DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS

STOCKHOLM, SWEDEN 2019

Heterogeneous 3D Exploration with UAVs

PEDRO BELO

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

KTH Computer Science
and Communication

Heterogeneous 3D Exploration with UAVs

PEDRO BELO

Master's Thesis at CAS
Supervisor: Fernando dos Santos Barbosa
Examiner: Patric Jensfelt

Abstract

Multi-robot exploration algorithms usually focus on exploration time minimization while ignoring map accuracy. In this thesis, it is presented a new heterogeneous multi-robot exploration strategy that finds a balance between time consumption and map accuracy. By ranking UAVs based on their sensor accuracy, it is possible to coordinate them and pick strategic points to explore rather than the most rewarding ones. In particular, with a function (in this case a Gaussian) that maps a voxel's uncertainty to a score, it is possible to tailor a UAV's preference (by tuning expected value and variance) towards certain features and not only unexplored space. This algorithm was compared with AEPlanner for several environments, achieving better accuracy towards complete map exploration.

Referat

Multi-robot-utforskningsalgoritmer fokuserar vanligtvis på tidsminimering av utforskningen medan man ignorerar kartnoggrannheten. I denna avhandling presenteras en ny heterogen multirobot-utforskningsstrategi som hittar en balans mellan tidsförbrukningen och kartnoggrannheten. Genom att rangordna UAVer baserat på sensorns noggrannhet är det möjligt att samordna dem och välja strategiska punkter för att utforska istället för de mest givande. I synnerhet med en funktion, (i detta fall en gaussisk) som kartlägger en voxels osäkerhet till en poäng, är det möjligt att skräddarsy en UAVs preferens (genom att ställa in förväntat värde och varians) till vissa funktioner och inte bara outforskat utrymme. Denna algoritm jämfördes med AEPlanner för flera miljöer, vilket uppnådde bättre noggrannhet mot fullständig kartutforskning.

Contents

Chapter 1

Introduction

The problem of exploring and mapping an unknown environment is a crucial topic in mobile robotics. Although single-robot exploration has been widely studied, multi-robot exploration is still a new area that looks very promising.

Multi-robot cooperation has been proven to be advantageous in several tasks. For instance, when having multiple interconnected robots, it is possible to improve the localization of each one of them [1]. Also, multiple robots tend to be more robust when performing tasks since, if one malfunctions, the others might still accomplish the task. Finally, the amount of information collected is usually higher compared to single-robot systems. This introduces redundancies that increase robustness. For example, when mapping, the more scans of the environment we get, the higher the quality of the map. Sometimes it is even advantageous to have several imprecise robots over a single very precise one.

In this paper, a new approach to multi-robot exploration and mapping is looked at. Most papers that deal with this problem, try to map an environment as fast as possible without focusing on the quality of the map. In this thesis, this issue is addressed.

The objective of this work is to go beyond one-time exploration, meaning that the UAVs should keep exploring until a satisfactory precision is achieved, even if that means re-scanning certain areas of the map.

Each robot's sensing capabilities can be defined using two variables, the probability of sensor hitting an object and the probability of missing it. With these variables, it is possible to rank UAVs and prioritize tasks. Intuitively, a UAV with low sensing capabilities should do a quick scan of the whole environment while the UAV with high sensing capabilities should focus on obstacles to obtain better precision. This is exactly what is done in this thesis although the tasks attributed are flexible, meaning that each UAV has a preference for certain tasks but is not restricted to them.

Although this work was developed and tested for 2 UAVs (one equipped with an RGB-D camera and the other with a Lidar), since the tasks are flexible, the algorithm proposed can easily be generalized to any number of UAVs with any type

of sensors.

All the work in this thesis is released as open-source[1] to allow future research.

1.1 Problem Description

The problem can be posed as follows: In a 3D bounded environment, having a UAV equipped with an RGB-D camera and another with a Lidar, how can we obtain an accurate map while exploring the environment as fast as possible?

The objective of this thesis is to take advantage of the characteristics of the sensors and make a more efficient, coordinated exploration system. More precisely, the resulting map of the exploration should be as accurate as if there was only the UAV with the RGB-D camera and also, it should be faster. Since the information from the Lidar is often imprecise (the measurements of the Lidar can be more precise than the camera but the range of the Lidar is greater, making it more sensitive to position error and thus less reliable), this does not offer a high-quality map but still, this information should not be discarded and should be used, at least, to guide the RGB-D camera-equipped UAV.

It is assumed that the exploration area is mainly empty compared to the size of the robot, which is not a very restrictive assumption since if the robot can move around then there is plenty of free space.

1.2 Sustainability, Ethics and Social Relevance

UAVs are ever more present in our society and their uses are endless. From 3D aerial imaging to intelligent farming, UAVs offer a solution to a wide range of needs. Incorporated with GPS, UAVs are very precise, which makes them perfect for mapping or security and surveillance. Also, they are easily deployable and require little knowledge to operate them, which makes it ideal for any type of consumer, associated with its low price.

Due to all these reasons, UAVs are becoming very popular but because it is such a new technology, there are also some issues associated with them. Security is one of them. Because UAVs are so easily accessible, misuse of them poses a direct risk to anyone around them. Also, the use of UAVs in "no-fly zones" is somewhat common and can pose a risk to airplanes for instance. Another issue is privacy since UAVs are usually equipped with cameras.

Despite these cons, UAVs are a technology that has come to stay and so they should be used responsibly and with appropriate equipment.

1.3 Outline

This work is divided into 4 main chapters.

[1] https://github.com/pedrobelo/Heterogeneous-Exploration

First, we have the **Related Work** chapter that goes through some exploration strategies for single and multi-robot systems. For single robot systems, there are several ways of guiding a robot during exploration. Usually, robots go towards the most rewarding point in the map but how this reward is calculated quite differs between algorithms.

In the **Proposed Approach** chapter, we start by explaining how the exploration algorithm works (Autonomous Exploration Planner) for single UAV exploration, focusing on the evaluation function that was changed to adapt it to multi-robot systems and solve the research question. Finally, it is described how collision avoidance was addressed.

In the chapter **Simulation and Results**, the simulation environment is described and the results comparing several algorithms for several environments are shown.

Finally, in the **Conclusion and Future Work** chapter, some improvements are suggested for future research.

Chapter 2

Related Work

The main objective of this thesis is to develop a faster mapping strategy by taking advantage of the heterogeneity of the UAVs while keeping map accuracy. In this section, several exploration strategies are explained.

Several works address the multi-robot exploration problem like [2] or [3]. Although there is a big amount of content related to this area, only a few of these papers address the problem of heterogeneous exploration.

All approaches that consider heterogeneous robots try to maximize exploration speed, not focusing on the quality of the map. In a way, these approaches can be thought of as a generalization of homogeneous exploration.

2.1 System Structure

Before delving into exploration algorithms, it is important to talk about multi-agent system structures. There are two main system structures, centralized and decentralized.

2.1.1 Centralized

A centralized system is a system where every piece of information is centered around a single agent. In the context of multi-robot exploration, it is usually used a computer that coordinates every robot like in [4] and merges the maps produced by every robot. We can see that good communication and system robustness are crucial for proper exploration. If communication is somehow cut off or if the central system malfunctions, the whole system is badly affected, if it doesn't stop working altogether.

A centralized approach has the advantage that every robot always has access to the latest version of the global map. Also, maximum cooperation for better efficiency is accomplished when exploring an unknown environment.

The centralized approach, however, has some disadvantages like scalability. When increasing the number of agents, a centralized approach might not be vi-

able due to the lack of computing power of the central agent. Also, it is not always possible to keep every agent connected for several reasons like signal loss or some malfunction.

2.1.2 Decentralized

Decentralized approaches are more common due to their robustness. They are not as efficient but are usually preferred. This is a topic that has been addressed in some papers like [2].

Decentralized approaches, although not as efficient as centralized ones, can withstand communication interruption. Also, if there is no interruption, decentralized approaches can be as optimal as centralized ones.

In [5], Yuan et al. address this optimality problem by constraining robot movement to communication range. This way, the robots stay completely coordinated while distributing tasks.

2.2 Map Representation

As will be seen, most exploration strategies that will be described in the next section, are based on a type of 3D map called Octomap but before explaining what an octomap is, it is important to first understand an octree.

2.2.1 Octree

When representing a 3D volume of space as an octree, space is divided into equally sized cubes (also called cells or voxels) and the cubes are organized in a tree structure as represented in figure 2.1. This type of representation was first proposed in [6] by Meagher, where each cell could have one of three values: "Empty", "Partial" or "Full".

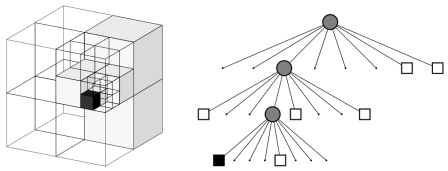


Figure 2.1. Octree representation. Extracted from [7]

2.2.2 Octomap

Building on [6], Hornung et al. proposed an improved representation of octrees in [7] called octomaps. Instead of having only three states for each voxel, they are now defined by a probability of being occupied (commonly called occupancy probability). Also, in order to improve the efficiency, the algorithm also prunes

the tree, by removing octets of leaves with similar occupancy probabilities. This can be thought of as grouping eight adjoining cells with similar probabilities and representing them as one, as can be seen in figure 2.2.

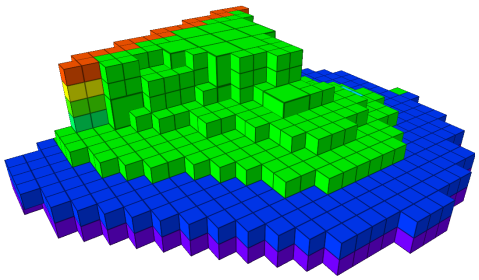


Figure 2.2. Octomap representation. Source:
https://answers.ros.org/question/287390/integrate-free-space-into-octomap/

2.3 Exploration Strategies

Now that system structures have been explained, we can look into exploration strategies. Several exploration strategies are presented in this section, both for single robot systems as well as multi-robot systems. The exploration strategies differ very much from one another but they all try to maximize the information gained by assigning each robot to a certain destination. This is score maximization although the way that the score is calculated differs for each approach.

Before diving into the exploration algorithms, three terms should be explained, *score*, *gain* and *cost*. The *gain* of a certain position is the information the robot gets when it reaches that position, it is usually associated with the volume of unexplored space around it. *Cost* is how much effort it is required of the robot to get to that position. The *cost* might be the distance that needs to be traveled or the acceleration required. Finally, the *score* is a mixture of both *cost* and *gain*. There are several ways to compute the *score*, one would be $score = gain - cost$.

2.3.1 Uncertainty

In [8], the authors take a look at the probabilistic nature of robotics.

By having different robots with different sensor capabilities, the uncertainty that the robot will be in a certain state after applying some control u will differ. With this in mind, the authors developed a way of computing traveling *cost*.

Imagine we have two UAVs, one with very precise measuring capabilities and low control over the state. The other, very precise when moving but not so much when taking measurements. With this system, it might not be worth to move the first robot to some distant area since it is quite unlikely that the robot will get there. Instead, it is more profitable to move the low sensing capabilities robot since

we are almost sure that it will get there (although the readings might not be the best, it is better than nothing).

To sum up, [8] introduces probabilistic reasoning. The algorithm computes the distance to the objective and then it multiplies it by the probability of the robot getting there, obtaining the *cost* of the path.

A probabilistic approach is also used in [9]. In this paper, Burgard et al. use *score* to coordinate the robots, where both the *cost* and *gain* consider probabilities. First, the *cost* is computed as the distance from the robot to the frontier assigned(the *cost* of each cell is proportional to its occupancy probability). Second, to compute the *gain*, the authors calculate it based on the probability that the robot will observe those cells when it gets there if it gets there.

In [10] Burgard et al. also do something similar as in [9] but they do not consider the probabilistic nature of robot movement since it is not that relevant and speeds up algorithms considerably. Because of this, they also make some adjustments to avoid the robot collision due to their uncertainty in the position (they introduce a *cost* of traveling close to objects, to push the robots away).

2.3.2 Market Economy

Market economy is a strategy used to maximize *score* among all robots while not requiring a centralized system. This strategy is versatile in a way that allows robots to get disconnected or malfunction without affecting the rest of the system. If a robot gets disconnected from the other, the overall system might not be as optimal but it works nonetheless.

Market economy strategies for multi-robot task allocation were first introduced in [11]. By using a free market where different robots can sell and/or buy services, it is possible to obtain a higher net *score*. This type of approach is very robust since each robot interacts independently, meaning that there is no central agent. Market-based approaches are also fault-tolerant, a robot can lose communication and/or malfunction but the whole system can still manage to accomplish its goals. This paper, however, does not directly address the exploration problem.

In [2], the first market-based approach to exploration was implemented. The way that this algorithm works is, first, each robot generates a set of promising exploration points (there are several ways to sample these points, randomness being one of them). After that, they make these goal points available for auction (every robot can buy every goal point). Now, the robot selling the goal point will be called the auctioneer and the one buying will be called bidder.

Imagine the auctioneer has a goal point *gp* available with an associated *score* s_a. If the bidder has a higher *score* s_b for that same point ($s_a < s_b$), the bidder will submit a bid and try to buy the goal point *gp*. In the case that there are more than two robots, then there will be multiple bidders and the goal point *gp* will be sold to the highest bidder as long as the highest bid satisfies the condition $s_a < s_b$. If not, the goal point is not sold. This is represented by equation 2.1.

$$\begin{cases} \text{if } \max_i\{s_{b_i}\} > s_a, \text{ sell } gp \text{ to } \arg\max_i\{s_{b_i}\} \\ \text{if } \max_i\{s_{b_i}\} < s_a, \text{ keep } gp \end{cases} \tag{2.1}$$

This approach, however, is not always optimal. Selling goal points to the highest bidder might not result in the highest overall *score* (sum of the *scores* of all the robots). Another case that might happen is that a single robot buys two-goal points that are close to the robot but in opposite directions, meaning that it is worth buying one but not the other. This, however, is not addressed in [2] since it seldom occurs.

