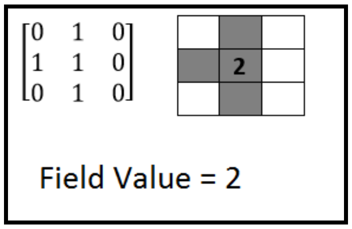


Figure 4.3(b): Landmark T2 with field value

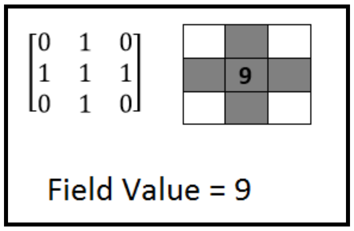


Figure 4.3(c): Landmark + with field value

### 4.2.5 Communication:

Uninterrupted communication is one of the assumptions made at the start. During simulation, it is acceptable to assume that there will not be any communication related errors. In real life, such errors can easily happen and error correction methodologies can be implemented depending on specific cases. In the algorithm, communication takes place at this position of each step. All the robots exchange their local maps and other relevant data. The next few steps are taken according to these updated values.

### 4.2.6 Map Matching:

The process of map building continues on every step from the beginning. Certain parameters are defined in order to achieve some amount of flexibility in the map matching process. These parameters are defined by the user at the start of beginning. Since the maps are defined as 2-dimensional matrices, parts of these maps (or matrices) can be compared to each other to find common appearing areas or subsets. The size of the windows/subsets (NxN) to be compared can be defined by the user and generally a value between 3 and 6 is considered good. This is called the ‘Window Size’ (WS) and the results depend heavily on this parameter. Once the WS is fixed the map matching algorithm waits till the sizes of local maps are larger than the defined window size. Then it starts comparing and trying to match every subset of the local map of every robot to that of every other robot. It calculates the match percentage of every subset pair in a 4-dimensional matrix per their positions in the respective local maps. Eventually the highest of the lot is obtained. This is the second parameter that the user can define: The Acceptance Percentage (AP). This is usually defined between 40 to 90 percent. If the highest match percentage is greater than or equal to the user defined AP, the match is said to be successful and the coefficients corresponding to this match are forwarded to the next stage: Map Merging.

Map matching is completely dependent on the user inputs of WS and AP. Therefore, the outcome of the runs is also completely dependent on their values. This is also one of the major causes of failure for this algorithm. If the WS is too small, or the AP is too small or large, the success is less likely. The code is designed such that it finishes the task even if there is no positive match between any or some of the robots’ local maps. Therefore, success of the algorithm lies in how early the robots finish the job (the number of steps required). This has been better illustrated in the results chapter. In situations where the WS or AP are too small, the map matching algorithm is fooled into matching two seemingly similar subsets which do not really belong to the same part of the real map. This causes wrong merging of the maps in the next stage. Some error correction strategies have been implemented in the algorithm that prevent the wrong merging of these maps to a large extent. But as happens with most algorithms, a few cases still fail.

### 4.2.7 Map Merging:

As soon as the right match is obtained, the control goes over to this stage, called the map merging stage, which merges the two local maps at coefficients determined at the map matching stage. This is the most important stage in cooperative robotics and depends entirely on the data provided by the map matching stage. The map merging algorithm merges the two local maps and creates identical maps in place of the two original local maps. These new maps are, in most cases, much larger than the original local maps. Thus, both the maps need to be expanded in the right directions and in the right amounts. The data in each map must be inserted at the correct new positions. During this insertion, it is important to see that there is no conflicting situation. Such a conflicting situation arises when two different field values appear on the same position in the two original maps. This clearly means that even though the parts inside the WS match, the rest of the maps do not. In such cases, the maps are not merged and next steps are taken.

At the same time, localization must be achieved because of the sudden change in the local maps. The variables responsible for maintaining the current position and orientation are updated according to the merger. After a successful merger, certain flags are updated. For example, if robot 1 and robot 4 have matched and merged, the flag called, 'one_four_done' will be set to True. This indicates that these maps have been merged and no more matching is required henceforth. In such a case only those two maps are merged at the right positions.

