Collaboration of Multiple Agents for Exploration and Mapping

A thesis report submitted for BTP phase II

by

Apoorva Kumar

(Roll No. 160108010)

And

Anupam Khandelwal

(Roll No. 160108008)

Under the guidance of

Dr. Indrani Kar

DEPARTMENT OF ELECTRONICS & ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

November 2019

Abstract

Map exploration has been an important task in the field of robotics and a substantial amount of work has been done towards this. From all the work that has been done, Rapidly Exploring Random Tree(RRT) and Sensor-Based Random Tree(SRT) has seen lot of development in the recent decades. Frontier detection has been another method of exploration which proves vital to mark areas of information gain. Multi-robot systems or swarm systems have emerged a lot in the recent years where each task is divided among multiple robots.

Here we state the use of both the above mentioned techniques to build an exploration algorithm which can be implemented on multi robot systems and can be used to map an area efficiently. SRT will be used for a robot's motion while RRT will be used on a soft agent to detect the frontiers. The frontiers will be later assigned to the robots for exploration after they have finished mapping an area to prevent inefficiency for which we will have both local and global assignment metric for efficient overall motion. The whole system will be modular i.e. independent of the number of robots so that it can be scaled without any hassle. Next we build the complete map for general use by clustering and connecting disconnected regions and building a path planning algorithm for the same map. Finally we test the algorithm in multiple environments to prove its validity.

Contents

List of Figures

Nomenclature

SRT	Sensor Based Random Tree
RRT	Rapidly exploring Random Tree
SMF	Sampling Based Multi Tree Fusion
UAV	Unmanned Aerial Vehicle
MRS	Multi Robot Systems
LSA	Local Safe Area

Chapter 1

Introduction

People have long dreamt about machines having abilities to perceive and understand the environment and react intelligently towards it. This gave rise to the field currently known as robotics. Initially, started under the notion of Artificial Intelligence this field gained relevance and in the early 90s when people started noticing patterns in natural behaviour which could be replicated easily in machines. One such phenomenon was the collective and collaborative behaviour of natural beings like ants, bees and fishes to achieve tasks which when other done alone would be utterly impossible. This set phenomenon gave rise to the field of robotics known as *Cooperative Multi Agent Robot Systems* or in layman terms *Swarm Robot Systems*.

These systems are mainly aimed at building machines that could understand each other and distribute tasks among each other according to needs and requirements, thus achieving tasks that would otherwise take an exponentially long time when performed by a single agent and in some cases, even impossible. Here we worked towards solving one such task exercising the power of MRS to explore unknown area [9] and map it efficiently in the shortest possible time. Such systems have heavy use in the field of military and rescue [15], where time is the prime factor. Multi UAV systems can be used to scan battlefields for hidden enemies or in avalanche-prone areas to find people stuck in debris. Also, while offering a reduction in time, MRS are fail-safe in case one of them malfunctions the others can still work and finish the task assigned.

Nowadays, another requirement is the efficiency and needs to save data stored while getting the maximum output. The algorithms proposed before stored all the data with each robot while also stored the information of the area irreverent for exploration. However, with the emergence of cloud and efficient data-transfer mechanisms, data need not be stored everywhere and can be retrieved on the go as and when the required so that the robots themselves can expend all their computation towards perceiving the environment and extracting data from their present while not caring about the past.

In this report, we present the RRT and SRT exploration algorithm for unknown area exploration. RRT and SRT both of them are algorithms highly biased towards exploring unknown area [1] [3]. Adding the technology of Frontier based exploration helps us to weight this search more towards reaching areas left unexplored before. In Chapter 2, we discuss the relevant literature regarding our proposed algorithm. Chapters 3 discuss the main algorithm which we will be using to build upon, while Chapter 4 and 5 present different aspects of our algorithm. Chapter 6 finally concludes the report with experimentation results.

Chapter 2

Related Work

The work of robotic motion planning bases its roots on ways to perceive and understand its environment. The first stage towards this is the exploration of an area in which the agent needs to work and assess. Several algorithms were proposed over the past two-three decades for exploration path planning, but the one which has stood out a lot was proposed by Lavalle in 1998 known as *RRT* [1]. This is a randomized search technique biased towards unexplored areas while also being computationally light and formed the basis of many later algorithms. RRT was also shown to be complete and would always end up completely exploring the area. After the proper advent of advanced sensors, Oriolo et al. proposed a new algorithm the *SRT* [3] which used the data from sensors to map the explored area while still maintaining the computation efficiently.

Another algorithm used appointed the use of *frontiers* [4] which are edges connecting explored and unexplored areas to favour biased exploration towards unknown area. After the emergence of SRT and RRT attempts were made to combine the two algorithms to assign robots, area of exploration based on detected frontiers [8] [2]. Frontiers helped keep track of areas which were left behind unexplored [17] by the robot. Backtracking algorithms [5] [14] were also important so that frontiers can be explored and exploited for faster continuous area cover resolving frontiers.

After the perfection of exploration on a single robot, attempts were made to apply the exploration algorithms using MRS [9] [10] to combine and stitch the maps provided by each of them while maintaining the rule that robots don't waste time travelling in a known area. Fusion of multiple trees using multi-tree fusion leading to completeness and resulting in maps which can later be explored by a single robot without any help.

Here we proposed to work on combining the work leading from all these three fields namely the work of multiple SRT [6] while using frontiers to keep mark of areas [12] [13] which have been left behind. We also worked to minimize the data being stored and mark areas which would be rendered useless for exploration [11] as they don't contain any nodes which would lead to new exploration points. This would help save time and data as robots would only have points to travel along and closer to the frontiers. UAVs [12] [13] are preferred for this task as they have a broader vision and perception over ground robots while facing fewer obstacles but this algorithm works for any kind of automated agent.

While the systems stating efficient exploration of areas have been well established, using multiple agents for this work is something which has not been worked upon. As new and efficient technologies emerge, this can help change the scenario totally as to how robots can be used to achieve task and ease tasks for humans.

Chapter 3

Random Tree Based Exploration

RRT [1] and SRT [3], as stated before, will be discussed here with requirements and assumptions. As they are both tree-based path planning algorithms, we will be common terminologies while other requirements will be specified on the go. Both these algorithms were designed for single robot exploration, and we will later state the modifications to the algorithm for multi-robot exploration [9].

3.1 Terminologies for Path Planning

The following preliminary terminologies which are used in path planning algorithms using tree based structure.

Map X: The set of total space containing occupied *(obstacles)*, unoccupied *(free)* space and unknown *(unexplored)* area.

Occupancy Map M: A geometric division of the continuous map into squares of a selected dimension where each square either denotes one of the three states namely, 1 for occupied, 0 for free and -1 for unexplored.

Free Space X_{free} : The space not occupied by obstacles.

Occupied Space $X_{obstacle}$: The space occupied by obstacles.

Known Region X_{known} : The space combining the X_{free} and $X_{obstacle}$ to form the subset of *Map* known by the robot.

3.2 RRT : Rapidly Exploring Random Tree

Introduced by S. Lavalle [1] in 1998. First we will discuss the assumptions pertaining to this algorithm and finally we will state the algorithm and proceed with its discussion.

3.2.1 Assumptions

The following assumptions about the robot and its environment need to be taken into consideration when studying the algorithm.

- All agents travel in a *planar or* $\mathbb{R}^2$ *plane* and do not move in vertical directions except when landing and launching.
- The agent is always aware of its configuration in the environment.
- The agent's configuration space is a subset of the environment, and it is in the form of a disk or *holonomic* i.e. it can translate in any direction possible.

3.2.2 Terminologies

Here we state some functions and variables pertaining to the RRT algorithm.

Vertices V: Any sample node or point in the space is known as vertex. It is represented in terms of coordinate. All elements of form $x_{something}$ is belong to this set V.

Edge E: A branch connecting two coordinates from the space V is called an edge. The set of all these edges constitute *E*.

Graph G: The set of edges and vertices form the Graph $G = (V, E)$. The whole structure of SRT and RRT are some form of this Graph G.

RandomSample: A function that returns random points which are independent and identically sampled around the given position *x*

Closest$(Graph, x \subset X_{free})$: A function that takes a $Graph = (V, E)$ and a point x in the free space as input and returns a vertex $v \subset V$ to satisfy the condition $argmin_{v \subset V} \|v - x\|$. If multiple v satisfy this condition then the one sampled first will be selected.

NextVertex: Takes two points x_1, x_2 and returns a point x_{next} such that $\|x_{next} - x_2\|$ is minimized and $\|x_{next} - x_1\| < \delta$ where δ is the maximum length of tree edge towards exploration.

Valid: Takes two points x_1, x_{next} and a map X and returns 1 if its possible to travel from x_1 to x_{next}, 0 if there is an obstacle in between them and -1 if there is an unexplored area between them.