In [4], the authors developed a centralized system. The executive (central system) keeps an updated global map and all the bids. Every time a robot receives an update of the map from the executive, it comes up with a new bid and sends it to the executive. After this, the executive assigns the highest bids to each robot. It starts with the highest bid and assigns it. Then goes to the second-highest and so on. It should be noted that this approach does not always obtain the highest overall utility as well. Although this might be true, in practice, this greedy solution works very well. Also, when a bid is assigned, the *score* of the area around this bid is reduced. This avoids the problem of several robots being assigned very close bids (or destinations).

In [12] it is also used a market-based approach to minimize the time of exploration. In this paper, the authors spread the bidding communication through several time steps (each time step corresponds to a small interval of time where the robots' state changes). By spreading communication, the authors claim they can achieve better efficiency since each robot does not wait a significant amount of time for other bids. This is possible because they were able to reduce bidding messages required for the correct functioning of market-based approaches, compared to other similar algorithms.

2.3.3 Potential Field

Potential fields are another approach to multi-robot exploration. By representing the occupancy grid as a potential field, each robot moves towards the stronger attraction.

This method was introduced in [13] where Haye Lau defined obstacles and other robots as repulsive and frontiers as attractive. By summing all these forces for each grid cell, the robot is guided towards frontiers while also avoiding obstacles and collisions with other robots. Each obstacle, robot or frontier, generates a potential field around it defined by equation 2.2. In 2.2, *gain* refers to how attractive/repulsive a certain element is and d is the distance from the element to a certain grid cell. This formula is computed for every grid cell around every obstacle, robot, and frontier.

$$F = \begin{cases} {}^{gain}/_{d^2}, & \text{if } d \leq d_{\text{threshold}} \\ 0, & \text{if } d > d_{\text{threshold}} \end{cases} \tag{2.2}$$

It is easy to see that, although potential fields are very simple, some problems arise that compromise the robustness of the algorithm. First, local minimums might appear, meaning that the robot is situated somewhere where the forces cancel each other out and so the robot stays still (this problem is addressed in [13] by using a behavioral approach that will be described later. Simply put, every time the robot finds itself in a local minimum, it gets out by traveling to some frontier, ignoring potential forces).

Another problem with this algorithm is that it is hard to tune since the $Gain$ and the $d_{threshold}$ should be adjusted for each element, which might not be easy and it might not work good enough in every environment.

Although this might be true, this algorithm still proves itself very useful for collision avoidance.

2.3.4 Behavioral

Behavioral architectures are yet another option to solve the multi-robot exploration. The basic idea behind behavioral architectures is very simple, each robot behaves differently (runs on a different algorithm) depending on its conditions. For instance, if we have a single robot that wants to explore a room, the first behavior would be to explore the whole room and the second behavior would be to return to the docking station after having explored everything. Each behavior has a trigger.

In [13], as mentioned before, Haye Lau uses a behavioral algorithm to escape local minimums in the potential field. Every time the robot senses that it is stuck (trigger), it changes behavior to a path planning algorithm so it can escape the minimum. Once it has escaped, it can go back to explore using the potential field.

A more complex behavioral architecture is used in [14]. Here, Jose Vazquez and Chris Malcolm force a network of multiple robots to stay connected (each robot has contact with every other, directly or indirectly). To do this, the authors use a behavioral approach. Simply put, each robot explores independently but every time it senses that it is getting disconnected from the rest of the graph, then it goes back (new behavior) so it stays connected. After having a strong enough connection, it goes back to exploring.

We can see the behaviors work similarly to switches. In a way, it is an easy solution to exploration but on the other hand, it might not be robust enough since the triggers for the behaviors might not always be easy to define and might also prevent us from achieving optimality.

2.3.5 Graph-based

Graph-based strategies are another form of exploration that uses an alternative to occupancy grids. These graphs have the advantage of having a very light representation although they might not contain as much information as an occupancy-grid.

In [15], Franchi et al. make use of this to develop an exploration strategy. More specifically, they develop a Sensor-based Random Tree (SRT). This tree's nodes are

positions where at least one robot has been at and its edges correspond to collision-free paths between nodes. The way their algorithm works is, each robot travels in the direction of a frontier and every time a robot reaches a new position, it adds it to the graph as a node. Each robot does this independently, meaning that each robot has its own SRT although they may help each other building each other's SRT and avoiding collisions, allowing cooperative exploration.

The SRTs are then generalized to SRGs (Sensor-based Random Graphs) in [16]. Here, the logic is the same but for connectivity purposes, some extra edges are added (edges between nodes that have visibility over each other although they might not have been traveled by a robot).

Also, in [16], every robot acts independently, seeking its own reward (exploration of unknown cells). They do this for short distances, making collision avoidance easier (when two robots establish communication, then they are close and necessary measures are taken). No planning is necessary. Although this approach is fairly simple and computationally light, it might not be optimal.

2.3.6 Information-based

Another approach to quantify the *gain* of a certain cell is to use entropy.

In [3], the authors use this approach to quantify how much information is gained by exploring a certain cell. Intuitively, a cell is worth more if we are very uncertain about its nature (occupied or unoccupied). This means that the robot will tend to go to unexplored areas.

Formalizing the intuition, we have a grid map composed of cells with a value between 0 and 1. 0 represents an unoccupied cell and 1 represents occupied. With this in mind, we can now formulate the information gained around the robot.

$$information = \sum_{x} occ[x] \log_2 \frac{1}{occ[x]} + (1 - occ[x]) \log_2 \frac{1}{(1 - occ[x])} \tag{2.3}$$

Where x represents all the cells in range of the sensors of the robot that might be explored and $occ[x]$ represents the probability of cell x being occupied.

With this formula, we can conclude that cells with a probability of 0 or 1 yield no information. On the contrary, a cell with a probability of being occupied of 0.5 yields the most information. This function is represented in figure 2.3.

2.3.7 Genetic Algorithms

Yet another approach to multi-robot exploration is genetic algorithms. These types of algorithms can be viewed as an oriented random search of overall maximum *score*.

This type of algorithm is presented in [17]. The authors claim the algorithm is very efficient and obtains good results compared to other exploration strategies.

The algorithm presented in [17] can be summarized as:

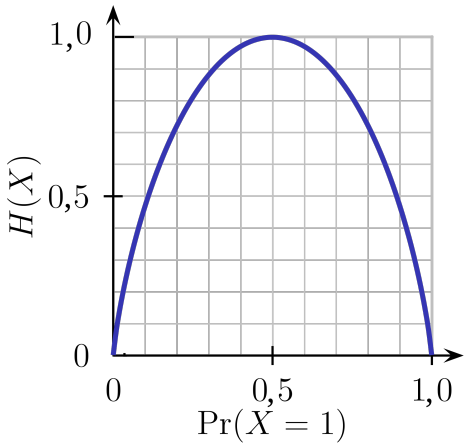


Figure 2.3. Information *gain* of a grid cell. $H(X)$ represents the information gain and $Pr(X = 1)$ represents the probability of cell X being occupied. Source: https://www.kisspng.com/png-binary-entropy-function-information-theory-plot-in-1563751/download-png.html

1. **Generate a certain amount of chromosomes.** In this paper, the chromosomes are numbers that assign each robot to a frontier. Imagine having a vector with the same size as the number of frontiers. For each element (frontier), we have either a robot assigned or no robot. A chromosome would be something similar to figure 2.4 where F is short for frontier and R for robot.

2. **Sample the chromosomes.** Here, a second population is sampled from the initial one. In [17], the authors use systematic resampling. The weight of each chromosome is measured by its fitness (which is equal to the *score*).

3. **Crossover.** After resampling, crossover is used to generate the offspring. Simply put, every two chromosomes give origin to another two (offspring). This is done by merging the genetic information of both parents. More specifically, in this paper, the technique used is one-point crossover.

4. **Mutation.** Finally, the offspring is mutated by introducing small random changes in the chromosomes. This, although it might slow down convergence, it allows the algorithm to not get stuck in local minimums.

5. **Repeat 2-4 a certain amount of iterations.**

2.3.8 Voronoi diagrams

Voronoi diagrams are another way to guide each robot.

This solution was first introduced in [18]. In this paper, Haumann et al. introduce an algorithm called *DisCoverage*. The way it works is, first, space is divided

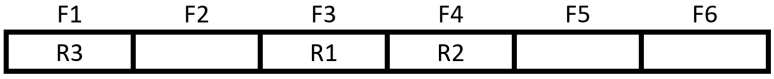


Figure 2.4. Chromosome representation in [17]

into Voronoi cells. After this partitioning, the robots try to maximize the coverage of unknown cells inside their Voronoi partition and move to that location. Then, a new Voronoi partition of the environment is generated and the process is repeated until all space has been covered. This can be summarized in figure 2.5.

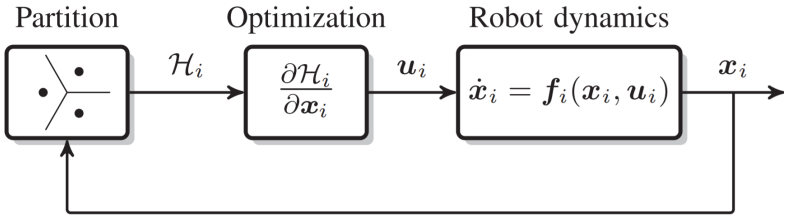


Figure 2.5. Voronoi-based solution to multi-robot exploration. First, space is partitioned into Voronoi cells. Then, information gained $\mathcal{H}$ is maximized through optimization. From this optimization, a control signal is obtained for each robot. Figure extracted from [18]

This type of algorithm has some advantages. First, since every robot stays inside their Voronoi cell, collisions are inherently avoided. Also, this algorithm automatically generates the next destination in a straight line for each robot, avoiding the need to use path planning strategies.

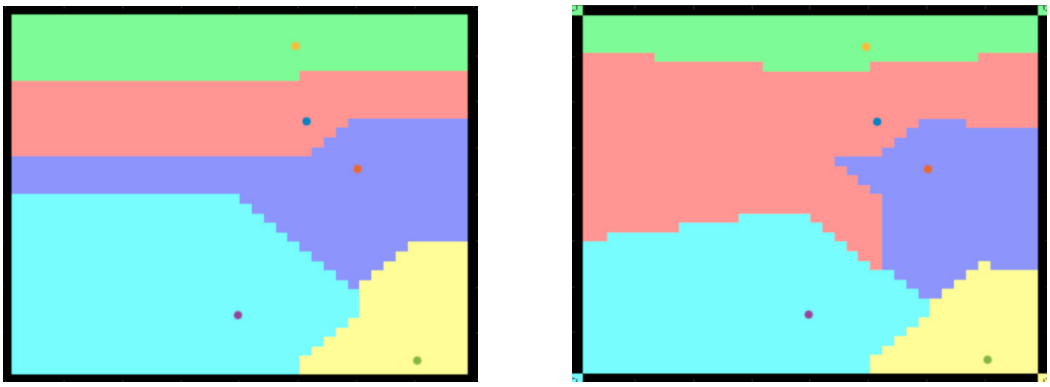
Building on this, [19] was published to address the heterogeneity problem. If two robots have different velocities, then the line separating both robots in the Voronoi diagram should not be equidistant to both robots. Instead, the line should be placed somewhere where both robots take the same time getting there like represented in figure 2.6b. This is exactly what Yanguas-Rojas and Mojica-Nava did.

2.4 RRT-based

The most optimal solution to exploration would be to compute the *score* for every cell in the map. This, however, can be very costly, which makes it impractical for online problems.

For online problems, less optimal approaches are usually preferred. They still achieve good results for a fraction of the computational cost.

One such approach is Rapidly-exploring Random Trees (RRT). This is a widely used sampling method that allows online exploration and was also used in this thesis. In this section, RRT and some of its variants are explained.



(a) Homogeneous Voronoi diagram (b) Heterogeneous Voronoi diagram

Figure 2.6.

We start with the original RRT. Then, RRT* is explained and how it tries to obtain optimal paths. Following these, RH-NBVP is introduced as an algorithm that makes use of RRTs to fully explore a bounded environment and FEP as an alternative exploration algorithm. Finally, the AEP merges these last two and STL-AEP makes some improvements on the last. Figure 2.7 summarizes all the dependencies of these algorithms.

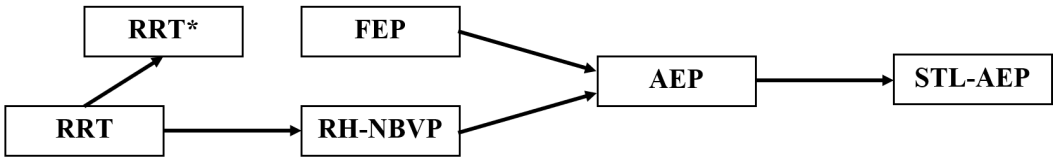


Figure 2.7. Dependency tree of RRT-based algorithms.

Now, RRTs are not the only path planning sample-based algorithms. Before RRTs there were others like randomized potential fields [20] and probabilistic roadmaps [21]. These algorithms, however, do not address the problem of non-holonomic systems (which is our case). This is mostly why RRTs are so widely used.

Also, before delving into each algorithm, it is important to clarify some terms: node, root and edge. Each tree can be represented as a sum of connected nodes without loops. In the context of exploration, a node is a representation of a specific position in space, and an edge is the connection between two nodes. Finally, the root is a special type of node and corresponds to the robot's current position.

2.4.1 RRT

Rapidly-exploring Random Trees were first introduced in [22]. The algorithm starts by sampling random points and connects them to the closest node of the existing tree, constraining the connections according to the robot's capabilities (movement restrictions for example). The algorithm is represented by algorithm 1.