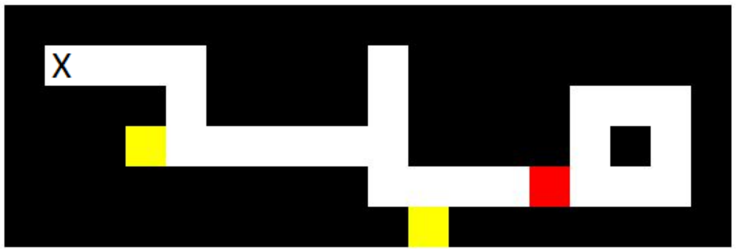


Figure 4.4: Local map of Robot 1 just before it merges with Robot 2

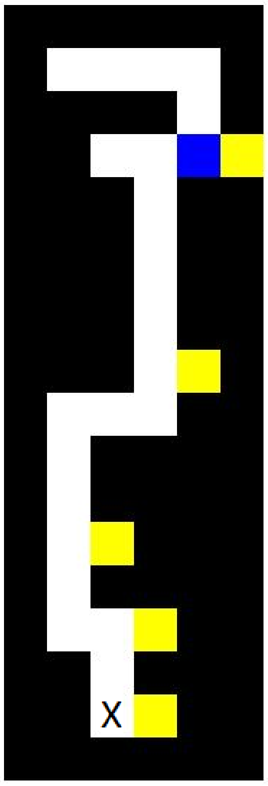


Figure 4.5: Local map of Robot 2 just before it merges with Robot 1

For example, in Figure 4.4, Robot 1 has explored some part of its map starting at the position marked with X. Similarly, Robot 2 in Figure 4.5 after having started at the position marked with X, has covered a part of the map. This case is for WS = 4 and AP = 80%. Since the algorithm has been continuously trying to match all the 4x4 parts of one map with that of the other, it will soon realize that one such window does match at this step. The marked window in Figure 4.2 is the common window that satisfies the acceptance percentage. Figure 4.6 shows the values of the common fields in the local maps. As soon as the maps are matched, they are merged in this step and the new integrated map looks like the one in Figure 4.7. This map is copied into the local memories of both the robots and their maps will be continuously and simultaneously updated from here on.

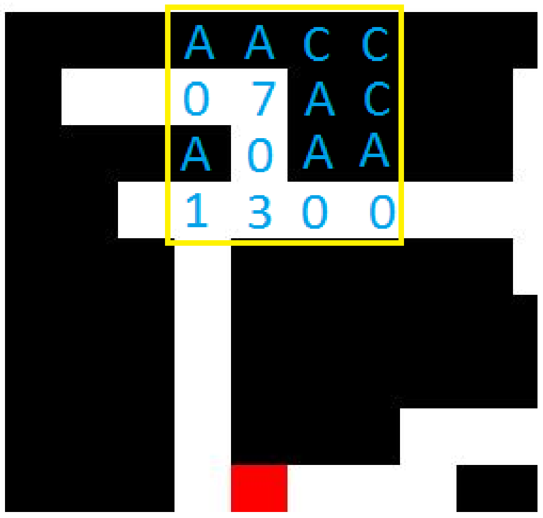


Figure 4.6: Field values of the common area of the two maps

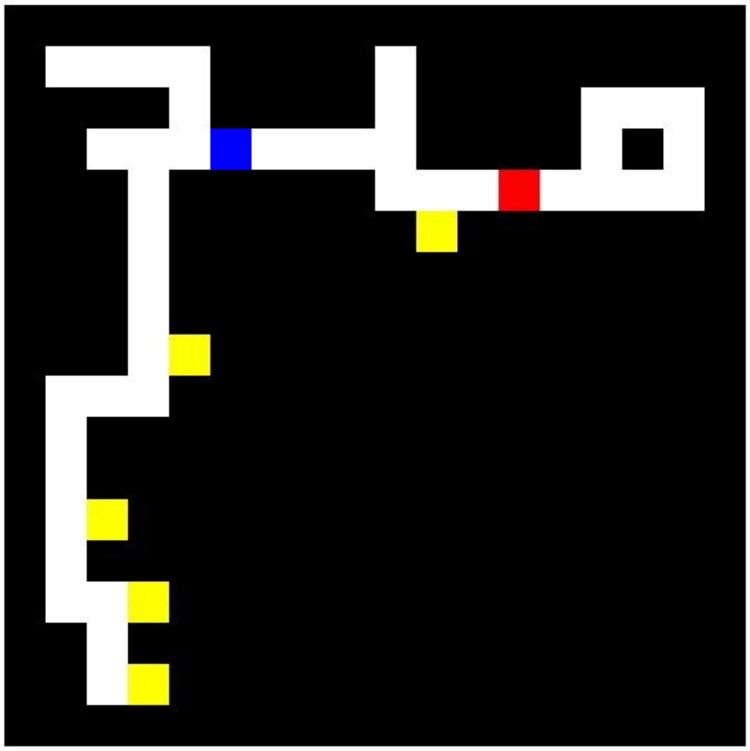


Figure 4.7: Local map of both robots just after the merge

A few strategies are employed to prevent errors in merging. Some of these even deal with erroneous data provided by the map matching stage and prevent bad mergers. Others deal with wrong mergers that may happen in the map merging stage itself. Eventually, when at least 1 robot completes the exploration, the experiment is said to be over. If all the robots have merged their maps before any one robot finishes its job, it is said to be a success. This is the most probable case and always finishes the job in much fewer steps than otherwise.