Algorithm 1: RRT: Rapidly Exploring Random Tree

```
1  Build RRT($x_{init}, \delta$):
2      $V \leftarrow x_{init}$ ; $E \leftarrow 0$
3      while True do
4          $x_{rand} \leftarrow$ RandomSample
5          $x_{nearest} \leftarrow$ Closest ($G(V, E), x_{rand}$)
6          $x_{new} \leftarrow$ NextVertex ($x_{nearest}, x_{rand}, \delta$)
7          if Valid($map_X, x_{nearest}, x_{new}$) =1 then
8              $V \leftarrow V \cup x_{new}; E \leftarrow E(x_{nearest}; x_{new})$
9          end
10     end
11     return $G = (V, E)$
12 end
```

3.2.3 Algorithm

We start by initializing a tree with the present position of robot x_{init} and then run a loop until the full map is not constructed. The algorithm samples a random point using x_{rand} and then finds a point in the graph $x_{nearest}$ which is closest to this point. Next, we try to find a point which to which the robot can travel x_{new} and then check if it is possible to travel to this point or not. If it is possible we add this new point to this set V and add an Edge between $x_{nearest}$ and x_{new} indicating a path between them. Random Sampling of points helps to grow the tree to new points while maintaining the constraint of exploring the area in fixed steps. The value of δ controls how fast the tree will expand and needs to be altered depending on how complex or simple the environment is.

3.3 SRT : Sensor Based Random Tree

G. Oriolo [3] first came up with this idea of maintaining the input from sensors to keep track of areas travelled so that we can tackle the sub-optimality of RRT while also build a map and use the random biasing of RRT towards exploration [18]. This algorithm has some extra assumptions over those of RRT which is stated below.

3.3.1 Assumptions

Other than the assumptions from RRT the following extra assumptions about the robot and its environment needs to be taken into consideration for the algorithm.

- The robot has full data about its surrounding from its sensor which always a subset of the final original map. The sensor gives an estimate of the LSA in which the robot can translate.

3.3.2 Perception Method

The perception technique used by the robot to perceive its environment is an integral part of this algorithm. The robot's present location x_{curr} with the perceived environment forms the Node data-structure of the tree. The following are the two algorithms:

- **SRT-Ball**: In this perception algorithm, the LSA is a circular area of radium r around the robot. The area is that the robot can traverse to any point in that area. No obstacle needs to lie inside that area of radium r. It is a conservative and safe exploration technique so that the robot takes only fully safe step. It is shown in Figure 3.1a
- **SRT-Star**: This is a comparatively more confident approach where we make use of the directional nature of sensors and divide the perceived area into arcs. The radius of each arc is the minimum of the farthest obstacle in that arc and the sensor range. Figure 3.1b shows the radius of arc are.

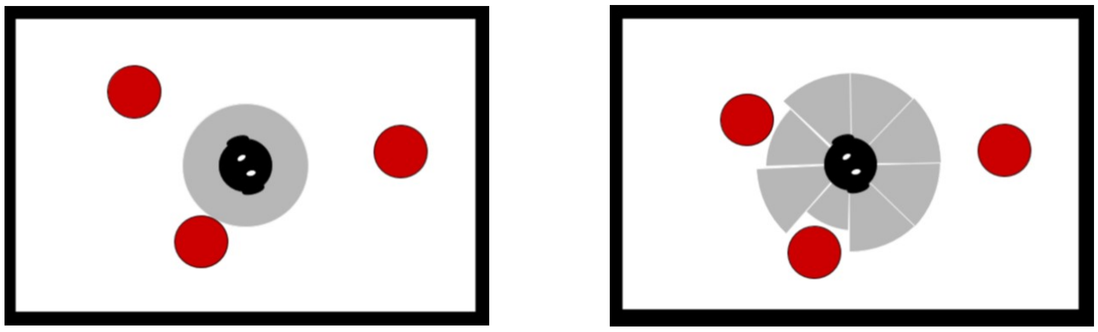

(a) SRT-Ball technique (b) SRT-Star technique

Figure 3.1: SRT Perception Techniques

3.3.3 Terminologies

Here we state some functions and variables pertaining to the SRT algorithm.

Scan A: Returns the scan perceived by the robot from its present position and returns it as a $map \subset X$ which can be concatenated to already known region.

Position P: Set of all points on map to which the robot has travelled. States the location on the map and is represented of form $p_{something}$. It is similar to X from RRT.

GraphMap τ: $GraphMap = (p \subset P, A_p)$ is modified data-structure which consists of points with scanned area around it. It is build by merging the position p_{scan} and the scan associated with it together.

RandomSample: A function that returns a random angle which is independent and identically sampled around the given position x

Ray: Takes an angle ϕ and area A and return the maximum radius possible in the LSA

Displace: This function takes a location p, a distance r, an angle ϕ and a constant factor α, and returns a point p_{next} in the ϕ direction at a distance $\alpha.r$ .This is done so as maintain safe travelling distance from the present location.

Move: Moves the current robot to the new position specified

Valid: Takes two points x_1, x_{next}, a map τ and a minimum distance δ_{min} and checks two condition. First that the distance between x_1, x_{next} is more than d_{min}. Secondly that the second point x_{next} doesn't lie in LSA of any other previously explored point.

Algorithm 2: SRT: Sensor based Random Tree

```
1  Build SRT ($p_{init}, \alpha, \delta_{min}, K_{max}, I_{max}$):
2      $p_{curr} \leftarrow p_{init}$ ; $\tau \leftarrow \emptyset$
3      for $k$=1 to $K_{max}$ do
4          $A \leftarrow$ Scan ($p_{current}$)
5          $\tau \leftarrow \tau \cup (p_{curr}, A)$
6          $i \leftarrow 0$
7          do
8              $\phi_{rand} \leftarrow$ RandomSample
9              $r \leftarrow$ Ray ($\phi_{rand}$,A)
10             $p_{cand} \leftarrow$ Displace ($p_{curr}, \alpha, \phi_{rand}, r$)
11             $i \leftarrow i + 1$
12         while Valid ($p_{cand}, p_{curr}, \delta_{min}, \tau$) $\neq 0$ or $i = I_{max}$
13         if Valid ($p_{cand}, p_{curr}, \delta_{min}, \tau$) $\neq 0$ then
14             Move ($p_{cand}$)
15             $p_{curr} \leftarrow p_{cand}$
16         else
17             $p_{curr} \leftarrow p_{curr}.parent$
18             Move ($p_{curr}.parent$)
19         end
20     end
21     return $\tau$            // Return final Scanned Tree
22 end
```

3.3.4 Algorithm

The algorithm starts with us initializing the current position of agent p_{int} and a new *GraphMap* τ. The area around the current position is scanned and added to the map. After that, we start a loop to sample a new point to which the agent needs to travel. This sampling is done by first taking a *RandomSample* angle ϕ_{rand} and then selecting a point on a line connecting our present location to the farthest obstacle-free point in the direction ϕ_{rand} which is returned as the length

r using *Ray* function.

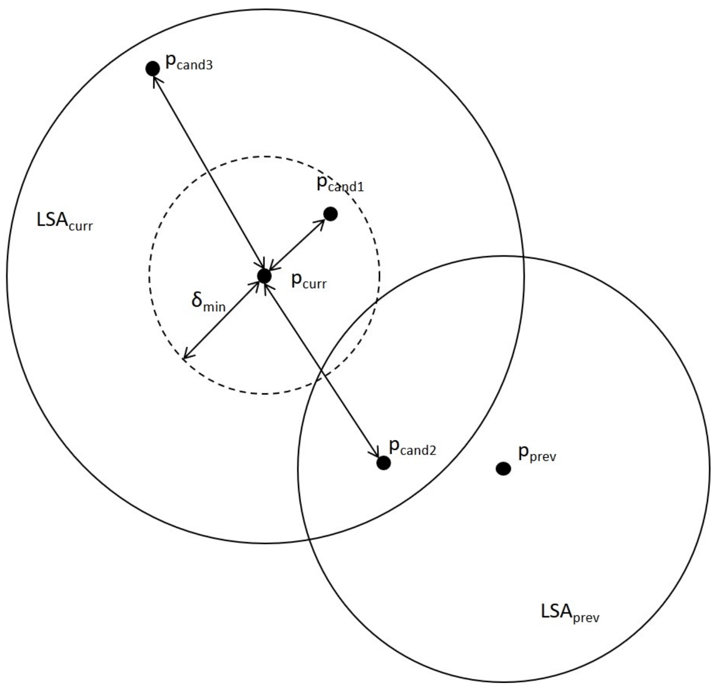


Figure 3.2: Validation function Implementation of SRT

The new candidate point is sampled using *Displace* function to a distance $\alpha.r$. New points are kept being sampled unless either a *Valid* candidate point is found or the number of iterations reaches I_{max}. The condition of δ_{min} is kept to take in consideration the size of a robot and a set distance outside which the robot must move so that the map expansion is not slow. An example of validity is shown in Figure 3.2, where three points are sampled one after the other.First point p_{cand1} is rejected because its inside δ_{min} and the second point p_{cand2} is rejected because it lies in LSA of some previous node. Finally, a third and final point p_{cand3} is selected, and the robot travels to that location. The limit to the number of points being sampled is kept to as to see if the robot reached the dead-end of a region which is surrounded by explored region. If a valid point is found the robot then uses the *Move* function to move to that point and if no valid point is found it retreats to its parent and starts searching for new points there.