Algorithm 1: GENERATE_RRT(x_{init}, K, Δt)

```
T.init(x_init);
for k ← 1 to K do
    x_rand ← RANDOM_STATE();
    x_near ← NEAREST_NEIGHBOR(x_rand, (T));
    u ← SELECT_INPUT(x_rans, x_near);
    x_new ← NEW_STATE(x_near, u, Δt);
    T.add_vertex(x_new);
    T.add_vertex(x_near, x_new, u);
end
return T
```

RRT starts by sampling a point x_{rand} from the state space. Then, it looks for the closest node x_{near} in the tree T. From x_{near}, x_{new} is generated by applying a set of controls u for a time interval Δt such that x_{new} is as close as possible to x_{rand}. Finally, x_{new} is added to the tree T.

Here, we can see how the RRT constrains sampled points to the reachable state space. This way, RRT can path plan for non-holonomic robots. For example, a tree generated for a non-holonomic robot might look something like 2.8b.

RRTs have several properties that make them very useful. For one, they are heavily biased towards unexplored space. An RRT picks points from a uniform distribution. This means that an RRT tends to cover all the reachable space in a uniform manner. Also, the paths generated tend to be close to optimal and finally, the RRT can be used in online systems, making it perfect for this thesis.

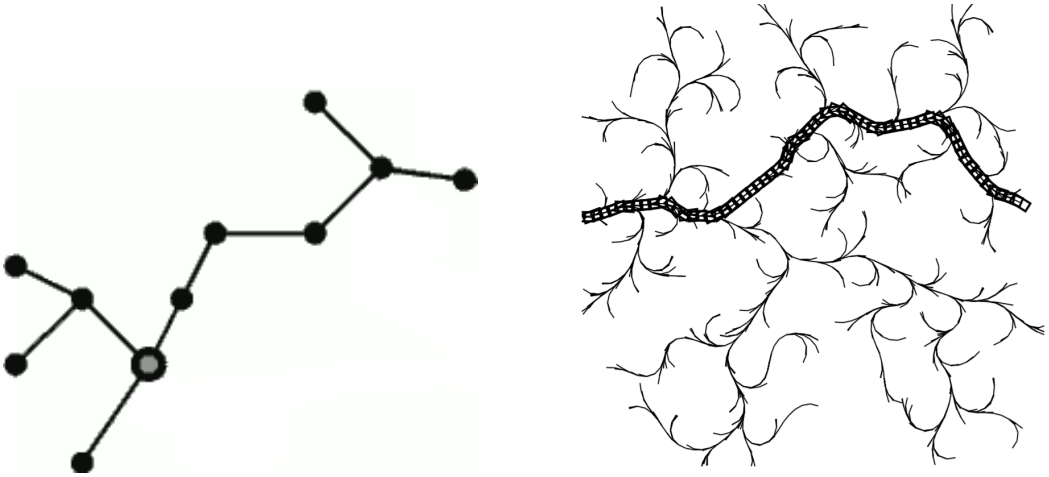
As we will see, the RRT was indeed used to coordinate the exploration of each UAV although it was used a variant of it, a more efficient one in this context. The algorithm used was the Autonomous Exploration Planning (AEP) that will be explained further below.

2.4.2 RRT*

When sampling points randomly, RRT connects it to the closest node in the tree and although this approach is guaranteed to find a solution (as the number of samples grows to infinity), it is not guaranteed to achieve an optimal one. RRT* was developed in [23] to answer this issue.

RRT* works very similarly to RRT. The only differences are the wiring and rewiring.

Wiring is the action of connecting a new sample to the tree. In RRT, the wiring is done to the closest node. In RRT*, it is done to the node that minimizes the



(a) Tree generated for a holonomic robot. Source: http://aigamedev.com/open/highlights/rapidly-exploring-random-trees/

(b) Tree generated for a non-holonomic robot. Taken from [22]

Figure 2.8.

distance to the root.

There is also rewiring. By rearranging connections, it is possible to achieve better paths. Whenever a new sample is added to the tree, some old edges might be replaced by more optimal ones.

This way, RRT* evolves asymptotically towards the optimal solution. The difference between the two algorithms can be illustrated in figure 2.9.

It should be noted that here, RRT* is described for only one robot but it can be generalized for any number of robots with the added factor that they have to avoid colliding with each other. This was done in [24], where each robot has a sector assigned to it where no other robot can enter.

2.4.3 RH-NBVP

Receding Horizon "Next-Best-View" Planner (RH-NBVP) was an algorithm introduced in [25] to guide robots towards complete 3D environment exploration. It uses the RRT to decide where to move next. The RRT allows the robot to pick the best locations to explore as well as how to get there without collisions.

The way it works is, an RRT is built throughout space where each node has an associated score. The robot then travels the first edge towards the highest score node and saves the rest of the branch. This process is then repeated but the new tree is now initialized with the previous best branch (this is known as "receding horizon" and makes the RRT faster). This can be illustrated in figure 2.10.

This score is calculated based on two factors, expected information gain and

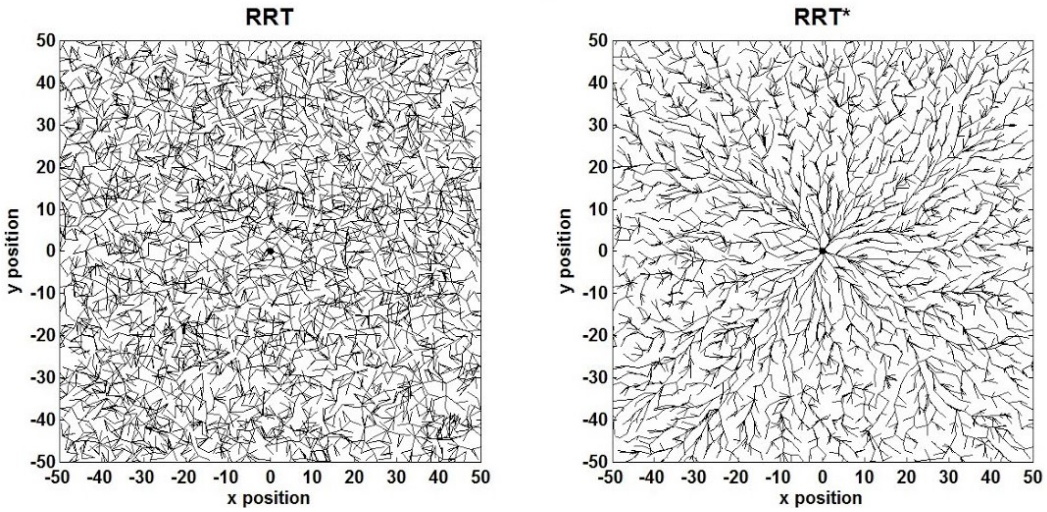


Figure 2.9. RRT represented on the right. RRT* represented on the left. Source: https://www.youtube.com/watch?v=JeEk_CWcRFI

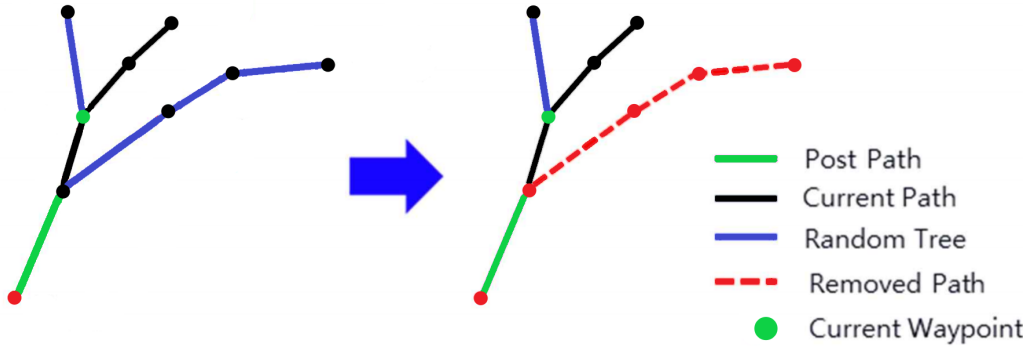


Figure 2.10. Illustration of Receding Horizon-based RRT. Extrated from [26].

distance. The more unexplored cells are around a node, the higher the score associated with it will be. On the other hand, the furthest a node is from the root, the lower its score will be. More formally, this can be calculated by equation 2.4 where $\boldsymbol{Score}(n_{k-1})$ is the score of the previous node, $\boldsymbol{Visible}$ is the number of visible unoccupied voxels from node n_k and $c(n_k)$ is the cost of traveling there. Finally, λ is a tuning constant.

$$\boldsymbol{Score}(n_k) = \boldsymbol{Score}(n_{k-1}) + \boldsymbol{Visible}(n_k)e^{-\lambda c(n_k)} \tag{2.4}$$

In the he RH-NBVP algorithm (algorithm 2), ξ is a robot's state, s corresponds to a score of a certain node and n represents a specific node. Also, N_T is the number of nodes in the tree and N_{TOL} is a tolerance number predefined. Finally, σ is the edge of the tree that is returned, which the UAV should follow.

Algorithm 2: Exploration Planner (Iterative Step)

$\xi_0 \leftarrow$ current vehicle configuration;
Intitialize T with ξ_0 and, unless first planner call, also previous best branch;
$s_{best} \leftarrow 0$; // Set best gain to zero
$n_{best} \leftarrow n_0(\xi_0)$; // Set best node to root
$N_T \leftarrow$ Number of initial nodes in T;
while $N_T < N_{max}$ ***or*** $s_{best} = 0$ **do**
    Incrementally build T by adding $n_{new}(\xi_{new})$;
    $N_T \leftarrow N_T + 1$;
    if ***Score***$(n_{new}) > g_{best}$ **then**
        $n_{best} \leftarrow n_{new}$;
        $s_{best} \leftarrow$ **Score**(n_{new});
    if $N_T > N_{TOL}$ **then**
        Terminate exploration;
$\sigma \leftarrow$ **ExtractBestPathSegment**(n_{best});
Delete T;
return σ;

2.4.4 FEP

Frontier exploration planning (FEP) strategies are very common for exploring unknown environments. Here, we give a closer look at how FEP works as how it was first introduced in [27] for single robots and [28] later for multiple robots. The algorithm developed by Yamauchi does not make use of RRTs but these are two algorithms that are often seen together and so it is described here under the RRT section.

The paper [27] presented a new way of exploring unknown environments without making any assumptions about the environment. The algorithm was initially designed for 2D environments but can easily be generalized.

Yamauchi used a probabilistic grid map to represent space much like an octomap for 3D environments. In this map, each cell is defined by a probability of being occupied.

1. $probability > 0.5 \leftarrow$ occupied cell
2. $probability = 0.5 \leftarrow$ unknown cell
3. $probability < 0.5 \leftarrow$ empty cell

With this representation, a frontier can be defined as a region in between free and unknown cells. These regions are the ones that the robot can access and explore.

A good way to completely explore an unknown environment would then be to travel towards all frontiers iteratively (once a robot gets to a frontier, that space is explored and new frontiers emerge). In [27], the robot picks the closest frontier to explore next and plans the route using depth-first search algorithm.

2.4.5 AEP

Autonomous Exploration Planning was an algorithm introduced by Selin et al. in [29]. It merges FEP and RH-NBVP for a more efficient and versatile algorithm.

RH-NBVP is very efficient for small environments but if they are big, this algorithm struggles to achieve complete exploration. Due to the exponential nature of the cost function, the score rapidly grows towards 0 as the distance increases. If the *score* is below a certain threshold, then the algorithm stops and exploration is considered complete, which is not necessarily true (there might be large portions of unknown space far away).

As opposed to RH-NBVP, FEP is good for large environments since it achieves complete exploration. For small environments, however, is not as good as RH-NBVP. FEP goes towards the closest frontier, which is not necessarily the most rewarding one. This results in a lot of "back and forth" and, although it will eventually reach complete exploration, it is not very efficient.

What Selin et al. came up with was an algorithm capable of exploring both small and big environments by using both RH-NBVP and FEP. The robot starts by using RH-NBVP for local exploration. Along the way, it caches good exploration points for later use (these points do not have a very high *score* and so RH-NBVP does not pick them but they have a high *gain* and are worth saving for later use). When RH-NBVP finishes local exploration (there are no points with a score above a certain threshold in the map), then AEP switches to global exploration and uses FEP along with the cached points. The cached points are potential destinations to be picked by the FEP.

This way, [29] presents a more general and efficient approach to explore any kind of environment.

2.4.6 STL-AEP

STL-AEP is yet another form of autonomous exploration. STL-AEP stands for Signal Temporal Logic - Autonomous Exploration Planner and it was proposed as an improvement to AEP in [30]. In this paper, Barbosa et al. proposed using STL to improve exploration algorithms in a user-defined way.

User-defined exploration strategies such as never getting closer than a certain distance from obstacles or focusing more on them can be translated into an STL equation that then will be interpreted as a cost function. The STL is then able to plan paths that maximally satisfy these constraints.

This type of approach can lead to several advantageous exploration properties such as efficiency (reducing back and forth), robustness, predictability, and safety.

The authors then integrated the STL with the AEP making a new exploration algorithm, the STL-AEP. Besides adding the temporal logics constraints to the AEP, Barbosa et al. also improved the AEP's efficiency. One key change was upgrading the RRT to RRT*, guarantying asymptotic optimality.

In the context of this thesis, the signal temporal logics were not used but it was

used this improved version of the AEP, the AEP with RRT*.

Chapter 3

Proposed Approach

It is now introduced the algorithm developed in this thesis, the Heterogeneous Exploration algorithm (HE). In this section, we start with some background overview in 3.1. In 3.2, the true innovation of this thesis is explained and in 3.3, the collision avoidance problem is addressed.

3.1 Exploration

As was mentioned earlier, this thesis was built on the algorithm developed in [29] and later improved in [30]. The AEP algorithm has two modes, local exploration and global exploration. The global exploration was not altered and so, for now, we will focus on the local exploration algorithm, the RH-NBVP.