# Chapter 5

# RESULTS

A variety of simulations were performed using the algorithm described in Chapter 4. A variety of maps were used to perform different tests to demonstrate flexibility of the algorithm. These tests were performed according to the Design of Experiments methods used to determine the cause-and-effect relationship between all our input and output variables. The maps are shown in Figures 5.1(a), (b), (c), (d) and (e). Each of these maps have about 90 to 120 fields where the robots can travel to. These maps were created by the simulation with the help of the Python library pygame. The red squares in each of them are just the last position of one of the Robots. As stated in previous chapters, the code was fairly flexible. The inputs like number of robots, their starting positions, comparison window size (WS), acceptance percentage (AP), type of map etc. determined outputs like number of steps, time to compute etc. Thus the 'experimental unit' can be said to be one complete exploration of any one map, with n number of robots, 'mxm' size of WS, p percent of (AP). All the experiments try to find the case with least number of steps required to cover the entire area. Therefore, various experiments were done on a random selection of starting positions for the robots and one or two factors were changed with every type of experiment.

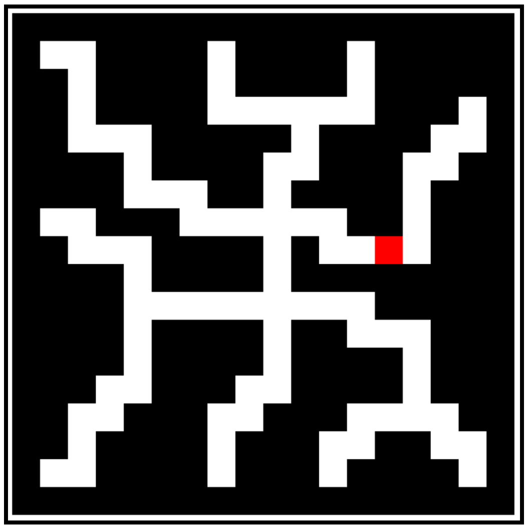


Figure 5.1(a): Map 1

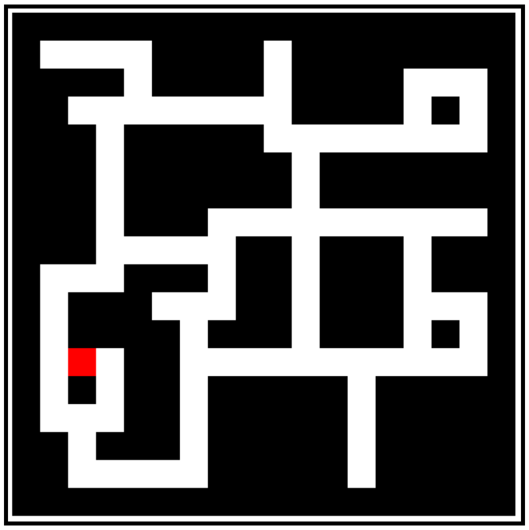


Figure 5.1(b): Map 2

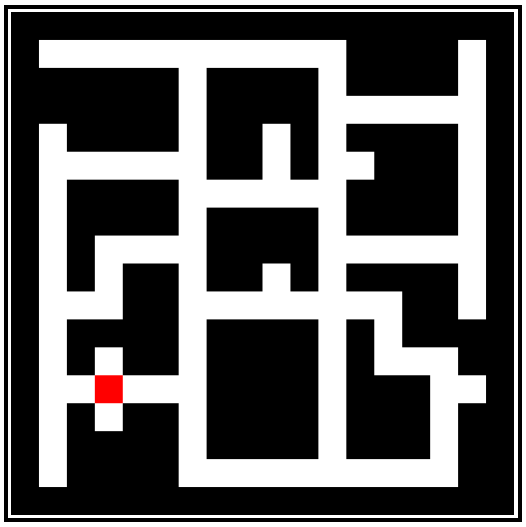


Figure 5.1(c): Map 3

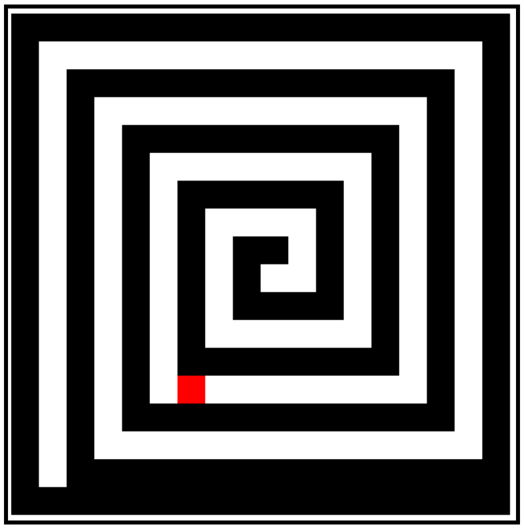


Figure 5.1(d): Map 4

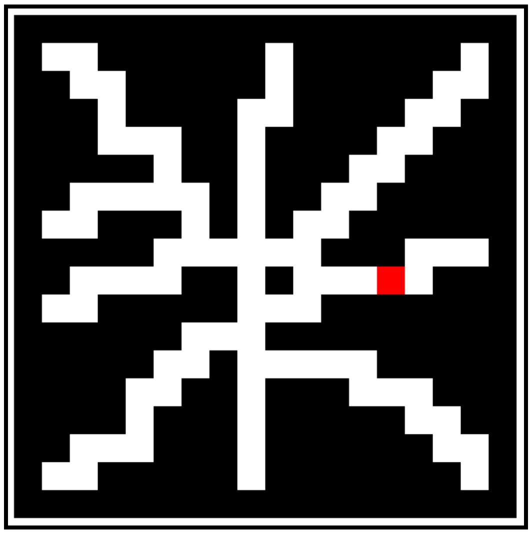


Figure 5.1(e): Map 5

