Autonomously exploring indoor environments with an UAS

Areg Shahbazian
Mick van het Nederend
Wessel Klijnsma

June 30, 2013

Contents

1 Introduction

1.1 National Aerospace Laboratory of The Netherlands

The National Aerospace Laboratory (NLR) is a non-profit organisation in The Netherlands that focusses on research on aerospace technology. It provides scientific knowledge about aviation and space travel to the government, the public sector and the industry.

The research facility was founded in 1919 as RSL in order to speed up the development of aeronautics in The Netherlands. Although having limited resources, a wind tunnel was built to test airplane designs.

Since 1919, the NLR has grown out to be one of the biggest research organisitions in The Netherlands. The NLR now has about 650 employees divided over three locations. Its research covers everything aerospace.

This project assignment was issued by a research group within the NLR that focusses on technologies concerning Unmanned Arial Systems.[**?**] [**?**]

1.2 UAS

An UAS is short for Unmanned Aerial System. An UAS consists of three parts:

1. UAV: Unmanned aerial vehicle
2. DL: Data Link
3. RCS: Remote Control Station

The UAV part is the flying part of the system. There are various forms in which this part can occur. There are helicopters, quadcopter, octocopter, plane models, etc, ranging in size. They can be as small as small as a couple of centimeters or as big as five meters. The choice of the UAV depends on requirements like: the payload weight, speed, maneuverability.

The UAV is controlled from a Remote Control Station. In this station commands are given to the UAV to for instance move to a position.

The link between the RCS and the UAV is called the data link. This can be wireless data links like wifi or bluetooth.

Nowadays, UASs are big bussiness. They are being deployed to do various tasks. The millitary uses them to drop bombs or for the surveyance of hostile areas. Also in the public sector UAVs are no unkowns, one can imagine them being used to film sports events from above or to provide an up-to-date map of an area.

1.3 Rescue workers scenario

Imagine a scenario where there is a building on fire. The outside (bounds) of the building is known because it can be seen from the street. The inside of the building is unknown because it can't be seen from the street and going inside would be to dangerous because of the fire.

As a fireman, there is the constant dillema of risking your own live or saving another. Obviously, firemen are trained into dealing with these dangerous situations, but that doesn't mean that things never go wrong. It was reported that in the United States 82 firefighters died in their line of duty. [1]

In order to create saver working conditions for firemen and other rescue workers, UASs can be used. They can be sent into burning or collapsing building and provide rescue workers with valuable information of the environment.

Another sitution where UASs can be deployed to contribute to safety is in cases where the inside of a building changes frequently. An example of such a situation is a conference hall. For every conference there is a new floorplan. Making these floorplans manually takes a lot of time. Letting a UAV fly accross the conference hall creating a 3d model, saving the time it takes to do this manually. Also 3d models will prove more usefull to rescue workers in case of an emergancy than a 2d floorplan, because of the extra information a 3d model has.

1.4 SLAM

In order to map unknown areas, an agent, in this case the UAS, needs to know its position and have some data of this position to map to it. This data can come from for instance RBGD (RGB + Depth) cameras or laserrange sensors. It provides the visual information needed to create the map.

SLAM is short for Simulatanous Localisation And Mapping. The thesis by NLR intern Robbert [3] Proost describes a method in which a Pelican is used in combination with a Kinect to do SLAM. The RGBD data from the Kinect sensor is mapped to to location which is determined by a visual odometric algorithm. Visual odometry uses visual input to calculate the position by extracting and analysing features from the images provided by the camera.

1.5 Autonomous exploration

The method developed by Proost requires manual control over the UAV. Controlling a UAV like the Pelican without crashing it, requires a lot of experience and skill of flying with such vehicles.

This means that in order use SLAM in combination with a UAV in rescue situations, trained people with this experience and skill are needed, and need to be paid. Autonomous control of the UAV would mean that less of these people are needed, making it easier for rescue services to start using UASs in their work.

This project focusses on researching and implementing methods which can be used to autonomously explore unkown areas using an UAV. As these methods will differ depending on the situation in which they will be used, this project aims to develop a method which works in at least one of these situations.

[1] http://apps.usfa.fema.gov/firefighter-fatalities/fatalityData/incidentDataReport?idrYearStart=2012idrYearEnd=2012

The situation on which this project focusses, is the one in which UAV is used to create a 3d floorplan of a conference hall. This means that bounds, the outside walls, of the conference hall are known and that some assumptions on what to expect in the unknown area between the bounds can be made.

1.6 Product vision

For researchers who want to use a mini UAS to autonomously explore partially known, like conference halls, we document about and implement the possibilities and new ways of pathfinding and exploration strategies which minimize occlusions in the generated 3d maps. Unlike the current systems which have to be controlled manually, our product will require no human controlling.

2 Product overview

2.1 Harris' decomposition

The problem described above is a well-known one within robotics. Harris [4] developed a handy way to decompose this problem into three smaller ones.