When building the RRT, it is computed a score for each node, which is stated again for the convenience of the reader (equation 3.1).

$$\boldsymbol{Score}(n_k) = \boldsymbol{Score}(n_{k-1}) + \boldsymbol{Visible}(n_k)e^{-\lambda c(n_k)} \tag{3.1}$$

This means that each node's score is the sum of the previous node's score plus the visible unexplored cells multiplied by a cost factor. This cost factor is related to the distance needed to travel to that node but it was mostly left unchanged and so we will also skip it here.

The true innovation of this thesis lies within the $\boldsymbol{Visible}(n_k)$ part of the equation. Before delving into this new algorithm however, it is important to first understand how the AEP does it, so we can then describe how it was changed and why.

3.1.1 AEP's RRT

When executing the RRT algorithm, we start with the root node (current position of the robot) and add more nodes by sampling them as explained before. Each of these nodes corresponds to a flat state. A UAV's flat state can be described by the variables $state_i = (x, y, z, \phi)$ where ϕ is the yaw of the UAV.

To add a new node to the tree, x, y and z are sampled. ϕ is then calculated to maximize the *gain* when constrained to those variables. The way AEP computes the *gain* is based on the volume of unexplored space within line of sight and in range of the sampled node (the sensors have limited range).

The innovative part of this thesis is precisely what cells to consider when computing the *gain*. Instead of only considering unexplored cells, the HE considers every cell. This makes sense because we can still gain information from re-scanning a cell that has a probability different from 0 or 1. As will be seen in the next section, this type of approach is especially useful when having multiple UAVs with different capabilities where our objective is not only to explore the whole environment but also obtain a high-quality map.

3.2 Evaluation function

This thesis was aimed to obtain a good map in the smallest time possible. One way to approach the problem would be to run the AEP in all UAVs. They would be coordinated in the sense that they build the same map and go towards unexplored parts of the map. This, however, is not optimal for heterogeneous UAVs.

Imagine we have two UAVs, one with very accurate and short-ranged sensor (UAV_a) and the other with not as accurate but with a long-ranged sensor (UAV_b). To obtain a high-quality map, we would have UAV_a explore the whole environment. This, however, takes no advantage of UAV_b.

To make most of both UAVs, we can assign targets. By having UAV_b focus on unexplored space, we can then have UAV_a focus on obstacles. This results in very accurate obstacle representations and also, smaller exploration times since UAV_a will only re-explore obstacles and not the whole environment. The less cluttered an environment is, the more this effect is noticeable. Also, free space will get a high precision since free cells are usually scanned several more times than occupied ones (if the sensor was a laser rangefinder, each time it scanned the environment, it would see several free cells but only one occupied).

The way this prioritization is done is by assigning different rewards of exploring a cell based on its occupancy value. This can be done using a function that maps the occupancy value of a cell into a score as will be explained in the next section.

In the context of this thesis, UAV_a would be a UAV equipped with an RGB-D camera and UAV_b would be another equipped with 270° 2D laser scan (Lidar).

3.2.1 Algorithm

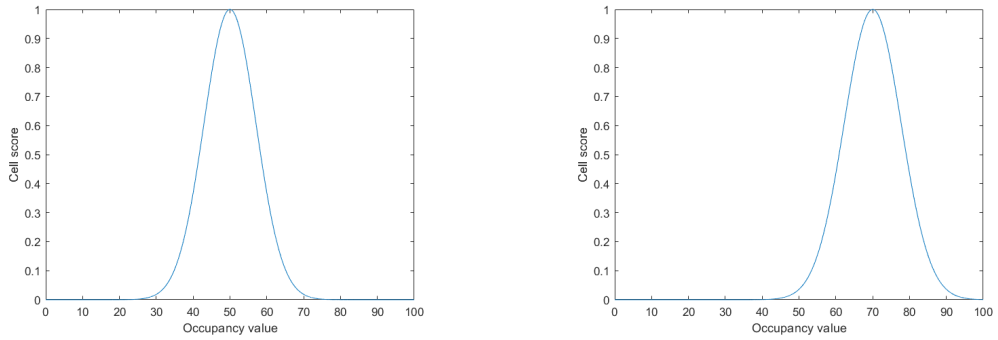
The total *gain* of a certain position is defined by summing all the *gains* of both explored and unexplored cells. The *gain* of a single cell is related to its probability of being occupied by a Gaussian-shaped function.

With this in mind, the *gain* of a single cell can be given by the following equation:

$$score(cell_i, robot_j) = K_j e^{-\frac{(occ_i - \mu_j)^2}{\sigma_j^2}} \tag{3.2}$$

In equation 3.2, the occ_i is the probability of $cell_i$ being occupied. μ_j and σ_j are two values characteristic to each UAV that should be tuned. Finally, K_j is a simple constant also different for each robot that is used to balance the *gain* and the *cost* of traveling to a certain position.

Now we can see how different UAVs prioritize different cells. In this thesis' spectrum, where we have a Lidar equipped UAV and another with an RGB-D camera, it is easy to see that the Lidar should focus on unexplored space (cells with 50% probability of being occupied) and the RGB-D should focus on already explored cells that are likely to be occupied but not very certain. This translates on focusing on cells with a probability of being occupied above 50%. This can be illustrated with image 3.1.



(a) Lidar: $K = 1$, $\mu = 50$, $\sigma^2 = 100$ (b) RGB-D camera: $K = 1$, $\mu = 70$, $\sigma^2 = 120$

Figure 3.1. Evaluation function illustrative example.

Since this method is not very restrictive, meaning that every robot can do every task, this is an easily scalable algorithm. Depending on the number of robots and on each sensor's capabilities, the values K, μ and σ have to be tweaked. These values' optimality is not covered in this thesis and is left for future research.

3.3 Collision Avoidance

When dealing with multi-robot systems, they mustn't collide. In this thesis, it is assumed that each UAV has some sort of communication system to broadcast its position. As will be explained later, this system can have a limited range without affecting the overall performance.

As explained in section 3.1, each UAV explores the environment using an RRT. In this case, the UAV travels the most promising branch of the tree. This branch can be represented by a line segment.

This way, every UAV broadcasts its path and knows the other UAVs current paths, which are the line segments.

3.3.1 Algorithm

The algorithm can be divided into two parts. First, how does each UAV keep track of the others. Second, how does the RRT take this into account.

Tracking

Each robot's path can be described by a line segment. Since a UAV occupies some space and also the trajectory is not exactly a line segment (due to perturbations like wind), it is useful to represent it as a cylinder in 3D space.

To avoid a collision, each UAV broadcasts continuously its path. When choosing a new path, the UAV first checks that there is no overlap between the cylinders and only then does it start to publish.

Algorithm 3: Unobstructed path

```
Read and save broadcasted paths;
Pick a path;
Check for collisions;
while Path has collisions do
    Pick a new path;
    Check for collisions;
end
```

As for the collision check, it is done by making sure that the cylinders do not touch. In other words, the two line segments that define the paths must be further from each other than a certain threshold.

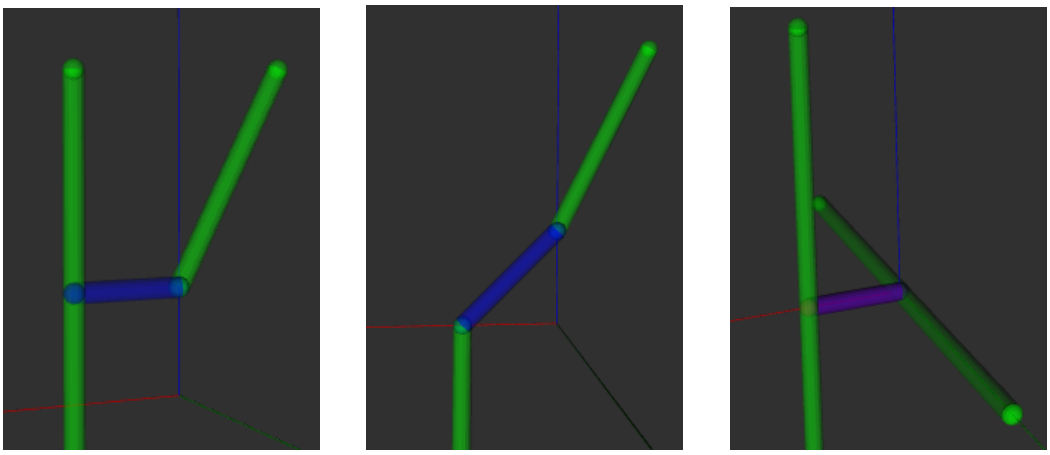
This algorithm's complexity grows linearly with the number of robots. As will be seen in the next sub-section, this algorithm is computed several times every iteration and so, a large number of robots might affect performance. This however, does not pose a problem because there can only be so many robots within communication range.

RRT Integration

The collision avoidance algorithm is used when building the RRT, to make sure that the tree created for the exploration, does not go through already established paths.

There are two ways that this could be done. The first one would be to discard every edge that intersects the established paths. The second one would be to only discard the edges connected to the root that intersect the established paths (this might be a good approach since the robot only traverses through the first edge of its tree before it generates a new tree).

The second approach seems promising, efficiency-wise, but might present some problems. Since there is no way to know how long the established paths will remain



(a) (b) (c)

Figure 3.2. Closest points scenarios

active, the UAV building the RRT might get stuck. It keeps building the RRT through the established path, never really able to go through it and so it gets stuck until the established paths are cleared. In an extreme case, this might result in a deadlock, where we have two UAVs trying to go through each other, never being able to do so.

With this in mind, the first approach was implemented. Indeed, it might not be as efficient since most established paths are cleared relatively fast but on the other hand, the risk of deadlock is eliminated.

3.3.2 Decentralized System

The collision avoidance algorithm was implemented in a decentralized fashion so that it is easily scaled to more than two UAVs and is more versatile.

Also, this algorithm is completely independent of the rest of the system, making it easy to integrate into other programs.

There is, however, one issue that might arise and that is, what happens when communication fails or what if the UAVs are not in communication range? Will they collide?

To solve the first problem, every UAV broadcasts its path at a high frequency, higher than the frequency at which paths are removed. The only way that a path would accidentally be removed would be if a UAV did not receive several messages in a row, which is highly unlikely.

The second problem, it was not addressed since if two UAVs are not in communication range then they are considered to be far enough to have any relevant impact on each others exploration.

3.3.3 Limitations

This algorithm has some limitations that should be addressed to make it more versatile and reliable.

The first limitation is that the paths are considered to be straight and cannot take any other shapes.

The second one has to do with asynchronicity. It might happen that two UAVs broadcast two intersecting paths at the same time. If this happens, it might result in a UAV crash although it is very unlikely for several reasons. Even though the paths intersect, there is a small chance that they will collide. The probability of two UAVs publishing the intersecting paths at the same time is very low. For these reasons, this problem was not addressed.

Chapter 4

Simulation and Results

4.1 Simulation environment

The exploration algorithm was implemented in C++ and uses ROS [31], Melodic version.

The simulation was ran using gazebo [32], on a computer with a Intel® Core™ i7-6850K CPU @ 3.60GHz × 12 processor, 128 GB RAM and running Ubuntu 18.04.

4.2 Evaluation function

The main hypothesis of this thesis is that, by using the evaluation function and giving priorities to each UAV, it is possible to achieve a more efficient exploration than if we were to use just the AEP on both UAVs. A more efficient exploration is a faster exploration.

By making the UAV with the camera focus on obstacles, it does not lose time exploring empty space. Since the UAV with the camera would have to scan the obstacles anyway to achieve a reasonable map accuracy, the proposed approach should be faster.

To test the hypothesis, it was developed an algorithm to compute the uncertainty of the map and not the absolute error. The error itself, as in the difference between obtained map and the actual map, is not very relevant to this thesis since we are only proving that the same accuracy in the map can be achieved in a shorter time. To compare accuracies between maps, explicit computation of the error does not need to be done, only a metric that is somehow related to the error. The metric chosen was map uncertainty that corresponds to how sure the algorithm is that the free space is actually free and the occupied space is indeed occupied. This way, a measure of uncertainty is computed for every cell and then all the cell's uncertainties are summed. To compute the cell uncertainty, it was used equation 4.1 that can be visualized in figure 4.1.

$$uncertainty_i = (0.5 - |0.5 - occupancy_i|)^2 \tag{4.1}$$

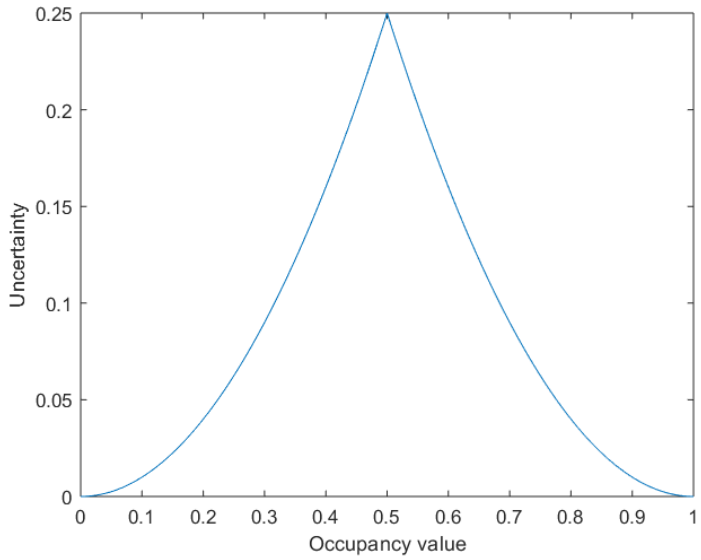


Figure 4.1. Uncertainty *score* of $cell_i$

The overall map uncertainty can then be computed by equation 4.2 where N is the total number of cells.

$$uncertainty = \sum_{i=1}^{N} uncertainty_i \tag{4.2}$$

It should be noted that this algorithm does not a measure error or precision. It is simply a way to compare the efficiency of algorithms. Also, the quality of the map is considered best when the *score* is 0, meaning that all the cells have an occupancy value of either 0 or 1. This, however, is not necessarily true. A cell that is half occupied should have an occupancy value of 0.5. This, however, was ignored for three reasons. First, the smaller the grid cells are, the less noticeable this effect is and so it can be tested for higher accuracies more close to reality. Secondly, the proportion of cells with occupancy values of 0 or 1 over the others, is extremely high and so these special cases can be ignored. Thirdly, this error can be viewed as an offset. Since all simulations will be run with this evaluation function, all results should have this offset. This means that when we compare the results among themselves, this should make no difference.