## 5.1 Number of Robots

The first test was done using one robot to explore the whole map on its own. The results for this were used as reference for the cases with multiple robots. These experiments with a single robot showed that the starting position of the robot determined the number of steps the robot will take on any single map. Therefore, it was important to replicate the behavior and randomize the sample space. The sample size of 30 has been considered large enough from the times of William Gosset, a statistician and Head Brewer for Guinness. He (under his pseudonym "Student") concludes his article 'Probable error of a correlation coefficient. Biometrika' by stating that with samples of 30, the mean value approaches the real value comparatively rapidly [36]. Effectively, it was decided that each experiment will be run exactly 30 times with random starting positions in order to get the optimal data. Accordingly, the code was consistently changed and updated with every new discovery that was made during different runs of the program. From these trials, it was determined that a WS of 5 and an AP of 80% was a reasonable estimate to hold constant while we change the number of robots working in the same map.

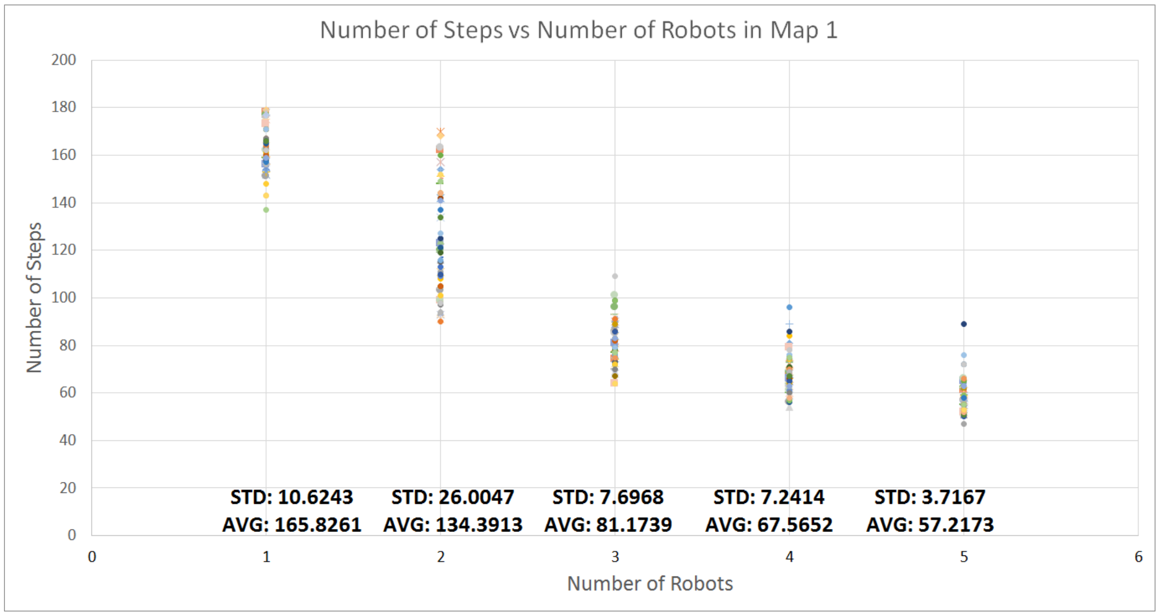


Figure 5.2: Number of steps versus Number of robots in Map 1.

A total of five cases were used: With 1, 2, 3, 4, and 5 robots. Each of the cases was run 30 times with different robot starting positions. Five different maps were used to see if the hypothesis holds true for all of them. These maps are shown in Figure 5.1. It was seen that the number of steps required by the robots to cover an area monotonically decreases as the number of robots is increased. Figures 5.2, 5.3, 5.4 and 5.5 show graphs comparing the number of steps required by each case over a randomized sample of 30 different starting positions for all 5 cases as mentioned above. On each of these graphs, the y-axis represents the number of steps required by each of the cases to complete exploring and mapping the map. The x-axis represents the 5 cases: from the use of 1 robot to the use of 5 robots at a time.