3.4 Advantages

SRT and RRT offer a range of advantages for exploration namely:

- SRT is highly biased towards exploring the area which hasn't been explored before and backtracks only in sporadic cases. This depth first nature proves to very optimal for exploration [18].
- Being a modular program SRT can easily accommodate non-holonomic motion by editing it *Ray* and *Displace* function.
- These algorithms not only travel but also store map while exploring thus can be easily used for mapping and environment optimally.

3.5 Modification to Validation Algorithm

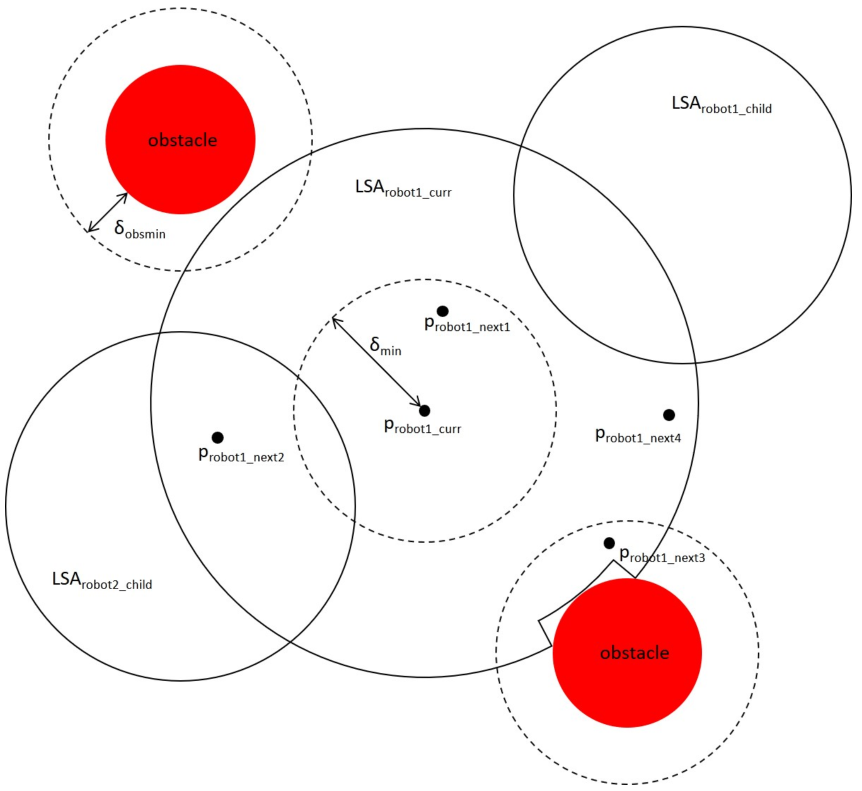


Figure 3.3: Modified Validity Condition for SRT

To accommodate multi-agent systems, we need to introduce some changes in the validations algorithm of SRT which are stated below:

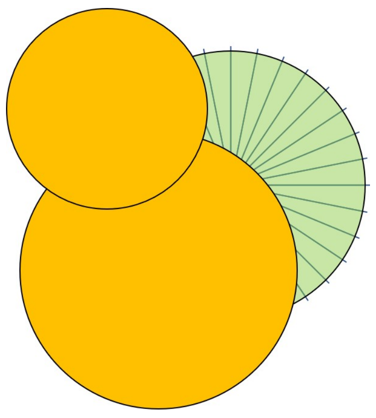
- The sampled point must not lie in the LSA of the any previously visited node and should also not lie in the LSA of any other robot and or its node.

- The sampled point must be at a safe distance from any obstacle of δ_{obsmin}, which is generally determined by the dimensions of the robot.

Both these new validation algorithm allow us to take a path which does not intersect with the area of any other robot while also dodging obstacle when moving past it, which was not tackled in the original algorithm. As we can see in Figure 3.3 point p_{robot1_next1} is invalid due to original validity condition while p_{robot1_next2} is invalid because it lies in LSA of another robot while p_{robot1_next3} is invalid because it lies in δ_{obsmin} from an obstacle. Finally p_{robot1_next4} is valid for travel.

3.6 Modified Backtrack Strategy

The present backtrack strategy makes the physical robot move back each step and sample points until it reaches a stage from where it can progress in a new direction. Various situations arise in a real-world scenario where this might cause the robot to do much backtracking before reaching a node with good information gain and thus consume much time. This can be solved by using software agent which will backtrack in real-time from the present robot location until it finds a node which has information gain more than a threshold G_{thresh} or has a *Frontier Point* on it. The concept of Frontier Points will be explained in Chapter 4. Below we state the changes to do in Algorithm 2, Statement 17


(a) Node with low Information Gain

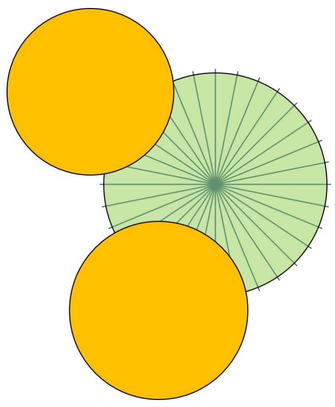

(b) Node with high Information Gain

Figure 3.4: Threshold measurement using sensor simulation

Algorithm 3: Modified Backtracking for SRT

```
    // Replace Algo 2, Statement 17 with this code
 1  do
 2      $q_{test} = q_{curr}$
 3      $G \leftarrow$ GetGain($q_{test}$)
 4      $F \leftarrow$ GetFrontiers($q_{test}$)
 5      $q_{curr} \leftarrow q_{curr}.parent$
 6  until $G \geq G_{thresh}$ or $F > 0$ or $q_{curr}.parent == NULL$
 7  if $G \geq G_{thresh}$ or $F > 0$ then
 8      Move($q_{test}$)
 9      $q_{curr} \leftarrow q_{test}$
10  else
11      Move($q_{init}$)
12      $q_{curr} \leftarrow q_{init}$
13  end
```

The calculation is G is done by simulating a sensor by going back to that node and checking out the LSA of the node which does not overlap with the LSA of any other node. Figure 3.4 shows the cases of high and low threshold for G_{thresh}

Chapter 4

Frontier Based Exploration

Frontiers stand as another exciting technique for robotic exploration. This method works on the simple fact that the edge between a known and unknown region [4]. Since its advent in 1998, it has gained immense importance and is still regarded as a complete method for robotics exploration. Here I would like to explain a technique in the field of Frontier Detection and later state how they are going to be used.

4.1 Introduction

Frontiers are boundaries between the known and unknown region. These boundaries are marked as regions of maximum information gain as agents travelling to this region will be able to explore new areas. Points are sampled from these edges to which the agents can travel. Frontier detection can be performed in multiple ways. As the map grows, detecting frontiers become a tedious task, and we need to perform it more efficiently for fast exploration.

4.2 Frontier Detection Using RRT

RRT is an algorithm which, as stated before, is biased towards optimal exploration. We will be using RRT to detect frontiers [2] in space. In the method stated below the agent has its motion algorithm while RRT is used to detect frontier on already scanned Maps. After frontiers are detected, they are assigned to each agent based on a cost function to which they need to go and perform exploration.

Algorithm 4: Frontier Detection Using RRT

```
1  Find FrontierRRT($x_{init}, \delta$):
2      $V \leftarrow x_{init}$ ; $E \leftarrow 0$
3      while True do
4          $x_{rand} \leftarrow$ RandomSample
5          $x_{nearest} \leftarrow$ Closest $(G(V, E), x_{rand})$
6          $x_{new} \leftarrow$ NextVertex $(x_{nearest}, x_{rand}, \delta)$
7          if Valid($map_X, x_{nearest}, x_{new}$) =-1 then
8              PublishPoint($x_{new}$)
9              $V \leftarrow x_{current}; E \leftarrow 0$      // not in global detection
10         else if Valid($map_X, x_{nearest}, x_{new}$) =1 then
11             $V \leftarrow V \cup x_{new}; E \leftarrow E(x_{nearest}; x_{new})$
12         end
13     end
14 end
```

4.2.1 Algorithm

Two kinds of frontier detectors run simultaneously on the map to detect frontiers. Namely the local and the global frontier detectors. Below is the explanation for each.