4.2.1 Expected Outcome

To prove the hypothesis, three scenarios will be simulated.

1. **Autonomous Exploration Planner (AEP)**: Both UAVs are running the AEPlanner algorithm while sharing the same map.

2. **Uncoordinated Heterogeneous Exploration (UHE)**: The UAVs are running the Heterogeneous Exploration algorithm but both Gaussians are centered around 50 ($\mu = 50$ in equation 3.2). This means that they give priority to unexplored cells over everything else. This algorithm is very similar to information-based exploration algorithms and should behave somewhat similar to AEPlanner.

3. **Heterogeneous Exploration (HE)**: This is the algorithm developed that should improve exploration efficiency over the two previous ones.

The overall uncertainty should evolve like figure 4.2 for all three approaches. This is especially true for open maps, with big areas of empty space.

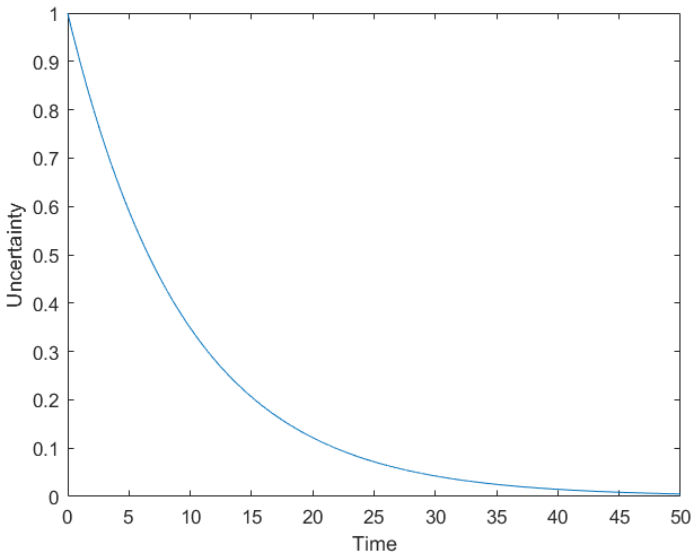


Figure 4.2. Normalized expected overall uncertainty evolution

The difference between the three graphs will be that, in the first and second case, the uncertainty starts decreasing very fast but slows significantly in a short period. In the third case, where UAVs are coordinated, the uncertainty should decrease a bit slower in the beginning but compensate towards the end. This is because exploring the empty space is the most rewarding action when considering uncertainty. Although this might be true, the purpose of this algorithm is not only exploring space but get an accurate map as well. This will result that certain areas will be explored more than once to refine the map. This is when the developed algorithm should make a difference and be more efficient.

4.3 Experimental Results

To prove the hypothesis, the three algorithms were tested in three different maps with different complexities. The HE algorithm is compared with both UHE and AEP for every map.

In every map, the UAVs start in distant positions from each other. This was done so that the AEP has the best performance. For example, in the big empty room (figure 4.3a), if the UAVs started close to each other, the camera-equipped UAV would have trouble exploring the unknown environment. This is because the Lidar has more than twice the range of the camera and so all the space around the UAV with the camera is already explored. Theoretically, this would not pose a problem but due to efficiency constraints, it is difficult to generate a tree with more than 20m in radius (equivalent to the Lidar's range) in a reasonable time. To avoid this problem, the robots were placed far apart.

It should be noted that the Heterogeneous Exploration algorithm is barely affected by this issue. This was implemented this way so that it is fair to compare HE and AEP performances.

It should also be said that not all cells were considered for evaluation purposes.

- The size of the space used to compute the score is not the same as the actual rooms. It was used a 100m square room that encloses each map. This allows some degree of comparison between different sized maps.

- The size of the actual voxels is bigger than the ones used to compute the score. This will result in counting all occupancy cells a constant amount of times. In practice, this has no relevant consequences.

- Only occupancy cells that were 60cm above ground level or higher were considered. This was done because, in large open areas, the floor constitutes the majority of occupied cells making it difficult to analyze the results. In principle, this should not affect the results since the floor can be considered as an obstacle. While this might be true, the UAV with the Lidar could not explore the floor due to the sensor's nature and so it made no sense to consider the floor's cells.

For each map, there are several graphs presented. For simplicity, they are described below. Also, the reader should bear in mind that the absolute values represented in graphs **(a)** and **(g)-(l)** are not relevant, only the comparison between graphs. These values presented depend on the size of the map. For example, a bigger map will have higher values than a smaller one since it has more grid cells in it.

1. **(b) Total uncertainty over time**: Represents the total uncertainty of the map as calculated by eq. 4.2.

2. **(c) Average uncertainty over time**: Same graph as (a) but divided over the total number of cells analyzed.
3. **(d) Average uncertainty of explored cells over time**: The uncertainty of all the cells with an occupancy value different from 0.5 was summed and divided by the total number of summed cells.
4. **(e) Average uncertainty of occupied cells over time**: The uncertainty of all the cells with an occupancy value greater than 0.5 was summed and divided by the total number of summed cells.
5. **(f) Average uncertainty of free cells over time**: The uncertainty of all the cells with an occupancy value lesser than 0.5 was summed and divided by the total number of summed cells.
6. **(g), (i), (k): Number of free cells in uncertainty interval**: Free cells were counted and grouped per occupancy value.
7. **(h), (j), (l): Number of occupied cells in uncertainty interval**: Occupied cells were counted and grouped per occupancy value.

Figure	Cell Uncertainty	Map Uncertainty
b)	$u_i = (0.5 - \lvert 0.5 - occ_i \rvert)^2$	$u_T = \sum_{i=1}^{N} u_i$
c)		$u_A = \frac{1}{N} \sum_{i=1}^{N} u_i$
d)		$u_A = \frac{\sum_{i=1}^{N} \begin{cases} u_i, & \text{if } occ_i \neq 0.5 \\ 0, & \text{otherwise} \end{cases}}{\sum_{i=1}^{N} \begin{cases} 1, & \text{if } occ_i \neq 0.5 \\ 0, & \text{otherwise} \end{cases}}$
e)		$u_A = \frac{\sum_{i=1}^{N} \begin{cases} u_i, & \text{if } occ_i > 0.5 \\ 0, & \text{otherwise} \end{cases}}{\sum_{i=1}^{N} \begin{cases} 1, & \text{if } occ_i > 0.5 \\ 0, & \text{otherwise} \end{cases}}$
f)		$u_A = \frac{\sum_{i=1}^{N} \begin{cases} u_i, & \text{if } occ_i < 0.5 \\ 0, & \text{otherwise} \end{cases}}{\sum_{i=1}^{N} \begin{cases} 1, & \text{if } occ_i < 0.5 \\ 0, & \text{otherwise} \end{cases}}$
g), i), k)		$u_{[a,b[} = \sum_{i=1}^{N} \begin{cases} u_i, & \text{if } occ_i \in [a,b[\\ 0, & \text{otherwise} \end{cases}$, $(a,b) \in [(0,0.1),(0.1,0.2),(0.2,0.3),(0.3,0.4),(0.4,0.5)]$
h), j), l)		$u_{[a,b[} = \sum_{i=1}^{N} \begin{cases} u_i, & \text{if } occ_i \in [a,b[\\ 0, & \text{otherwise} \end{cases}$, $(a,b) \in [(0.5,0.6),(0.6,0.7),(0.7,0.8),(0.8,0.9),(0.9,1)]$

Now, the results for the three maps are presented. The maps are very distinct from one another. The first one, the Big Empty Room, was designed to show the main strengths of this thesis due to the large amount of empty space. The third map, the Office, is a very cluttered environment used to show that the algorithm developed is versatile enough to work in any kind of environment. Finally, the second map, the Room with Many Walls, is somewhere in between the two previous maps, concerning complexity.

4.3.1 Big empty room

The first map where the algorithms were tested was an empty square room with 50m sides (figure 4.3a) and with the UAVs 20 meters apart, close to the wall. The UAVs were placed like this because it allows easy tuning. The distance between the UAVs is the same as the Lidar range. This results in the UAV with the camera being exactly in the frontier created by the Lidar. This way, the camera can choose to go for unexplored space or re-explore the walls (which is preferred). This is the optimal case scenario. If the robots had another configuration, the results should be slightly worse but still better than the other two algorithms since this was the case for every map. It should also be said that the parameter values tuned with this map were the same for every map.

Despite the UAV's configuration, this type of environment is where the HE should perform best since it has large empty areas that the UAV with the camera can skip. What can be observed is that the UAV with the Lidar explores the whole environment relatively fast (Lidar has a range of 20m) and then the camera can only focus on the walls.

In fact, in this environment, the HE outperforms the other two algorithms in every aspect. This is explained by their behaviors.

The AEP is only guided by the unexplored environment. When there are no more accessible cells to explore, then the UAVs will start moving randomly (because of residual gains). This will result in a low-quality map since most of it was explored by the Lidar. The free space will present high accuracy since it is scanned considerably more than occupied space as explained before.

The UHE algorithm presents a similar behavior to the AEP. Since the evaluation Gaussian is centered around 50%, it will start exploring unexplored space and when everything's explored, it will move approximately randomly. This is because of two reasons. First, the free space has a high degree of certainty from the Lidar as explained. The occupied cells, on the other hand, are so few compared to free ones that will not have much of an impact on the gain.

Finally, HE was the algorithm that performed the best in all metrics. Occupied cells uncertainty was expected to be lower than other approaches but not free cells. The UAV with the camera showed a tendency to follow the wall and so focus on occupied cells mostly (the reason why occupied cells are more accurate) but by doing this, the camera ended up exploring a bigger volume of space as well, resulting in a higher free cell accuracy.

The HE achieved higher accuracy for occupied cells by 46% compared to both UHE and AEP.

Discussion

Both figures 4.3b and 4.3c have the same shape. This is because 4.3c can be obtained by dividing every value in 4.3b by the total number of cells (constant). By observing these graphs, we can see that the HE starts behaving a little worse than the other two algorithms. This is to be expected as mentioned before because the HE aims for efficiency in the long term. It starts not as efficient as the other two algorithms because it does not prioritize unexplored space but as the map goes towards fully explored, we can see the HE achieving better overall accuracy. Also, it can be seen that figure 4.3c starts at the value 0.25. This is the uncertainty value of an unexplored cell (as would be expected as no cells have been explored yet).

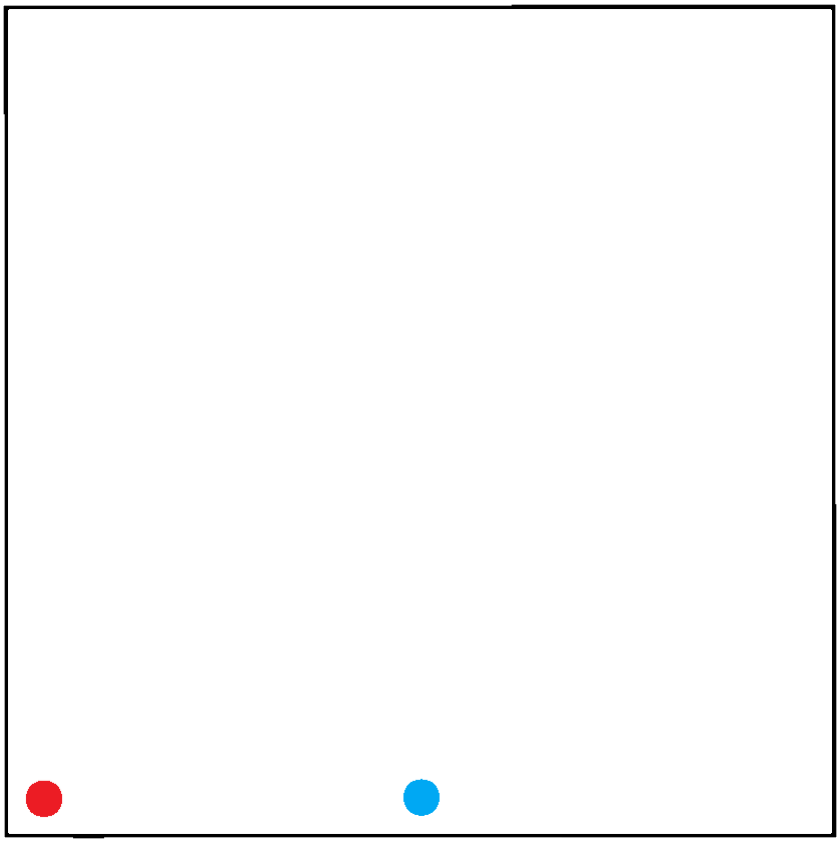
Plots 4.3d and 4.3f are also very similar but not equivalent. Figure 4.3d considers every explored cell while 4.3f only the free ones but since free cells are two orders of magnitude more than occupied ones, they contribute the most to plot 4.3d. In these plots, HE behaves a little bit better than the other algorithms although the difference is not very big. This happens because the Lidar scans every free cell several times and although it has low accuracy, due to the number of measurements, free cells tend to get a low uncertainty.

The plot where the HE algorithm advantages are noticeable the most is figure 4.3e. In this figure, the average uncertainty of occupied cells obtained by the HE is clearly lower than the other two algorithms. This is the most relevant experimental result. It shows that occupied cells are clearly the target of the system. It can also be seen that the average uncertainty of occupied cells is lower at all times for the HE. In conjunction with the other plots described above, it can be concluded that the HE clearly obtains better accuracy for occupied voxels while not lowering the overall accuracy in long term exploration. This shows that if accuracy is required, the HE algorithm has no drawbacks.