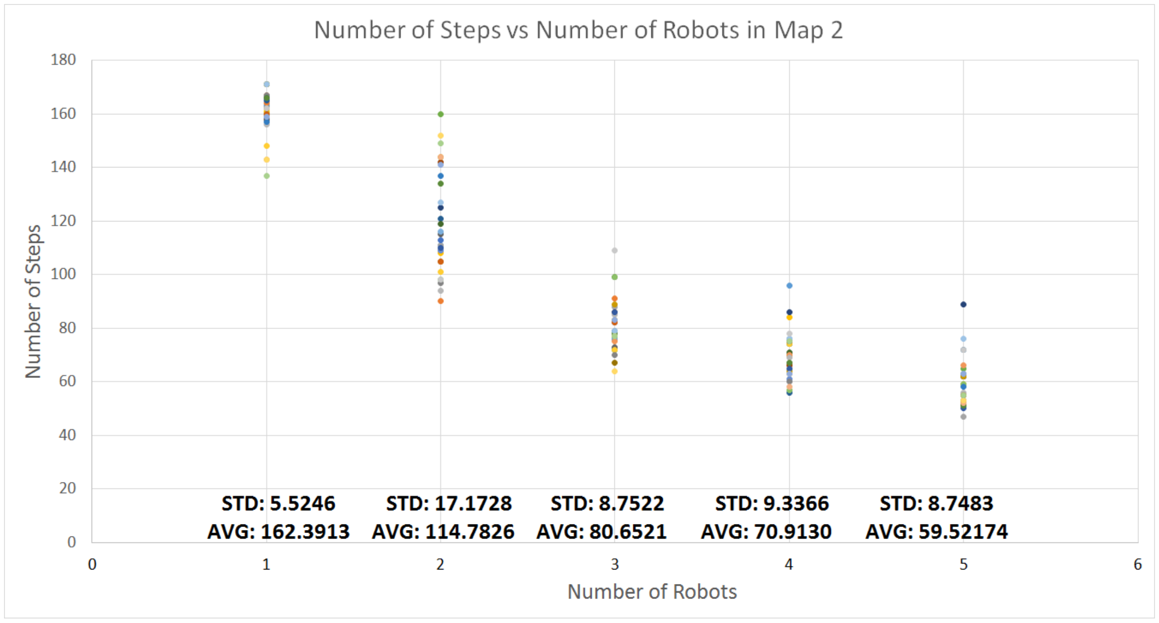


Figure 5.3: Number of steps versus Number of robots in Map 2.

Table 5.1: Average number of steps required to fully explore a map, with Acceptance Percentage of 80% and a Window Size of 5. Each cell is the average of 30 runs.

| | 1 Robot | 2 Robots | 3 Robots | 4 Robots | 5 Robots |
|---|---|---|---|---|---|
| Map1 | 165.82 | 134.39 | 81.17 | 67.56 | 57.21 |
| Map2 | 162.39 | 114.78 | 80.65 | 70.91 | 59.52 |
| Map3 | 129.26 | 111.21 | 80.39 | 70.56 | 62.26 |
| Map5 | 167.04 | 126.52 | 80.56 | 68.91 | 57.69 |

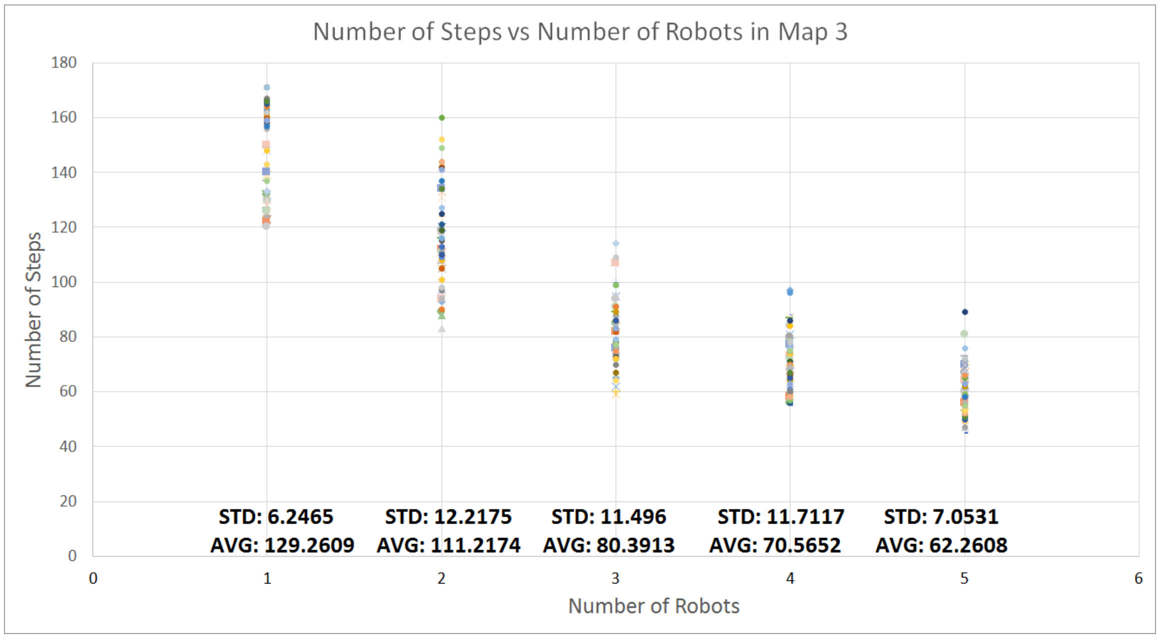


Figure 5.4: Number of steps versus Number of robots in Map 3.