Local Frontier Detection: Frontier detection using RRT as proposed in the above algorithm starts at the present location of the agent x_{init}. Next we sample a random point $x_{rand} \subset X_{free}$ and find the point on the graph $x_{nearest}$ to that point using the *Closest* function. Finally, the *NextVertex* function is used to choose a point under a considerable distance to travel as the next position. After this the function enters the Validation phase and checks if there lies an unknown region in between $x_{nearest}$ and x_{new} or if x_{new} itself lies in an unknown region. In either of the cases, the function marks the point as lying on a frontier using the *PublishPoint* function and resets the tree. It is shown in Algorithm 4

Global Frontier Detection: Global Frontier uses the same Algorithm as Local Frontier Detection with the minor change that it does not reset the tree after a frontier is detected and continues to grow until it has scanned the full map. The Statement 9 from Algorithm 4 needs to be removed.

Filter Module is also implemented in the *PublishPoint* functions which clusters multiple close frontiers and outputs only one point from the cluster to the agent task allocator. This necessary, otherwise too many closely placed points will be assigned to multiple agents and will result in sub-optimal exploration. This module also keeps removing frontier points which have been explored and have known region around them.

4.2.2 Advantages

The stated Algorithm contains two different methods of frontier detection, and here we state the needs and requirements of both the method and how they each have their use:

- Local Frontier Detection always starts as the present location of the agent so it is more likely to sample points which will be closer to the agent and thus helps in finding frontier points which can be efficiently explored. Also, as the tree grew from the agent's current position, it will be easier for the agent to backtrack to the detected frontier.
- Global Frontier Detection, on the other hand, is needed for points which might be left undetected by the agent while agent would have moved away from such high information gain regions. Also, these help us finds explored region near corners of maps which are missed by SRT or other exploration algorithms often. [2]

It is essential to detect all possible frontiers on the map as they inform the location of the points which were left unexplored thus both of these have their importance and absence of either can lead to sub-optimal motion. In major cases, the global frontier detector is implemented on the master agent, and it assigns the detected frontiers to slave agents for exploration based on a cost factor of information gain whereas the local frontier detector can be implemented on the slave agents individually who publish the detected frontiers to the master for assignment.

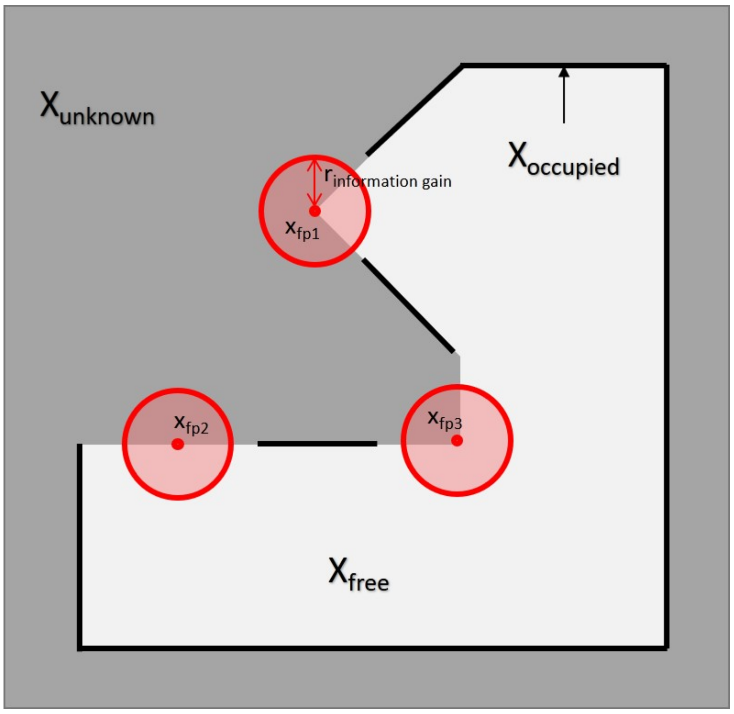


Figure 4.1: Frontier Detection and Information Gain

4.3 Cost of Information Gain and Frontier Assignment

Frontiers, after being detected, are assigned to the exploring agents for exploration based on the cost of information gained from which the cost of travelling to the location is subtracted [2]. Both the information are explained below briefly.

Navigation(N) Cost is the cost of an agent travelling from its present location to the given location. In standard cases, it can be estimated by taking the norm of coordinates of the agent and the frontier and taking its square. We take square of the distance. The reason for this will be explained in a further section soon. For a more accurate estimate, the graph can be traced back from the agent's present location along the local detected frontier RRT and add the total travel cost as navigation cost.

Information(I) Gain is the amount of information which will be gained by reaching that point. It is estimated by drawing a circle around the frontier point and estimating the area of the circle which is unknown. Shown in Figure 4.1 $r_{informationgain}$ shows this radius. This gives a fairly accurate estimate of how much new information will be gained by reaching that point. The radius of this circle is mostly the radius of the sensor range to replicate the information estimated by the agent. As we can see in Figure 4.1 three points are sampled out of which x_{fp1} has maximum information gain while x_{fp3} has minimum information gain. Black occupied cells denote obstacles or occupied area while the white area shows the free region and grey area shows unknown region.

$$R(x_{fp}, x_r) = \lambda h(x_{fp}, x_r) I(x_{fp}) - N(x_{fp}, x_r) \tag{4.1}$$

$$N(x_{fp}, x_r) = \beta d^2(x_{fp}, x_r) \tag{4.2}$$

$$h(x_{fp}, x_r) = \begin{cases} 1 & \text{if } \|x_r - x_{fp}\| > h_{rad} \\ h_{gain} & h_{gain} > 1 \end{cases} \tag{4.3}$$

The following equation is used to estimate the **Revenue from a frontier point**. The value of $R(x_{fp})$ tells this value as a function of navigation(N) and information(I) cost. The value of λ and β is used to make *I* and *N* comparable. Hysteresis Cost h is either 1 if the frontier point is far from the present agent location by more than user estimated distance h_{rad} or is equal to h_{gain} which should be greater than 1 to if its under h_{rad} range of the agent. h_{gain} should be more than 1 to ensure that the agent explores points closer to it. The Revenue for each new frontier point is calculated for each free agent from the swarm, and the agent, which scores the maximum Revenue is assigned to explore this point.

4.3.1 Strict Frontier Assignment

RRT will also be used as a software agent to explore and mark frontier points, as mentioned above. The marked frontiers will be assigned to agents based on the revenue gained by making an agent to travel to that point.

A small modification is made to the agent assignment method, i.e. agents are assigned to explore a frontier point if and only if the revenue gained from that points is above some R_{thresh} (decided by user). In-case none of the agents has their Revenue gain from the frontier above this no agent will be assigned to explore that frontier, and the frontier will be there until some agent crosses R_{thresh} for that frontier.

This algorithm is going to work from the fact that if the agent is close to the frontier, the revenue will be high and they will be assigned, one having the highest revenue gain. Incase the agents are far away from the frontier; two cases may arise.

Case 1 arises when the agents have regions close to them which are left for exploration or have frontiers closer to them. In either case, assignment of frontiers to any of these agents would be sub-optimal.

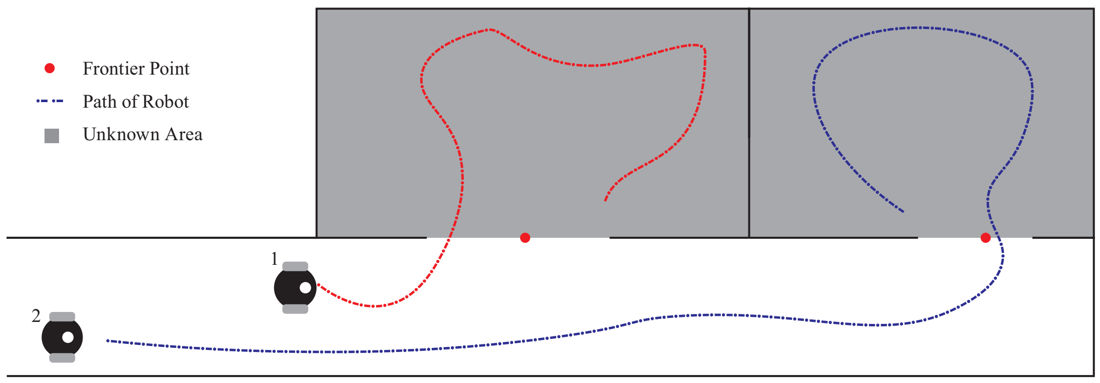
Case 2 arises when the agent's close by areas do not have regions for exploration, and it is going to backtrack. In that case, the agent will backtrack itself to this frontier itself for exploration, and thus, they will eventually end up exploring it even without assignment.