Figures 4.3g, 4.3i and 4.3k show the amount of explored free cells in the map, grouped by occupancy value. As was explained before, we can see that the amount of high accuracy cells for the HE is indeed higher than the other two approaches contrary to what was to be expected. In fact, when all the map is explored, both the AEP and the UHE stop having clear objectives and their behavior appears more or less erratic. As for the HE, it will continue exploring. The UAV with the camera, in particular, will tend to follow the wall all around which will result in high accuracy occupied cells but free cells as well. By following the wall, the camera will cover a big portion of free space as well.

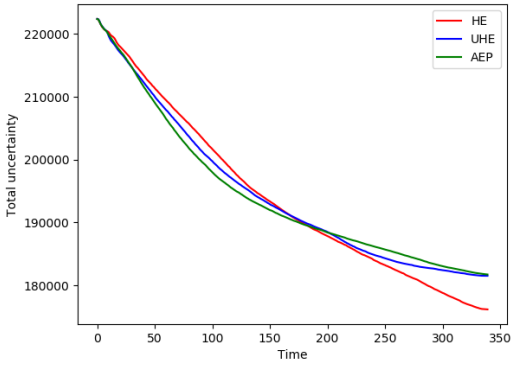
Finally, by looking at plots 4.3h, 4.3j and 4.3l we can see that the HE does indeed obtain more high accuracy cells. In figures 4.3j and 4.3l which correspond to the UHE and AEP respectively, we can see that every interval of uncertainty gains more and more cells as time passes but at a certain point, this growth more or less stops. This happens because the environment is fully explored and so there

are no more objectives and exploration starts being very inefficient. As for figure 4.3h corresponding to the HE algorithm, this does not happen. In fact, it keeps improving explored cells and the low uncertainty cells (brown line) keep increasing as time passes in a more or less linear form. As we will see, linearity is characteristic to this map since the camera follows the wall at a more or less constant speed.

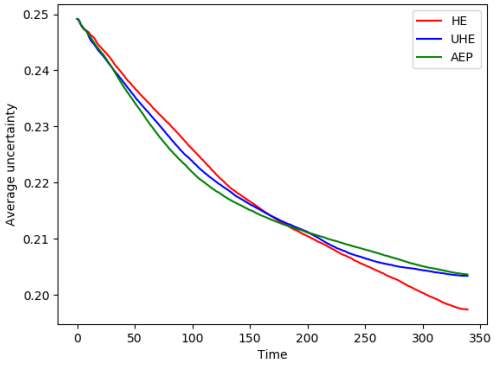


(a) Big Empty Room map and UAVs starting positions. UAV with Lidar in red and UAV with RGB-D camera in blue.

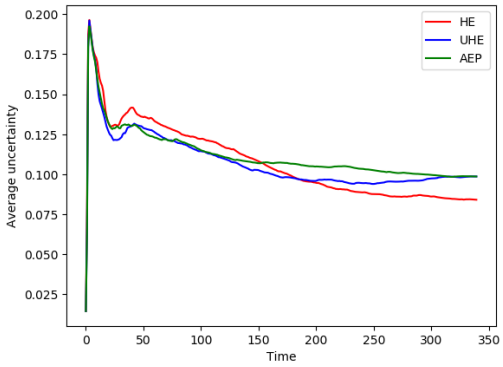
Figure 4.3. Big Empty Room



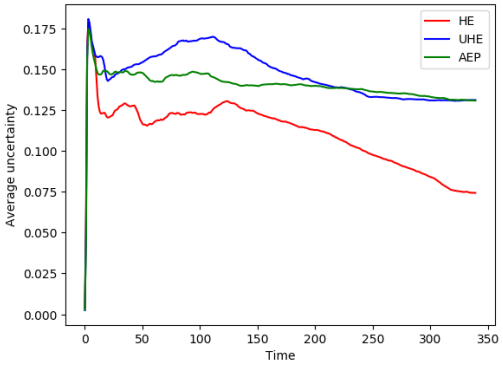
(b) Total uncertainty over time



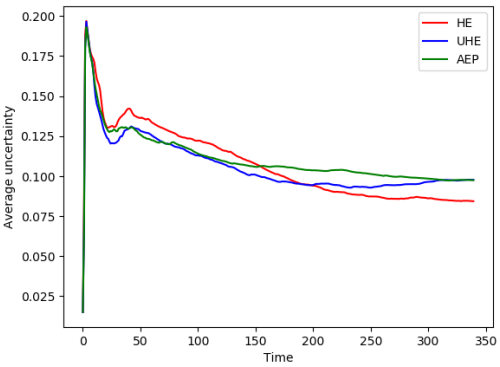
(c) Average uncertainty over time



(d) Average uncertainty of explored cells over time

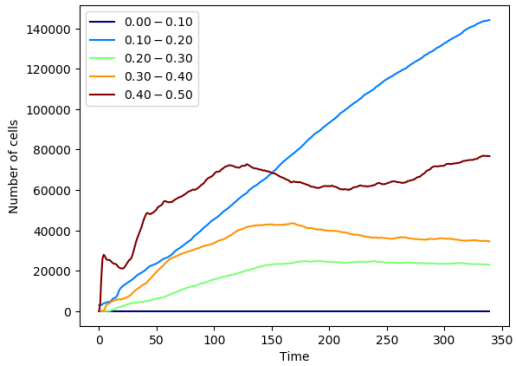


(e) Average uncertainty of occupied cells over time

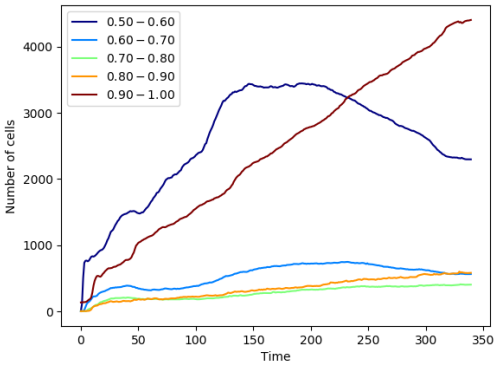


(f) Average uncertainty of free cells over time

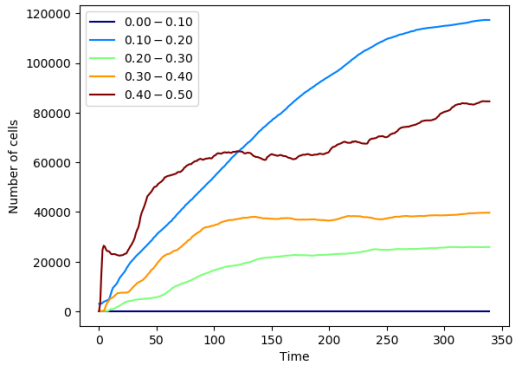
Figure 4.3. Big Empty Room



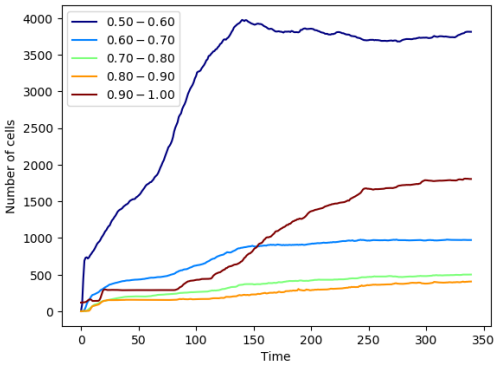
(g) Number of free cells in uncertainty interval for HE algorithm



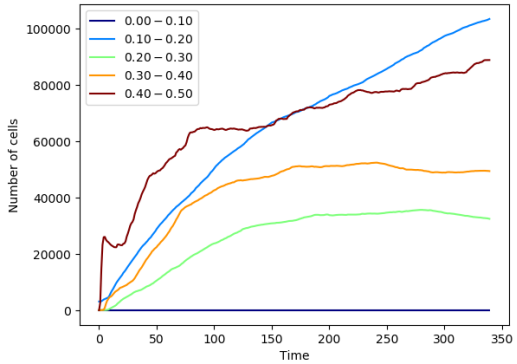
(h) Number of occupied cells in uncertainty interval for HE algorithm



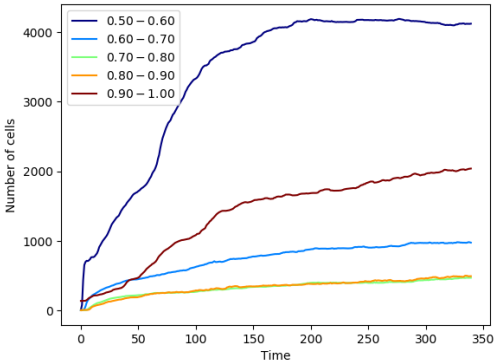
(i) Number of free cells in uncertainty interval for UHE algorithm



(j) Number of occupied cells in uncertainty interval for UHE algorithm



(k) Number of free cells in uncertainty interval for AEP algorithm



(l) Number of occupied cells in uncertainty interval for AEP algorithm

Figure 4.3. Big Empty Room

4.3.2 Room with Many Walls

This was the only map were the HE algorithm did not outperform the other in all metrics. It behaved slightly worse than UHE when considering free cell accuracy. This can be explained by the fact that the UHE focuses mainly on unexplored and low-accuracy empty space. Despite this, HE still achieved higher accuracy for occupied cells by 20 % compared to UHE and 33% compared to the AEP.

Discussion

In this map, the HE performs slightly worse than the UHE, when considering the overall uncertainty. By analyzing the plots 4.4b and 4.4c, total uncertainty and average uncertainty of the whole map respectively, we can see that the blue line has the lowest uncertainty. The HE and AEP, on the other hand, are very similar. The reason why this happens is that the rooms are relatively long. To explain this, it is easier to describe the behavior of the UAV with the camera when running the UHE and HE algorithms.

The HE algorithm pushes the UAV with the camera towards obstacles but since the back walls of each room are relatively far back and have a small area, the reward of going there is small. This results in the fact that the camera will not go there and so, free space will not be explored as much.

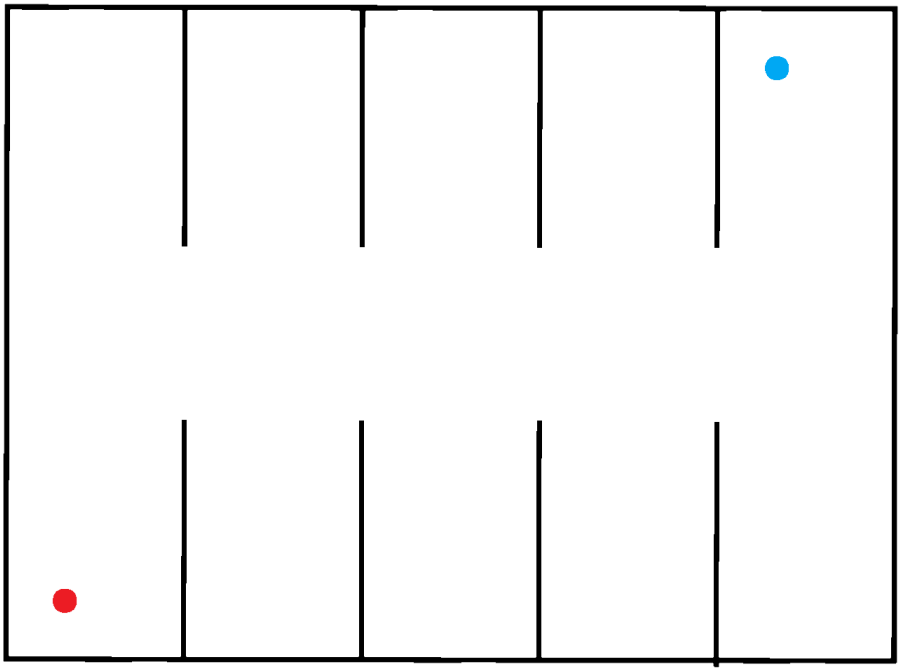
On the other hand, when using UHE, the camera feels a lot more compelled to travel to the end of the room because it has a lot of free space and so it ends exploring more.

This explains the difference when looking at the accuracy difference of free space represented in figure 4.4d and 4.4f. It should also be mentioned that in this environment, the Lidar does not obtain as high accuracies of free space as in the big empty room. This is because the environment is a lot more cluttered and so the laser scans do not cover as much space.

With all this, the HE still obtains better accuracy for occupied cells than the other algorithms as can be seen in figure 4.4e. Now, there seems to be a contradiction here, if the UHE pushes the camera further back into the rooms, should it not obtain higher obstacle accuracy as well? The answer is no. In fact, the Lidar tends to spend considerably more time in the starting room and so, the free space around it gets very accurate with just the Lidar. Since the free space is very accurate, the UHE algorithm will not send the camera there but the HE will since the Lidar alone can never achieve high accuracy occupied cells.

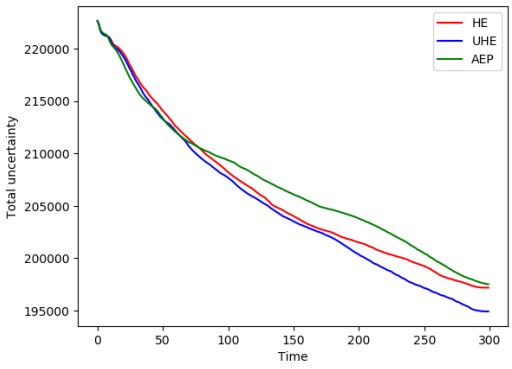
All this that was said can also be extrapolated from the other figures. We can see that the HE (figure 4.4g) has less high accuracy free cells than the other two algorithms (figures 4.4i and 4.4k) since these last two have a tendency to explore more free cells. As for the occupied cells, we can see a decrease in low accuracy occupied cells in image 4.4h around second 50 that corresponds to the point where all space is more or less explored but the camera keeps re-scanning obstacles for better accuracy. In figures 4.4j and 4.4l, the decrease in low accuracy occupied cells

and increase in high accuracy occupied cells is a lot less noticeable from around second 50 onwards.