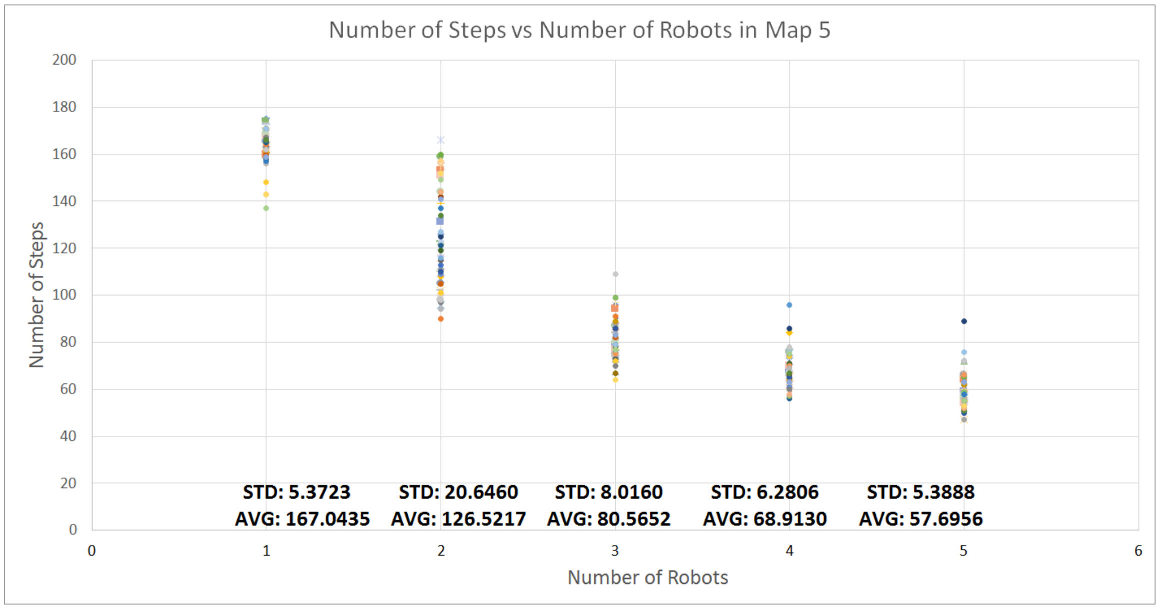


Figure 5.5: Number of steps versus Number of robots in Map 5.

Table 5.1 also shows the same trend as we see in the graph. The average number of steps required for the different cases decrease with the increase in the number of robots. It is conclusively evident from these graphs and the table that in the case of this algorithm, cooperation works in the favor of our target. The more robots used, the earlier is the whole area covered and mapped.

However, when the number of robots is increased, a few cases were observed where the map gets explored before all of the robot maps are merged together. In such cases, the robots that have completed can retire and the others continue to explore. In this case, these robots can be called redundant. Such cases become more frequent when we increase the number of robots for the same map. Therefore, depending on the size and type of the map used, an optimal number of robots can be obtained that will effectively explore the entire area in the shortest time possible.

## 5.2 Acceptance Percentage (AP)

The acceptance percentage is a value that is defined by the user. When two windows are selected from the two local maps being compared, a match percentage is calculated for the pair. Likewise, a 4-dimensional matrix is obtained that stores the values of all the match percentages obtained. When all the windows from one map are compared to all from the other map, the pair with the highest percentage is chosen. Here, if its match percentage is higher than the AP defined for that case, the match is considered successful. This has a significant effect on the number of steps as well as the computation time.

For experimental purposes, Maps 1, 2 and 5 were selected for this part of the study. The number of robots was fixed at 5. WS was fixed at a value equal to 5 and AP was varied from 40 to 90 in steps of 10. With each of these combinations, 30 runs were performed with randomly selected starting positions for all the 6 cases. At the end of each case, the total steps and time required to compute all the results was recorded. Table

5.2(a) displays the total number of steps taken by the three cases. From this table, it can be seen that from the level 50, the average number of steps required for exploring the whole map increase monotonically with the value of AP. The AP level of 40% is not significantly different from the case when AP = 50%.

Table 5.2 (a): Average number of steps to explore the area at different levels of Acceptance Percentage, with 5 robots and a Window Size of 5. Each cell is the average of 30 runs. Standard deviations range from 5.4 to 10.3 with no evident trend or correlation to the other parameters.

| **AP(in%)** | **40** | **50** | **60** | **70** | **80** | **90** |
|---|---|---|---|---|---|---|
| **Map 1** | 51.73 | 51.13 | 53.4 | 52.73 | 59.66 | 68.86 |
| **Map 2** | 52.53 | 49.1 | 51 | 52.53 | 61.1 | 68.3 |
| **Map 5** | 50.5 | 50.93 | 54.13 | 56.7 | 58.46 | 66.6 |

Table 5.2(b) displays the average time (in seconds) taken by the computer to compute the results for different cases. The time taken by the computer to compute these values varies monotonically with the value of the Acceptance Percentage. It can be concluded that the computer takes lesser time to compute the exploration of the whole map as the AP is reduced.