From the above cases, we see that in either case, this task assignment function is optimal and works in all the scenarios considered. The task allocator will be checking the gain for each agent w.r.t to a frontier and also vice versa so that if a new frontier is found closer to an agent who was assigned another frontier, then the agent can change its path and move more optimally to this new frontier.

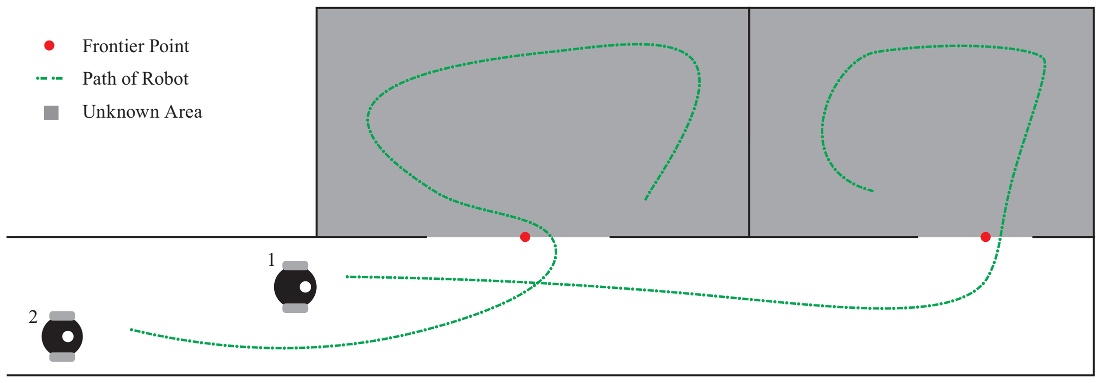
4.3.2 Global Frontier Assignment for Multi-Agent Systems

Now we see that all the whole job of assigning frontiers has been developed for a single agent as of now and the way to scale up might seems simple initially with the fact that we need to run the said algorithm for each agent once. But there are some cases which arise and cause this assignment to work really suboptimally and thus slow down the search. One such case is presented in Figure 4.2.

Here we see two cases of frontier assignment. In the first case shown in Figure 4.2a we see that the time taken by both the agents to completely map the whole region is much higher than the time which is taken by them to achieve same task if the assignment of frontiers were switched as shown in Figure 4.2b. Now we see that the problem posed here was caused by pre sub-optimal assignment of frontiers to agents. This brings us to the question of whether we should refresh the frontier assignments after very interval and also the fact that the agents are actually working in coorperation so the total time taken by them to map and area is determined by the agent which finishes its work last. To take into consideration the fact that the final time taken for the agents to complete the job is decided by the $\mathrm{argmax}_i(t_i)$ i.e. the maximum time taken or the longest trajectory travelled. Therefore to pay penalty for long trajectories, we take the square of the total trajectory distance as shown in Equation 4.2. Now to we will deal with



(a) Sub-Optimal Path



(b) Optimal Path

Figure 4.2: Global Frontier Assignment

the problems of assignment one by one.

First taking the problem of coordinated motion we can see that the total time taken can be considerably reduced if both the agents take a routes which is optimal overall rather than optimal to their own self. This statement itself calls for collaboration which we will perform by taking the total summation of Revenue gained by the agents when the search a frontier i.e. the result of their collaborative work. Lets define some terms first.

$$a_i = \text{a complete set of frontier assignment} \tag{4.4}$$

$$a^n_{r,i} = \text{ assignment of } n^{th} \text{ frontier to } r^{th} \text{ agent in the } i^{th} \text{ set of assignment} \tag{4.5}$$

$$^nP_r = \frac{n!}{(n-r)!} = \text{permuatation} \tag{4.6}$$

Here a_i denotes a full set of assignment where either all the agents are assigned to some frontier if the number of agents are less than the total number of frontiers or vice-versa. In the case of when total number of agents(r) are are less than frontier(n) we get the count of possible sets a_i

i.e. all set of assignment by ${}^{n}P_r$ and when the total number of frontiers are less than agents we get the count of all possible sets a_i as ${}^{r}P_n$. Now we can define our modified assignment metric as:

$$\underset{[a_1,a_2,\ldots]}{\operatorname{argmax}} \sum_{a^n_{r,i} \in a_i} R(a^n_{r,i})$$

What we are doing here is maximising the sum of all revenue values rather than focusing on the revenue of each agent receptively. This way we will find through all possible assignment that one assignment which gives maximum output.

We can also see that the total number of possible sets to iterate through can grow exponentially as the number of agents or frontiers increase. To reduce the size of set a_i we will use the R_{thresh} established in Section 4.3.1 and if any $R(a^n_{r,i})$ is less than R_{thresh} we will remove that full set from a_i.

Now we come to the problem of how many times do we need to refresh the assignment algorithm or in other words reassign the frontiers to the agents. We chose to rerun the assignment when one of more of these events occurred in the environment.

A new frontier is found. This cases seems like the most inevitable case which needs to trigger the algorithm because a new frontier can surely change the assignment of agents to them.

An agent starts backtracking. In this case rather than causing the agent to backtrack which can take a considerable amount of time to travel we can directly assign it a frontier to take care of.

A frontier is explored. This trigger is one of the most important as there can be cases when there are two frontiers close by assigned to different agents. Now if one agent has cleared one of those frontiers it will surely have a lower revenue if the other frontier was assigned to it rather than any other agent.

Iterating at a constant interval is another option but it poses the problme of sub-optimal allocation if too many updates occur on the map between two assignment. It also causes the problem of too many paths change incase the map doesn't update very often then computation power is wasted and as the set a_i increases computation power becomes something which needs to be looked after.

Chapter 5

Search Processing & Path planning

After the final map is built by the agents we need to ready it up for traversal by other agents who later use the map. Below we list a set algorithms which are essential to be applied to the map before it can be used by any other agent.

5.1 Clustering for Linear Motion Regions

After the map is built, it will be used by some agent to traverse it when it needs to travel from one location to another. To reduce this travel time, we need an optimal path rather than a tree-based structure which can be used for moving into the map. So to ease that motion, we will cluster the points on the map into what we like to call *sections of linear travel*. This clustering of points is performed in such a way that in a single cluster, you can move from one point to another in a linear motion without taking care of the map. Figure 5.1 shows how a small direct path if travelled along the tree would take a lot more time and is not even close to optimal.

Now, on the other hand, if these points and their corresponding LSA were clustered into a single large LSA, it would facilitate a lot of factors for both the algorithm and the frontier search algorithm also. As we cluster the points one point becomes obvious that if a point is surrounded by a sufficient number of other points, then it can be neglected when backtracking or for frontier search and would optimize the search a lot in-case of large maps and make our search quicker as we keep exploring more of the map.

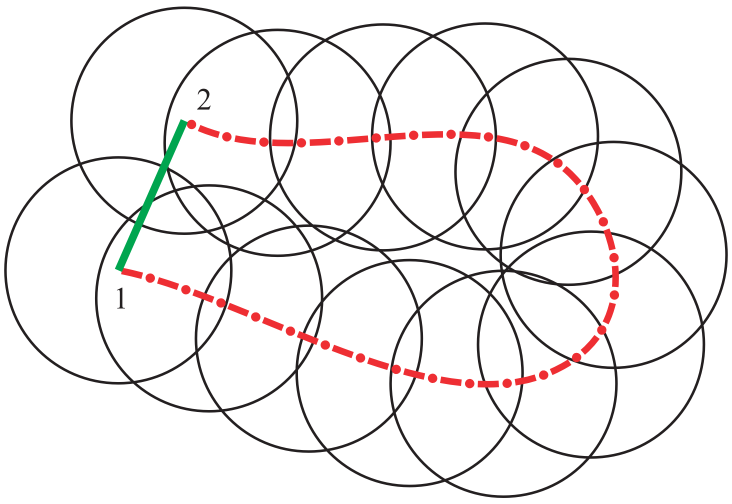


Figure 5.1: Actual Path v/s Optimal Path in-case of Map Traversal

For clustering, we take a point and check whether it can linearly traverse to each and every node on the boundary of the cluster. To understand it better, let's have two kinds of nodes or points in the tree. First, one is boundary nodes or more systematically called $x_{external}$ having the property that they have not enough explored region around them(high probability of having a frontier on it). The other node is $x_{internal}$ are those points, the region around which has been thoroughly explored and they can be skipped for any exploration activity. For now on we will assume that we have a method to classify these nodes, then the Algorithm 5 checks which cluster a newly checked node should be assigned to or if it should start a new cluster.

We first check whether the new node overlaps with any part of the chosen cluster. If that is confirmed we use *CheckLinear* function to check whether the new point x can travel linearly to all other nodes x_{node} in the chosen cluster. If it does so then the *Add*, the function adds the node and its area to that cluster else if none of the clusters satisfies both the condition it initializes a new cluster using *New* function to create a new cluster in the set of clusters. We had to add the condition of overlapping of the nodes with the cluster to avoid the situation of having far away outlier nodes being classified into a cluster. In contrast, it should have been part of a better cluster even though that point can be linearly traversed from this cluster. Also if any such pair of outlier points and cluster occur the path formed between then will be anyway straight as the agent moves from one cluster to another in a linear fashion.