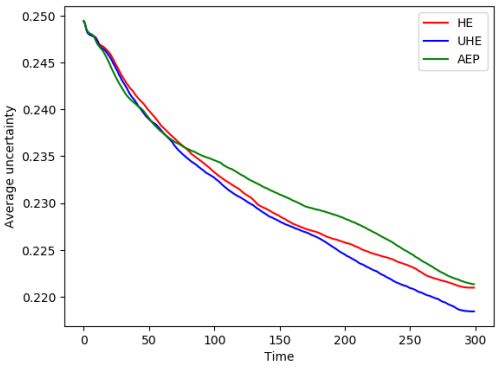


(a) Room with Many Walls map and UAVs starting positions. UAV with Lidar in red and UAV with RGB-D camera in blue.

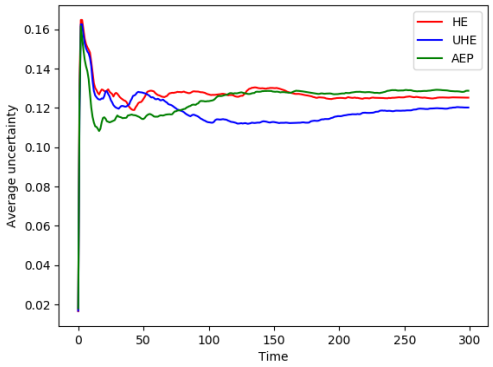
Figure 4.4. Room with Many Walls



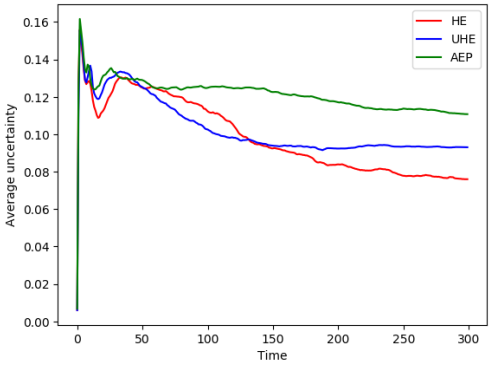
(b) Total uncertainty over time



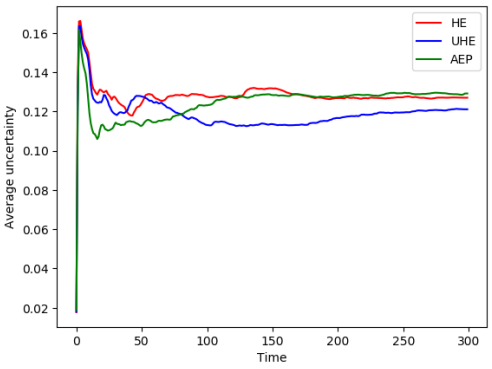
(c) Average uncertainty over time



(d) Average uncertainty of explored cells over time

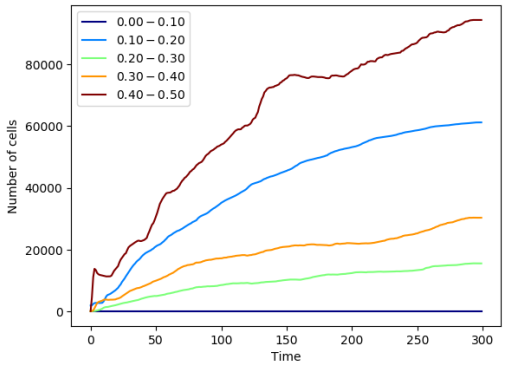


(e) Average uncertainty of occupied cells over time

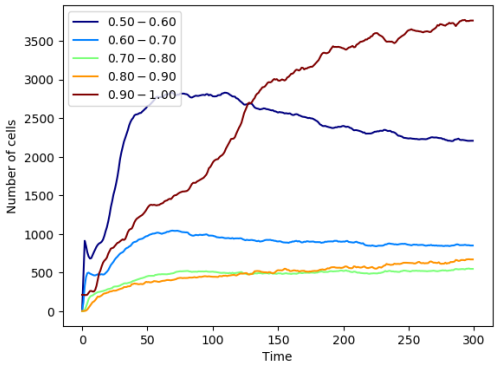


(f) Average uncertainty of free cells over time

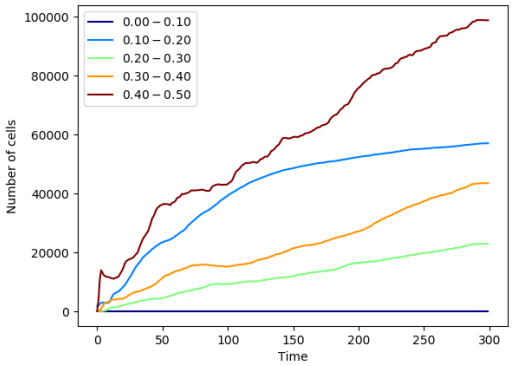
Figure 4.4. Room with Many Walls



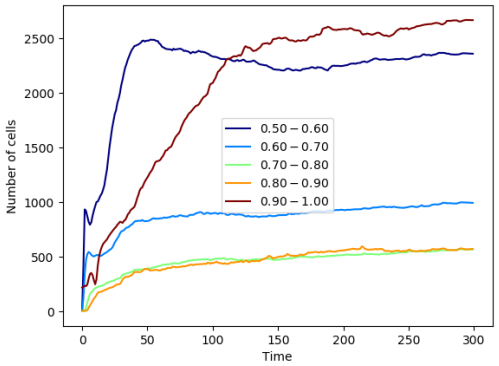
(g) Number of free cells in uncertainty interval for HE algorithm



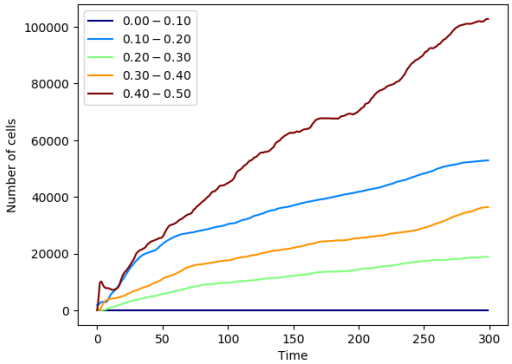
(h) Number of occupied cells in uncertainty interval for HE algorithm



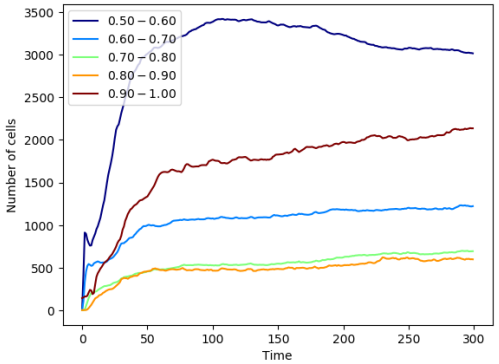
(i) Number of free cells in uncertainty interval for UHE algorithm



(j) Number of occupied cells in uncertainty interval for UHE algorithm



(k) Number of free cells in uncertainty interval for AEP algorithm



(l) Number of occupied cells in uncertainty interval for AEP algorithm

Figure 4.4. Room with Many Walls

4.3.3 Office

The last environment where the algorithm was tested was in an office. The office has the characteristic of being considerably more complex and with narrower passages. This was the environment with the smallest volumes of free space but still, the HE managed to outperform the other algorithms. This was mainly because the HE does not run out of waypoints as fast as the other algorithms (free space gets explored with high accuracy relatively fast as opposed to occupied space).

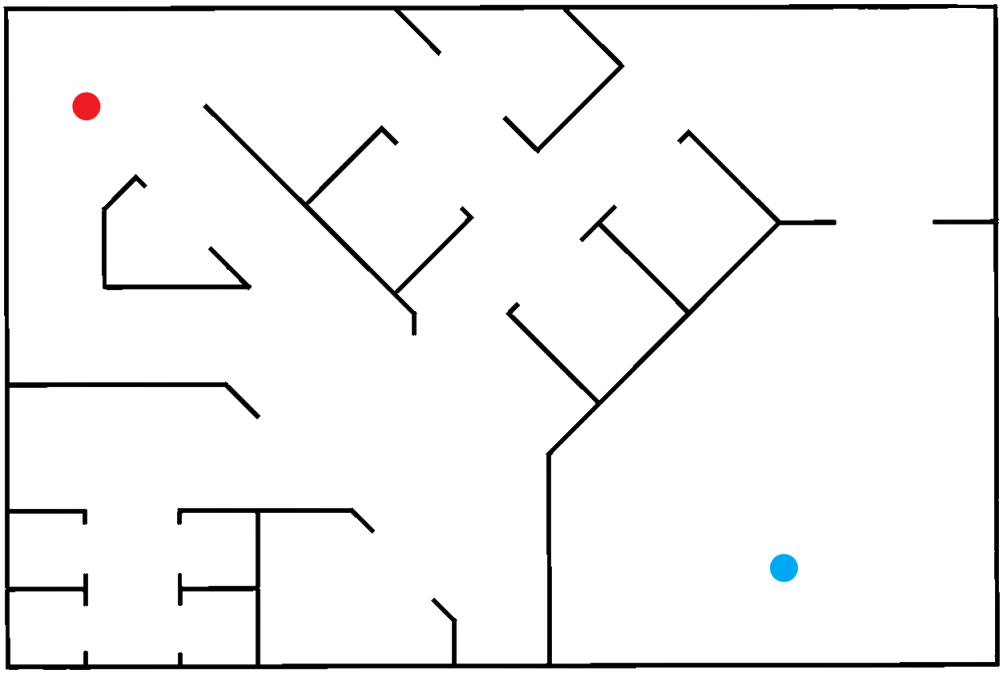
HE still achieved higher accuracy for occupied cells by 30% compared to UHE and 15% compared to the AEP.

Discussion

In this map, the results are fairly similar for all algorithms and this is mainly because there are no large empty areas that can be skipped by the camera. This being said, the HE still performs the best in all metrics.

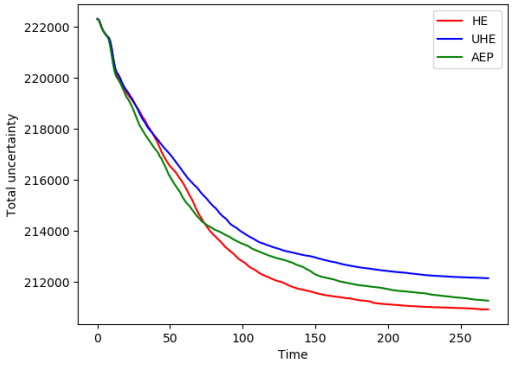
When analyzing figures 4.5b and 4.5c, we can see that the red curve (HE) is slightly lower than the others. The reason why this happens is that the HE is able to keep exploring even when the whole environment has been scanned at least once. Both the UHE and AEP show an erratic behavior when there is no unknown space left and so the camera does not re-explore as much space. This explains why the accuracy is better for the HE in all metrics. The same conclusions can be drawn from figures 4.5d, 4.5e and 4.5f.

When looking at the graphs 4.5g through 4.5l, the HE obtains more high accuracy cells (both free and occupied) but the shapes of the graphs between algorithms are fairly similar, indicating that the advantage of the HE lies in having more goal-points as explained in the previous paragraph.

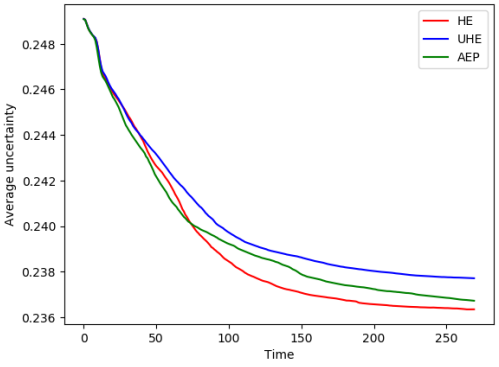


(a) Office map and UAVs starting positions. UAV with Lidar in red and UAV with RGB-D camera in blue.