Table 5.2(b): Average time (seconds) required to complete the maps when the AP level is varied from 40 through 90 in steps of 10, with 5 robots and a Window Size of 5. Each cell is the average of 30 runs.

| **AP(in%)** | **40** | **50** | **60** | **70** | **80** | **90** |
|---|---|---|---|---|---|---|
| **Map 1** | 3.30 | 3.74 | 3.87 | 4.34 | 7.02 | 15.13 |
| **Map 2** | 2.59 | 2.95 | 4.36 | 4.56 | 22.73 | 39.13 |
| **Map 5** | 3.59 | 4.37 | 4.28 | 5.50 | 8.69 | 12.76 |

These tables clearly show the effect of AP on the overall outcome of the simulation. Through more simulations using other factors as explanatory variables it was established that factors like the window size (WS) and starting position also affect the outcome of the simulation. However, the results were scattered and no conclusive trend could be formulated despite analyzing a large data sample.

# Chapter 6

# CONCLUSIONS

This chapter consists of two sections. The results are summarized into the first section and conclusions are stated. In the second section, the possible future modifications are listed.

## 6.1 Summary

The simulation was done with the new algorithm for map building and merging stated in the previous chapters. All results show that the new algorithm fairs favorably with the traditional algorithms. Following are some of the conclusions that can be made from the various simulations performed.

It can be solidly concluded that increasing the number of robots used in the simulation decreases the number of steps for the robots to completely explore and accurately map the unknown area. With 5 robots working at the same time, for a map with around a 100 field positions available, the exploration and mapping took around 40-50 steps. This was significantly lower than the case with 1 robot: about 5 times less. In the case with 2 robots, the situation was very unpredictable. The average was still around 120 steps.

Another important parameter in the simulation was the Acceptance Percentage (AP). From Table 5.2, when the AP is increased, the average number of steps the robots take also increases. Also, the time taken by the simulation to cover the complete area and map it increases as the AP is increased, as shown in Table 5.3. The AP is therefore an important factor in deciding the outcome of the simulation.

Other factors like the Window Size (WS) and starting positions did not yield conclusive results and can be declared as random factors.

As mentioned in Chapter 4, one of the important assumptions was that the map consists of only alleys and no rooms. This is still a practical constraint since there a lot of situations where vehicles travel on roads/paths outdoors that can be included under this constraint. Indoor gangways/corridors can also be included under this constraint. But it cannot be denied that this algorithm definitely fails when we include rooms in the maps. However, this algorithm  can be modified so that the robots can travel to all parts of the map even though there is a room or an alley.

## 6.2 Future Scope

In the beginning of the project, there were several assumptions made. A few of them can be modified and tested in the future. The map consists of alleys only and this can be changed to make the map more realizable like any other room. The environment can be changed to have more than just 4 neighbors. More flexibility can be added by making both, the map size and the number of robots, user-dependent. This can then be used for large-scale swarm robotics applications. More number of landmarks can be identified and this means more accuracy while comparing and matching.

As a later development, the model can be adapted to include a 3-dimensional map instead of the 2-dimensional map that is being used currently. The communication can be made more secure and code can be implemented on real robots in a controlled environment where the communication is not always ideal.

# REFERENCES

[1] Siegwart, Roland, and Illah Nourbakhsh, "Introduction," in *Introduction to Autonomous Mobile Robots,* Cambridge, MIT Press, 2011

[2] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I," in IEEE Robotics & Automation Magazine, vol. 13, no. 2, pp. 99-110, June 2006.

[3] R. Smith, M. Self, and P. Cheeseman, “Estimating uncertain spatial relationships in robotics,” in Autonomous Robot Vehicles, I.J. Cox and G.T. Wilfon, Eds. New York: Springer-Verlag, pp. 167–193, 1990.

[4] H. Durrant-Whyte, D. Rye, and E. Nebot, “Localisation of automatic guided vehicles,” in Robotics Research: The 7th International Symposium (ISRR’95), G. Giralt and G. Hirzinger, Eds. New York: Springer Verlag, pp. 613–625, 1996.

[5] Peng Qi and Lu Wang, "On simulation and analysis of mobile robot SLAM using Rao-Blackwellized particle filters," 2011 IEEE/SICE International Symposium on System Integration (SII), Kyoto, 2011, pp. 1239-1244.

[6] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, “Fast-SLAM: A factored solution to the simultaneous localization and mapping problem,” in Proc. AAAI Nat. Conf. Artif. Intell., 2002, pp. 593–598.

[7] H. C. Lee, Seung-Hwan Lee, Tae-Seok Lee, Doo-Jin Kim and B. H. Lee, "A survey of map merging techniques for cooperative-SLAM," 2012 9th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI), Daejeon, 2012, pp. 285-287.