Algorithm 5: Clustering

```
Build Cluster($x, clusters$):
    $A \leftarrow$Scan($x$) ; $clusters_{temp} \leftarrow clusters$
    $Assigned \leftarrow NULL$
    do
        $cluster \leftarrow cluster_{temp}.pop$
        if Overlap($cluster, A$) then
            $Assigned \leftarrow cluster$
            for each $x_{node}$ in $cluster$ do
                if node is internal then
                    continue to line 8
                else
                    if CheckLinear($x_{node}, x$) then
                        continue to line 8
                    else
                        $Assigned \leftarrow NULL$
                        $break$
                    end
                end
            end
    until $Assigned = NULL$ or $cluster_{temp}$ is empty
    if $Assigned = NULL$ then
        $clusters = clusters \cup$ New($x, A$)
    else
        Add($Assigned, (x, A)$)
    end
    return $clusters$
end
```

We also need to establish a connection between two clusters and paths of travel between them which can be achieved by using the connected external or boundary nodes on both the clusters as paths and points of traversal as there will overlapping area between them allowing for a smooth transition from one cluster to another.

To classify a point as to whether it is internal or external, we are using the count of the nodes whose area overlap with the node in consideration. If a node has more than eight LSA overlapping it from its own cluster, then it can be classified as an internal point. To reach this count of eight, we ran multiple simulations, and hand-picked and counted overlapping nodes and started marking them on the map starting from 2 and checked whether those nodes were actually internal or external ourselves. As we increased the count, we saw that at five overlapping nodes more than 97% of nodes were internal with the count rising to 98.4%, 99.6% and 100% with six, seven and eight overlapping nodes respectively. All nodes having eight overlapping nodes were internal. We checked on multiple maps to confirm our result, and it reached 100% at eight nodes. The implementation of the same is also easy as we need to mark two nodes as overlapping whenever in line 6 of Algorithm 5 when we get an output True from the *Overlap* function and we will easily classify.

The clusters then formed are convex polygons, and each of them is stored in memory in terms of their coordinates of the vertices.

5.2 Path Selection

After the map is divided into many such clusters, we can now store the information of map in terms of a graph where each node corresponds to a cluster. This way we greatly reduce the memory consumption required in storing each and every single node traversed by the agent. Path selection is then done using the A* search algorithm [21] [22] described below:

The source node is denoted by $node_{start}$ and the goal node is denoted by $node_{goal}$. We maintain two lists: **OPEN** and **CLOSE**

OPEN: consists of nodes that have been visited but not expanded. This is the list of the pending nodes.

CLOSE: consists of nodes that have been visited and expanded (successors have been explored already and included in the open list, if that was the case).

g value: Denotes travel cost from starting node to current node.

Algorithm 6: A* Search Algorithm

```
Build A_Star($node_{start}, node_{goal}, clusters, map$):
    OPEN ← OPEN ∪ $node_{start}$ ; f($node_{start}$) = h($node_{start}$)
    do
        $node_{curr}$ ← MinimumFValue(OPEN)
        if $node_{curr} = node_{goal}$ then
            break
        end
        for each $node_{next}$ of $node_{curr}$ do
            $cost_{node_{next}}$ = g($node_{curr}$) + w($node_{curr}, node_{next}$)
            if $node_{next}$ in OPEN then
                if g($node_{next}$) $\leq cost_{node_{next}}$ then
                    continue to line 8
                end
            else if $node_{next}$ in CLOSED then
                if g($node_{next}$) $\leq cost_{node_{next}}$ then
                    continue to line 8
                end
                OPEN ← OPEN ∪ $node_{next}$ ; CLOSED ← CLOSED $-node_{next}$
            end
            else
                OPEN ← OPEN ∪ $node_{next}$
                h($node_{next}$) = Heuristic($node_{next}, node_{goal}$)
            end
            g($node_{next}$) = $cost_{node_{next}}$ ; $node_{next}.parent = node_{curr}$
        end
        CLOSED ← CLOSED ∪ $node_{curr}$
    while OPEN is not empty
end
```

h value: Heuristic or estimated cost from present node to the final node.

Heuristic: Takes two nodes as input and calculates the heuristic cost from first node to second node.

w value: Weight or cost of travel connecting present node to one of its immediate successors. In our case it will be the physical distance across map.

MinimumFValue: From the **OPEN** node list, pops and returns the node with minimum f value where $f(node) = g(node) + h(node)$

This way, we obtain the desired clusters (in order) that the agent needs to traverse to reach the destination.

One such advantage of using this search based approach on the cluster nodes based graph is that we do not need to check whether a path between two points exists or not. If the given starting and ending point lie inside any cluster would mean that surely there is a path between them otherwise they wouldn't had come inside any cluster polygon on the first place.

If any of the starting or ending points does not come inside any of the clusters would mean that it was never contained in any LSA while exploring on the first place and hence, no path exists between them. So, we don't need to run the complete algorithm to confirm whether a path may exist or not.

5.3 Estimation of Control Points

Control points are a specific set of points used to parametrize and generate smooth curves for efficient trajectory planning for agents in an environment full of obstacles since a direct straight line path between the starting and destination coordinates might not be possible always.

In this section, we describe how control points are extracted from the clusters; firstly at the transition from one cluster node to another and secondly as the centroid of the clusters itself.

Since the clusters are described by convex polygons, two connected clusters will be the ones that share a common edge or at least some part of it.

So we describe the control point, $C_{transition}$ for transition from one cluster to another be the middle point of smaller edge between the two common edges as shown in Figure 5.2.

$$i = \underset{j\in\{p,q\}}{\operatorname{argmin}} \sqrt{(x_{j1} - x_{j2})^2 + (y_{j1} - j_{j2})^2} \tag{5.1}$$

$$C_{transition} = (\frac{x_{i1} + x_{i2}}{2}, \frac{y_{i1} + y_{i2}}{2}) \tag{5.2}$$

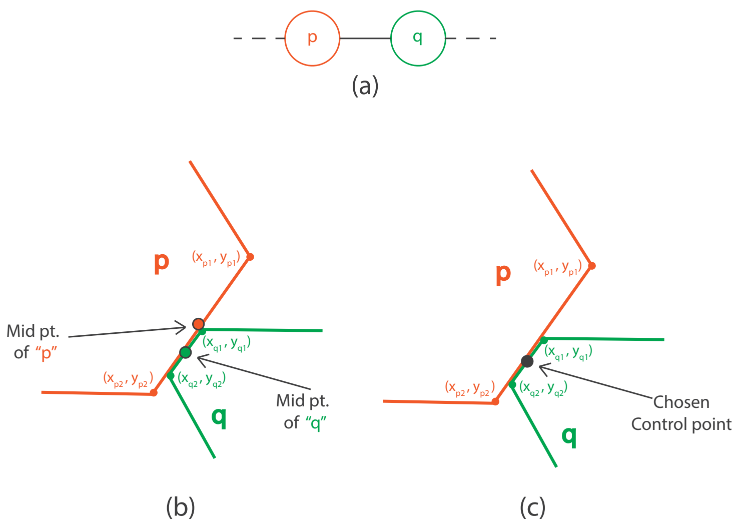


Figure 5.2: (a) Current Node 'p' and Next Node 'q' (b) Mid-point Selection (c) Chosen Control Point

The calculation of the centroid, $C_{centroid}$ is rather straightforward for a convex polygon over its n vertices and is given by the equation 5.3:

$$C_{centroid} = \frac{1}{3}(\frac{\sum_{j=1}^{n}(x_j + x_{j+1})(x_j y_{j+1} - x_{j+1} y_j)}{\sum_{j=1}^{n}(x_j y_{j+1} - x_{j+1} y_j)}, \frac{\sum_{j=1}^{n}(y_j + y_{j+1})(x_j y_{j+1} - x_{j+1} y_j)}{\sum_{j=1}^{n}(x_j y_{j+1} - x_{j+1} y_j)}) \tag{5.3}$$

The purpose of taking centroids as a control point is to introduce an extra point of freedom in terms of trajectory selection as planning just on the basis of transition points may sometimes lead to trajectories with extremely sharp turns which might be significantly hard to achieve in a practical scenario.

After the computation of control points, the whole path can be described as a series of control points which are actual physical coordinates in a given map with starting and destination coordinates at the ends, respectively as shown in Figure 5.3. S and G represents the starting and destination coordinates, $C_{i,c}$ represents the centroid of ith cluster while $C_{i,j,t}$ represents the transition control point from ith cluster to jth cluster.