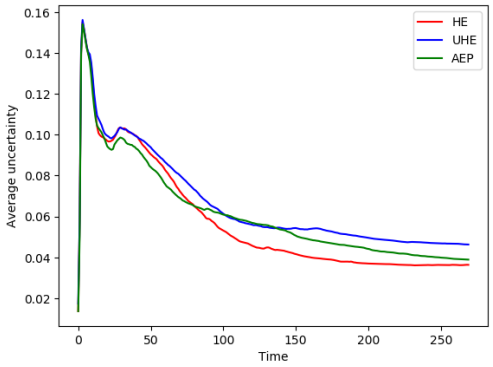
Figure 4.5. Office



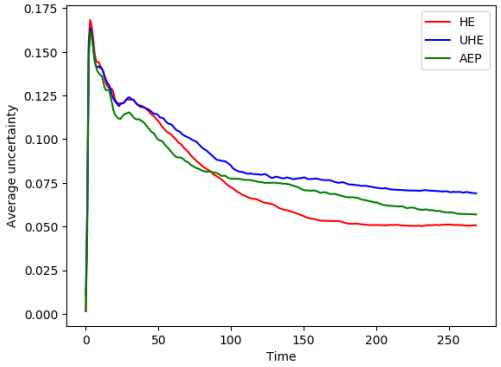
(b) Total uncertainty over time



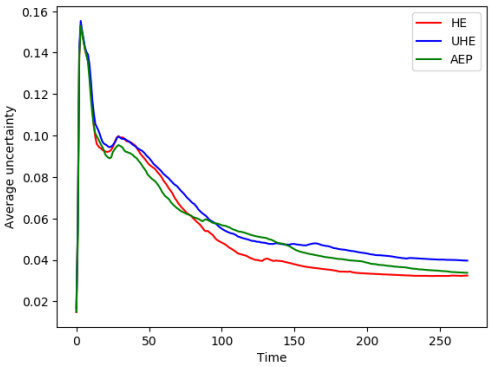
(c) Average uncertainty over time



(d) Average uncertainty of explored cells over time

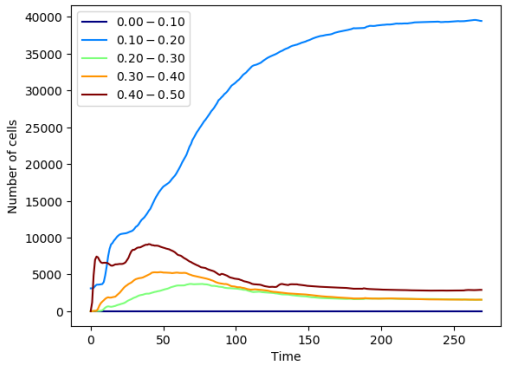


(e) Average uncertainty of occupied cells over time

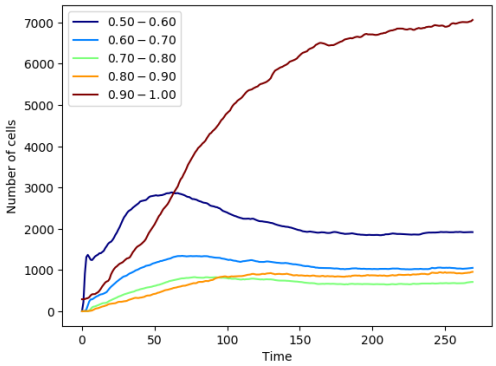


(f) Average uncertainty of free cells over time

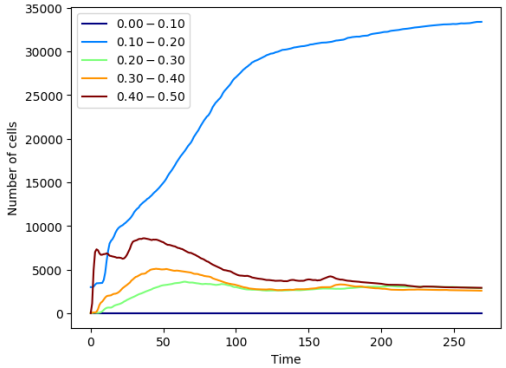
Figure 4.5. Office



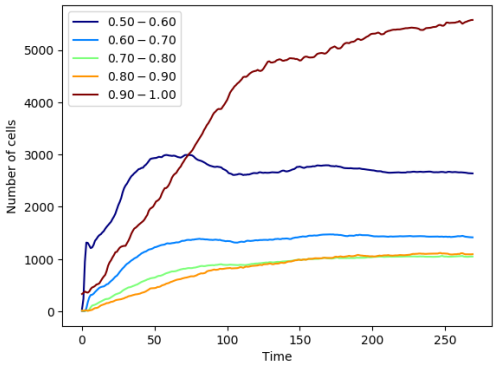
(g) Number of free cells in uncertainty interval for HE algorithm



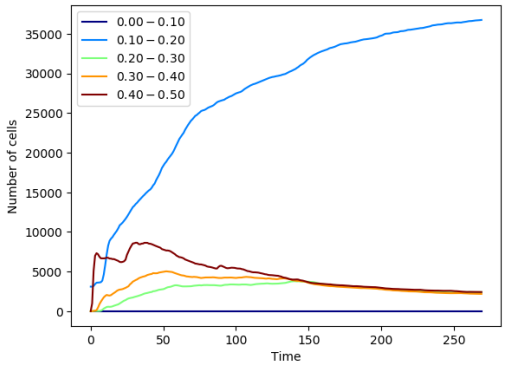
(h) Number of occupied cells in uncertainty interval for HE algorithm



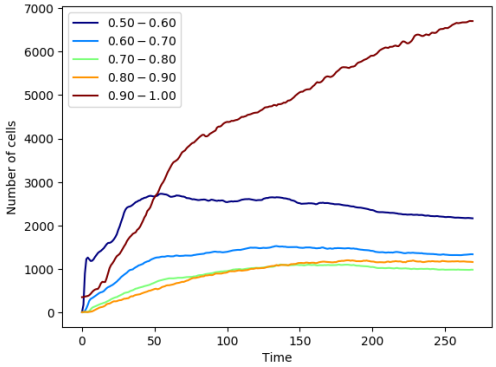
(i) Number of free cells in uncertainty interval for UHE algorithm



(j) Number of occupied cells in uncertainty interval for UHE algorithm



(k) Number of free cells in uncertainty interval for AEP algorithm



(l) Number of occupied cells in uncertainty interval for AEP algorithm

Figure 4.5. Office

Chapter 5

Conclusion and Future Work

The main objective of this thesis was to develop an algorithm to coordinate heterogeneous UAVs. This was achieved by prioritizing each UAV towards the most advantageous waypoint based on their characteristics. Instead of only focusing on complete exploration as most papers do, this thesis focused on both complete and high-quality exploration.

As shown by the results, the Heterogeneous Exploration algorithm achieves overall better precision in most cases compared to other algorithms. By prioritizing tasks but at the same time giving some freedom to each robot, HE can potentially explore indefinitely (it never runs out of goals), as opposed to AEP, which might be good if map accuracy is an important factor. Also, by giving some preference towards obstacles, the HE algorithm always achieves better precision around obstacles. This effect is especially noticeable in large open areas.

5.1 Future Work

The results are very promising but there are still some improvements that could be done to improve this algorithm's reliability and efficiency.

Multi Robot RRT

As explained in section 2.4.5, the AEP caches high gain nodes for later use. The issue with this in multi-robot systems is that each robot caches its nodes they do not share them. This will lead to robots avoiding to go into areas already explored by other robots. This is not ideal because some areas are worth exploring more than once, usually by different robots with different characteristics.

Also, the RRT proved to be quite computationally demanding when having the two UAVs running at the same time in the same computer. The algorithm's efficiency should be improved as well.

Bid System

The HE works especially good when the robots have different tasks. When we have several similar robots, however, it does not make sense to give them different tasks. When having similar tasks, however, the robots act uncoordinated. This is because they do not communicate other than sharing a map. When they have different tasks, they will inherently move towards different positions but when they have similar tasks, they may go to similar locations. This poses efficiency issues and should be improved to allow the HE to work with any combination of robots. One way this could be achieved would be to implement a bidding system like explained in 2.3.2 along with the HE algorithm.

Cell Evaluation

One issue with using Gaussians as a cell evaluation metric is that it assumes the occupancy value of all the cells is uniformly distributed. This is generally not the case since the number of free cells is usually several times more than occupied ones for most environments. This is an issue because of sensors' nature. Every scan will analyze free cells a lot more than occupied ones and so, free and unexplored space will have a bigger impact on the evaluation function compared to occupied space. This makes the system less robust and harder to tune.

If we go back to figure 3.1a in section 3.2.1, the Gaussian centered around 50% should give preference to neither occupied space nor free space. In reality, what happens is that the robot will give preference to free space since free cells give considerably more contribution to the reward calculation due to their relative occurrence.

One solution to this problem would be to weight every cell based on its occupancy value and the estimated distribution of occupancy probabilities.

Decentralized System

The system implemented in this thesis was a centralized one. This was done because it is simpler to implement and analyze and so it made proving the hypothesis easier. Many real-world applications, however, require decentralized systems. This might be because of communication constraints, scalability or other reasons and so, HE could still be improved by making it a decentralized system. This would mean each robot having its map representation and merging them whenever in communication range.

Parameters Optimality

A good parameter tuning proved to be very important and although a system badly tuned works as well, it is no longer efficient. This however, proved to be quite difficult and time-consuming. For small systems composed of just a few robots, this

is not an issue but when having more complex systems, it is impractical to trial and error all the parameters.

For complex systems, we must have a more systematic way of computing these parameters. The parameters that need tuning are the mean μ and standard deviation σ of the Gaussian of the evaluation function. Changing the function itself, from Gaussian to another function might also be a solution.

5.2 Conclusion

This thesis was aimed at developing an efficient and accurate exploration strategy. By developing a prioritization strategy, it was possible to assign different UAVs to different areas and so improve efficiency.

In the HE algorithm, each UAV weighs grid cells differently, allowing them to move towards unexplored or explored but inaccurate space. By doing this, it is possible to save time towards complete map exploration.

The HE algorithm was compared to two other algorithms in three different scenarios and was proven to be better in almost every metric in the long run. In fact, obstacle-wise, it always obtains better precision. It is only concerning free space that the HE does not always achieve better accuracy.

This thesis was developed so that the algorithm is versatile and can be scaled. The proposed strategy can be adapted to any number of UAVs and sensors.

Bibliography

[1] Dieter Fox et al. “A probabilistic approach to collaborative multi-robot localization”. In: *Autonomous robots* 8.3 (2000), pp. 325–344.

[2] Robert Zlot et al. “Multi-robot exploration controlled by a market economy”. In: *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*. Vol. 3. IEEE. 2002, pp. 3016–3023.

[3] M Baglietto et al. “Information-based multi-agent exploration”. In: *Robot Motion and Control, 2002. RoMoCo'02. Proceedings of the Third International Workshop on*. IEEE. 2002, pp. 173–179.

[4] Reid Simmons et al. “Coordination for multi-robot exploration and mapping”. In: *Aaai/Iaai*. 2000, pp. 852–858.

[5] Jing Yuan et al. “A cooperative approach for multi-robot area exploration”. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2010, pp. 1390–1395.

[6] Kazunori Yamaguchi et al. “Octree-related data structures and algorithms”. In: *IEEE Computer Graphics and Applications* 1 (1984), pp. 53–59.

[7] Armin Hornung et al. “OctoMap: An efficient probabilistic 3D mapping framework based on octrees”. In: *Autonomous robots* 34.3 (2013), pp. 189–206.

[8] Lourdes Munoz-Gómez et al. “Exploration and map-building under uncertainty with multiple heterogeneous robots”. In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 2295–2301.

[9] Wolfram Burgard et al. “Collaborative multi-robot exploration”. In: *ICRA*. 2000, pp. 476–481.

[10] Wolfram Burgard et al. “Coordinated multi-robot exploration”. In: *IEEE Transactions on robotics* 21.3 (2005), pp. 376–386.

[11] M Bernardine Dias and Anthony Stentz. “A free market architecture for distributed control of a multirobot system”. In: *6th International Conference on Intelligent Autonomous Systems (IAS-6)*. 2000, pp. 115–122.

[12] Xuefeng Dai, Laihao Jiang, and Yan Zhao. “Cooperative exploration based on supervisory control of multi-robot systems”. In: *Applied Intelligence* 45.1 (2016), pp. 18–29.

[13] Haye Lau. "Behavioural approach for multi-robot exploration". In: *Australasian Conference on Robotics and Automation*. Australian Robotics and Automation Association Inc. 2003.

[14] Jose Vazquez and Chris Malcolm. "Distributed multirobot exploration maintaining a mobile network". In: *2004 2nd International IEEE Conference on 'Intelligent Systems'. Proceedings (IEEE Cat. No. 04EX791)*. Vol. 3. IEEE. 2004, pp. 113–118.

[15] Antonio Franchi et al. "A randomized strategy for cooperative robot exploration". In: *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE. 2007, pp. 768–774.

[16] Antonio Franchi et al. "The sensor-based random graph method for cooperative robot exploration". In: *IEEE/ASME Transactions on Mechatronics* 14.2 (2009), pp. 163–175.

[17] Xin Ma, Qin Zhang, and Yibin Li. "Genetic algorithm-based multi-robot cooperative exploration". In: *2007 IEEE International Conference on Control and Automation*. IEEE. 2007, pp. 1018–1023.

[18] A Dominik Haumann, Kim D Listmann, and Volker Willert. "Discoverage: A new paradigm for multi-robot exploration". In: *2010 IEEE International Conference on Robotics and Automation*. IEEE. 2010, pp. 929–934.

[19] David Yanguas-Rojas and Eduardo Mojica-Nava. "Exploration with heterogeneous robots networks for search and rescue". In: *IFAC-PapersOnLine* 50.1 (2017), pp. 7935–7940.

[20] Nancy M Amato and Yan Wu. "A randomized roadmap method for path and manipulation planning". In: *Proceedings of IEEE international conference on robotics and automation*. Vol. 1. IEEE. 1996, pp. 113–120.

[21] Lydia Kavraki, Petr Svestka, and Mark H Overmars. *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*. Vol. 1994. Unknown Publisher, 1994.

[22] Steven M LaValle. "Rapidly-exploring random trees: A new tool for path planning". In: (1998).

[23] Sertac Karaman and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning". In: *The international journal of robotics research* 30.7 (2011), pp. 846–894.

[24] Yang Li et al. "Multi-robot path planning based on the developed RRT* algorithm". In: *Proceedings of the 32nd Chinese Control Conference*. IEEE. 2013, pp. 7049–7053.

[25] Andreas Bircher et al. "Receding horizon" next-best-view" planner for 3d exploration". In: *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2016, pp. 1462–1468.

[26] Hanseob Lee, Dasol Lee, and David Hyunchul Shim. "Receding Horizon-based RRT* Algorithm for a UAV Real-time Path Planner". In: *AIAA Information Systems-AIAA Infotech@ Aerospace*. 2017, p. 0676.

[27] Brian Yamauchi. "A frontier-based approach for autonomous exploration". In: *Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on*. IEEE. 1997, pp. 146–151.

[28] Brian Yamauchi. "Frontier-based exploration using multiple robots". In: *Proceedings of the second international conference on Autonomous agents*. ACM. 1998, pp. 47–53.

[29] Magnus Selin et al. "Efficient Autonomous Exploration Planning of Large-Scale 3-D Environments". In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 1699–1706.

[30] F.S. Barbosa et al. "Guiding autonomous exploration with signal temporal logic". In: *IEEE Robotics and Automation Letters* (2019).

[31] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5.

[32] Nathan Koenig and Andrew Howard. "Design and use paradigms for gazebo, an open-source multi-robot simulator". In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*. Vol. 3. IEEE, pp. 2149–2154.