[8] J.W. Fenwick, P.M. Newman, J.J. Leonard, "Cooperative Concurrent Mapping and Localisation," IEEE International Conference on Robotics and Automation, Washington, DC, 2002

[9] A.I. Mourrkis, S.I. Roumeliotis, "Performance Bounds for Cooperative Simultaneous Localisation and Mapping (C-SLAM)," Robotics: Science and Systems Conference, Massachusetts, 2005

[10] M. Walter, J. Leonard, "An Experimental Investigation of Cooperative SLAM," 5th International Symposium on Intelligent Autonomous Vehicles, Lisbon, 2004

[11] S. Thrun, Y. Lui, "Multi-Robot SLAM with Sparse Extended Information Filters," International Symposium on Robotics Research, Siena, 2003

[12] E. Nettleton, S. Thrun, H. Durrant-Whyte, S. Sukkarieh, "Decentralised SLAM with Low-Bandwidth Communication for Teams of Vehicles," 4th International Conference on Field and Service Robotics, Yamanashi, 2003

[13] S. Ong, M. Ridley, J.H. Kim, E. Nettleton, S. Sukkarieh, "Six DoF Decentralised SLAM," Australiasian Conference on Robotics and Automation, Brisbane, 2003

[14] A. I. Mourikis and S. I. Roumeliotis, Predicting the Performance of C-SLAM, International Journal of Robotics Research, Vol. 25, pp. 1273-1286, 2006

[15] K. Konolige et al., Map Merging for Distributed Robot Navigation, IEEE/RSJ International Conf. Intelligent Robots and Systems, Las Vegas, NV, 2003.

[16] H. S. Lee and K. M. Lee, Multi-robot SLAM Using Ceiling Vision, IEEE/RSJ International Conf. Intelligent Robots and Systems, St. Louis, MO, 2009.

[17] X. S. Zhou and S. I. Roumeliotis, Multi-Robot SLAM with Unknown Initial Correspondence: The Robot Rendezvous Case, IEEE/RSJ International Conf. Intelligent Robots and Systems, Beijing, China, 2006

[18] F. Tungadi et al., Robust Online Map Merging System using Laser Scan Matching and Omnidirectional Vision, IEEE/RSJ Int. Conf. Intelligent Robots and Systems, Taipei, Taiwan, 2010.

[19] S. Carpin, Fast and Accurate Map Merging for Multi-Robot Systems, Autonomous Robots, Vol. 25, pp. 305-316, 2008.

[20] H. C. Lee and B. H. Lee, Improved Feature Map Merging using Virtual Supporting Lines for Multi-Robot Systems, Advanced Robotics, Vol. 25, No. 13-14, pp. 1675-1696, 2011

[21] O. Souissi, R. Benatitallah, D. Duvivier, A. Artiba, N. Belanger and P. Feyzeau, "Path planning: A 2013 survey," Proceedings of 2013 International Conference on Industrial Engineering and Systems Management (IESM), Rabat, 2013, pp. 1-8

[22] P. Hart, N. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” IEEE Transactions on Systems Science and Cybernetics, vol. 4, pp. 100–107, 1968.

[23] R. Finkel and J. Bentley, “Quad trees a data structure for retrieval on composite keys,” Acta Informatica, vol. 4, pp. 1–9, 1974

[24] A. Knoll, “A short survey of octree volume rendering techniques,” Proceedings of 1st IRTG Workshop, June 2006.

[25] D. Meagher, “Geometric modeling using octree encoding,” Computer Graphics and Image Processing, vol. 19, pp. 129–147, 1982.

[26] E. W. Dijkstra, “A note on two problems in connexion with graphs,” Numerische Mathematik, vol. 1, pp. 269–271, 1959.

[27] S. M. Lavalle, “Rapidly-exploring random trees: A new tool for path planning,” Computer Science Dept. Iowa State University, 1998.

[28] A. Stentz, “Optimal and efficient path planning for partially-known environments,” IEEE International Conference on Robotics and Automation, pp. 3310–3317, 1994.

[29] K. S. and L. M., “Incremental A*,” In Advances in Neural Information Processing Systems (NIPS), pp. 1539–1546, 2002.

[30] O. Khatib, “Real-time obstacle avoidance for manipulators and mobile robots,” Int. J. Rob. Res., vol. 5, pp. 90–98, 1986

[31] S. M. Lavalle and J. J. Kuffner, “Randomized kinodynamic planning,” IEEE International Conference on Robotics and Automation, vol. 1, pp. 473–479, 1999.

[32] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” International Journal of Robotics Research, vol. 30, pp. 846–894, 2011.

[33] Peters, Tim (19 August 2004). "PEP 20 – The Zen of Python". Python Enhancement Proposals. Python Software Foundation. Retrieved 24 November 2008

[34] DeBill, Erik. "Module Counts". Retrieved 20 August 2016, from www.modulecounts.com.

[35] PyGame, “About – wiki”, https://www.pygame.org/wiki/about?parent

[36] Student. “Probable Error of a Correlation Coefficient.” Biometrika, vol. 6, no. 2/3, 1908, pp. 302–310., www.jstor.org/stable/2331474.

[37] S. Riaz un Nabi Jafri, W. Ahmed, Z. Ashraf and R. Chellali, "Multi robot SLAM for features based environment modelling," 2014 IEEE International Conference on Mechatronics and Automation, Tianjin, 2014, pp. 711-716