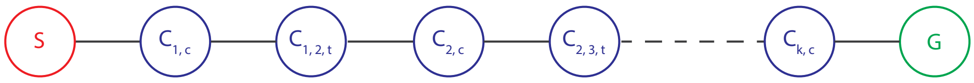


Figure 5.3: Serial Representation of Control Points

Now that we have obtained the control points, we can use standard parametric curves such as Bézier Curves [19] [20], B-splines, etc in obtaining the smooth curve equations for a desired trajectory.

Chapter 6

Experimentation and Conclusion

6.1 Experimentation

We wrote the complete exploration algorithm with all its constituents in Python 3.6 and ran it for multiple instances to check how it performed. The agents we run for multiple iterations each iteration consisting of taking one step forward or backtrack and these were the results observed with frontier detection running after every motion of either robot.

6.2 Results

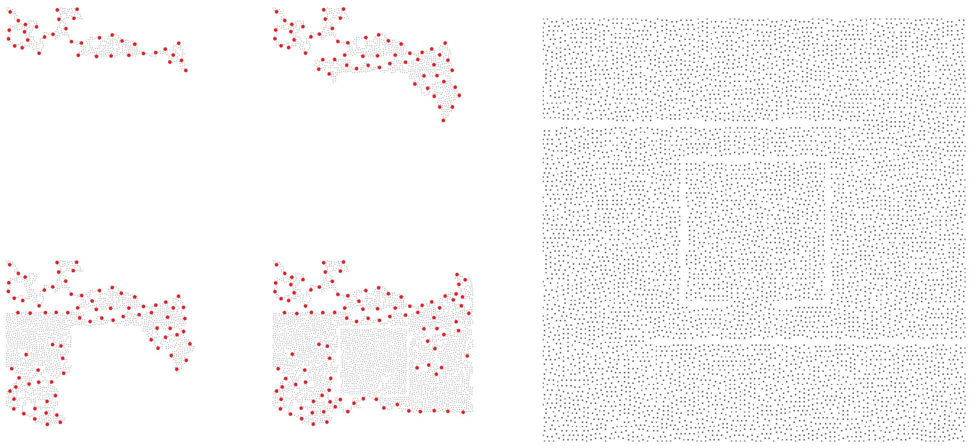
We tested the algorithm on different maps using two robot agents and a master pc system for soft agents.

We show our results on the first map in Figure 6.1. The red points marked in Figure 6.1a shows the detected frontier points and how they are being removed from the map as the search progresses.Some things which can inferred from this results is:

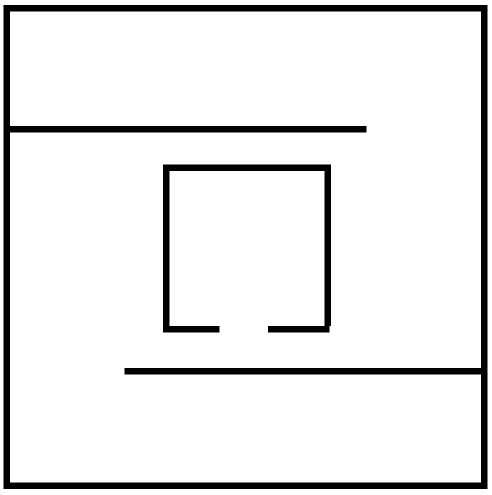
- We also saw considerably fast search as compared to our previous search results stated in our last report.
- The agent was able to search and map complex areas with prime efficiency and no bottle-

necks or slowdown.

- Detected frontier points were only explored in the need of backtracking and not otherwise.
- We see an search optimization in the 4th state of Figure 6.1a where the two regions namely inside the box and the region to the right of it are explored by the agents separately as a result of optimal allocation.



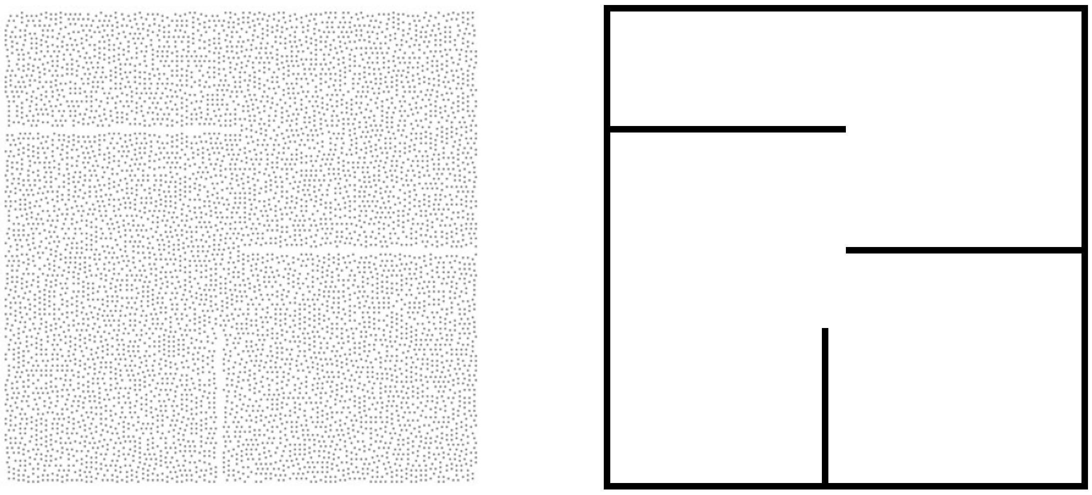
(a) Map built by the robot with frontier points



(b) Actual Map

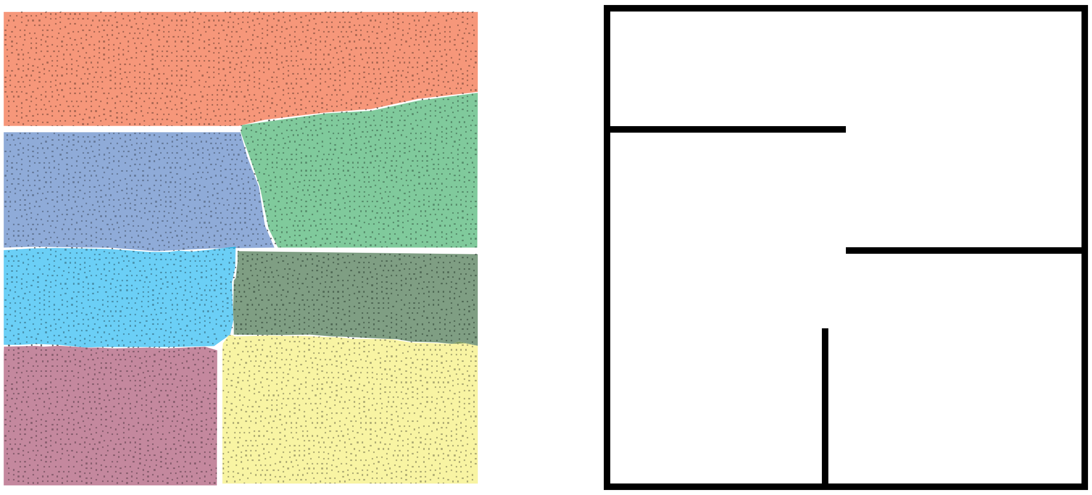
Figure 6.1: Results for First Map

The second map and its traversal can be seen in Figure 6.2, We will be using this map to test our clustering and path planning algorithms as shown below.



(a) Final Map as stored by robot (b) Map

Figure 6.2: Results for Second Map



(a) Clustered Map (b) Actual Map

Figure 6.3: Result from Clustering Linear Travel Regions

Here in Figure 6.3a, we see the clustered output for traversal of the map beside it. We can note some shortcomings like the area marked with light blue and dark green can also be grouped. But this will in no way hamper any travel and neither give suboptimal paths. We can also see that all clusters formed to build a convex polygon that means each point internal to the polygon lies can be reached from any other point using a straight line which is what we wanted to achieve.

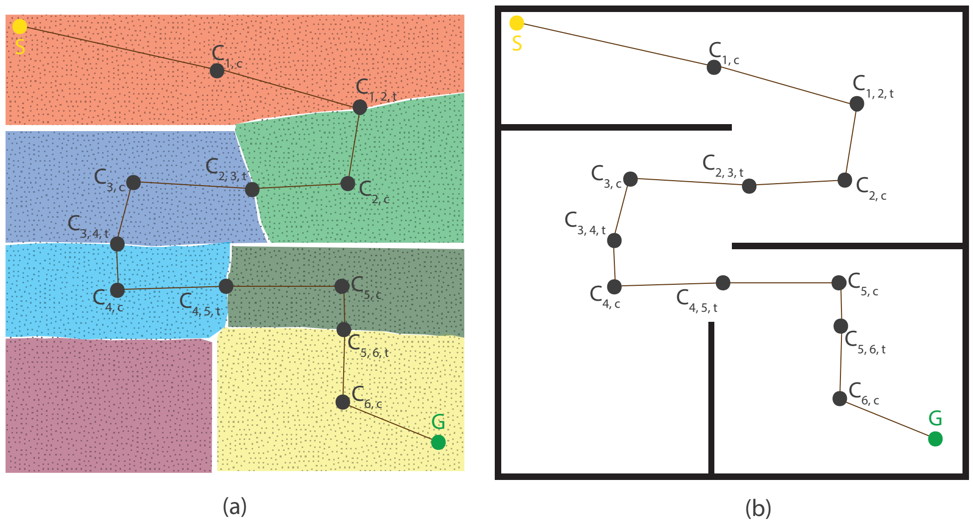


Figure 6.4: Visual Representation of Path Selection by Control Points

Figure 6.4 represents the visual representation of the path selected using the control points described in Section 5.3. S and G represent the starting and destination coordinates, $C_{i,c}$ represents the centroid of the ith cluster while $C_{i,j,t}$ represents the transition control point from ith cluster to jth cluster.

Note the path obtained through such method is not optimal. Instead, it can be considered as the safest because of its nature to stay far away from the obstacles as much as possible because the path is deliberately made to go through the centroids of each visited cluster.

6.3 Conclusion and Future Work

A modified map exploration strategy was proposed to enhance already existing methods. SRT was used for moving the agent while frontier detection was done using RRT. The usual ways of exploration are systematic methods which take a definite mathematical path like the linear motion of spiral motion for exploration. Such practices tend to fail under regions with high obstacle density. After performing multiple experiments using this technique, the following conclusions were drawn about the method and how it fared with respect to other technologies present out there.

- The agent was able to traverse and completely scan any scenario as compared to systematic linear or spiral methods who can get stuck in different regions.

- As the agents diverged in the map they started mapping the area faster as compared to before which establishes the use of swarm.
- Random Walk is actually complete algorithm and can be used to move around a map and eventually reach your desired destination at some point in time.

Further, after the map of unexplored area was created, a novel idea to decompose the map into smaller convex polygons (clusters) for path planning was proposed. Following are the advantages of the method proposed:

- The decomposition of the map by clustering of traversed points into convex polygons significantly reduced the memory consumption as only vertices of the polygons were required to be kept into memory, compared to the formal SRT case in which all the coordinates of the traversed points need to be maintained.
- Secondly, the property of convex polygons that all points inside it can be connected using a straight line became a crucial property in path planning along with a representation of the map in the form of a graph where each node represented a unique cluster made the computational expense to reduce significantly.
- Since map exploration and trajectory planning are two widely researched independent fields, and this algorithm tries to connect the two using the method proposed. Usually, in trajectory planning algorithms, it is assumed that control points are known, and optimization is done for computing better trajectories, the proposed method instead tries to find the control points thereby allowing it to be combined with already existing state-of-the-art methods.

There can be further modifications to the algorithm to make it more optimal for multi-agent systems so that the agents can diverge and explore exclusive regions in the environment. Also the frontier detection techniques and motions techniques can be expanded for 3D motion for UAVs directly without much modification.

The method of path planning proposed can be extended to multi-agent systems with some modifications to avoid collision among different agents at the intersection of the planned trajectories.

The clustering algorithm can also be further improved to connect multiple clusters to perform a more optimized path planning between regions.

Finally, We would like to conclude by saying that while this novel technique of implementing and expanding single robot search to multi-robot search worked efficiently to achieve its required task without any failure. There were some bottlenecks and slowdowns but the the algorithms implemented were fast enough to recover from them without any loss of information. We went forward to also optimize our output so that it can later be used by anyone else without knowing any background of how the map was built thus preparing it for general use.

Bibliography

[1] S. M. Lavalle, “Rapidly-exploring random trees: A new tool for path planning,” Tech. Rep., 1998

[2] Umari, Hassan, and Shayok Mukhopadhyay. ”Autonomous robotic exploration based on multiple rapidly-exploring randomized trees.” In 2017 *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1396-1402. IEEE, 2017.

[3] G. Oriolo, M. Vendittelli, L. Freda and G. Troso, ”The SRT method: randomized strategies for exploration,” *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA ’04.* 2004, New Orleans, LA, USA, 2004, pp. 4688-4694 Vol.5.

[4] B. Yamauchi, “Frontier-based exploration using multiple robots,” in *Proceedings of the Second International Conference on Autonomous Agents (AGENTS ’98).* New York, NY, USA: ACM, 1998, pp. 47–53.

[5] H. El-Hussieny, S. F. M. Assal and M. Abdellatif, ”Improved Backtracking Algorithm for Efficient Sensor-Based Random Tree Exploration,” *2013 Fifth International Conference on Computational Intelligence, Communication Systems and Networks*, Madrid, 2013, pp. 19-24.

[6] Franchi, Antonio & Freda, Luigi & Oriolo, Giuseppe & Vendittelli, Marilena. (2009). ”The Sensor-based Random Graph Method for Cooperative Robot Exploration.” *Mechatronics, IEEE/ASME Transactions on.* 14. 163 - 175.10.1109/TMECH.2009.2013617.

[7] Zou Yiping, Guo Jian, Zhang Ruilei, Chen Qingwei, ”A SRT-Based Path Planning Algorithm in Unknown Complex Environment”, *2014 26th Chinese Control and Decision Conference (CCDC)*

[8] L. Freda and G. Oriolo, ”Frontier-Based Probabilistic Strategies for Sensor-Based Exploration,” *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, Barcelona, Spain, 2005, pp. 3881-3887.

[9] Hassan Abdul-Rahman Umari, ”Multi-robot Map Exploration based on multiple Rapidly Exploring Random Trees”, *American University of Sharjah College of Engineering*, May 2017

[10] Elshenawy Elsefy, Ayman & Mohamed, Khalil & Harb, Hany. (2018). ”Exploration Strategies of Coordinated Multi-Robot System: A Comparative Study.” 48-58. 10.11591/ijra.v7i1.pp.48-58.

[11] Qiao, Wenchuan, Zheng Fang, and Bailu Si. ”A sampling-based multi-tree fusion algorithm for frontier detection.” *International Journal of Advanced Robotic Systems 16*, no. 4 (2019): 1729881419865427.

[12] Adler, Benjamin, Junhao Xiao, and Jianwei Zhang. ”Autonomous exploration of urban environments using unmanned aerial vehicles.” *Journal of Field Robotics 31*, no. 6 (2014): 912-939.

[13] Zhu, Cheng, Rong Ding, Mengxiang Lin, and Yuanyuan Wu. ”A 3d frontier-based exploration tool for mavs.” In 2015 *IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 348-352. IEEE, 2015.

[14] Kim, Jinho, Stephanie Bonadies, Andrew Lee, and S. Andrew Gadsden. ”A cooperative exploration strategy with efficient backtracking for mobile robots.” In 2017 *IEEE International Symposium on Robotics and Intelligent Sensors (IRIS)*, pp. 104-110. IEEE, 2017.

[15] M. Bernard, K. Kondak, I. Maza, and A. Ollero, Autonomous transportation and deployment with aerial robots for search and rescue missions,” *J. Field Robot.*,vol. 28, no. 6, pp. 914-931, 2011.

[16] Papachristos, Christos, Shehryar Khattak, and Kostas Alexis. ”Uncertainty-aware receding horizon exploration and mapping using aerial robots.” In 2017 *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4568-4575. IEEE, 2017.

[17] Faigl, Jan, and Miroslav Kulich. ”On determination of goal candidates in frontier-based multi-robot exploration.” In 2013 *European Conference on Mobile Robots*, pp. 210-215. IEEE, 2013.

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

[19] J. Choi, R. Curry and G. Elkaim, ”Path Planning Based on Bézier Curve for Autonomous Ground Vehicles,” *Advances in Electrical and Electronics Engineering - IAENG Special Edition of the World Congress on Engineering and Computer Science 2008*, San Francisco, CA, 2008, pp. 158-166.

[20] Y. Ho and J. Liu, ”Collision-free curvature-bounded smooth path planning using composite Bezier curve based on Voronoi diagram,” *2009 IEEE International Symposium on Computational Intelligence in Robotics and Automation - (CIRA)*, Daejeon, 2009, pp. 463-468.

[21] Ferguson, Dave, Maxim Likhachev, and Anthony Stentz. ”A guide to heuristic-based path planning.” *In Proceedings of the international workshop on planning under uncertainty for autonomous systems, international conference on automated planning and scheduling (ICAPS)*, pp. 9-18. 2005.

[22] Yao-hong Qu, Quan Pan and Jian-guo Yan, ”Flight path planning of UAV based on heuristically search and genetic algorithms,” *31st Annual Conference of IEEE Industrial Electronics Society, 2005*. IECON 2005., Raleigh, NC, 2005, pp. 5 pp.-.