1. World and knowledge base representation
2. Path-planning
3. Exploration

The first item implies that a clever method needs to be found to create the internal representation of the map. In this case, this is done using *octrees*, described below. The mapping of the area, the knowledge base representation and data collecting and storage, will be done by Proost's software.

Then there's the path-planning: a way of going from A to B, the point B being being decided by the exploration strategy. It is important that the agent avoids obstacles and re-plans its path when it encounters one.

Finally, a global exploration strategy is required. The NLR required this aspect to be parameterizable, as different uses require different measures. A tradeoff between speed and thoroughness is inevitably to be made, and efficiency should always be high.

2.2 World representation

The physical world is observed with a RGBD-camera with a limited range. A sensor like this can observe surfaces and represent them as point-clouds. A point cloud is a data structure of 3D-points in a space. This data structure can be converted into a *octree* data-structure.

The *octree* data-structure can best be explained as a nested 3D-grid. The world is divided into big, equally big cubes, which in turn are divided into eight big cubes themselves, all of which are again divided into eight big cubes. This continues for as long as the level of preciseness requires. This data-structure is one that distinguishes between three different states that a block can be in. Some places in the world can be *occupied*, meaning there is some rigid volume in that area. Other places can be known to be free, meaning there is no rigid matter. Finally, some areas are represented as unknown, meaning there is no explicit information acquired by a sensor about that area.

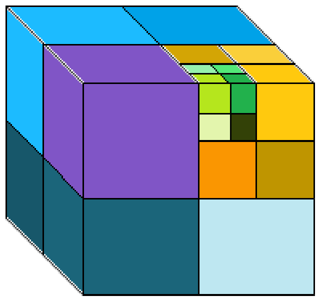

Figure 1: The structure of octrees

Since the three characteristics of a volume described above provide enough information for navigation tasks, the *octree* data-structure is a convenient way to represent the world. The data can be stored in different resolutions, resulting in different scales of accuracy of the world representation.

In our simulation of the problem we've represented both the world (the environment) and the knowledge base in octrees. Specifically, we used the *octree* implementation described in [8]. A C++ library called *Octomap* [2] is provided to read, write and use the data-structure. The library is open source and available. It also contains the *octree*-visualization software *Octovis*. Installation instructions are given on the website and are fairly straightforward.
Our software was written and tested in a Linux environment (Ubuntu 12.04 LTS).
The main framework of our program works as follows. It represents a UAS-agent as a 3d-coordinate with a depth sensor. The $OcTree$-data representing the complete world (nothing left out) is loaded from a command-line argument. From this map the nodes that should be visible from the agent-location are found, using a specified sensor-range, and stored in a second $OcTree$, representing the explored world. This is done every time the agent is relocated. In this way a moving agent with a sensor is simulated and the $OcTree$ data can be easily visualized in $Octovis$.
Finding the $OcTree$-nodes that should be visible from the current position is done using the $rayCast$ function provided in the library. This function receives a starting point, a direction-vector and a range as arguments and calculates whether there is an occupied node in that direction, within the range. This is

[2]http://octomap.github.io/

done in a spherical form around the agent. It thus maps all occupied nodes within the specified range from the agent that are not occluded by other occupied nodes. The nodes that have been traveled through by the casted ray are stored in the map as free nodes. All other nodes that have not been casted a ray through remain unknown.
This framework is a simplistic approach that uses neither real-world data nor a real UAS agent or a real sensor. It is generally implemented in a high-complexity manner and simply provides minimalistic environment to simulate and test different algorithms. It has not been the aim of the writers to develop a real-time simulation or visualization environment.

2.3 Path-planning

Because this project mainly focussed on developing an exploration strategy, we created a fairly simple path planning algorithm to get from A to B. The most important aspects are that it gets the agent to B, and it gets it there safely. Safely in this context means that the agent will never be too close to a rigid body. How 'too close' then in its turn is defined, depends on the size of the agent, the sensor range and the resolution of the octree, but it should be fairly clear when you realize that it's mostly a safety measure for measuring errors.

First, the direction towards B is computed. If it's possible to take a step into that direction without getting close to an obstacle it will do so. If not, it will turn until a direction is found in which a safe step is possible. This process is repeated until B is reached. If a bound (outer wall) is reached, the agent will go back and search in the other turning direction.

The initial direction in which the agent turns is the one that is closest to the center of the map, as we felt that this would yield better results.

Of course this is in no way a proper, failsafe algorithm. The proper alternatives are discussed in 'room for improvement', below. Still, it worked surprisingly well.

2.4 Exploration strategy

The exploration algorithm we've developed for this problem needs to be very adjustable, so other developers could use and alter it without too much work. In addition, it has to be very parametrizable, so the user can change the input to match the problem.

The algorithm is a hierarchical one, existing of two layers. The top layer chooses an optimal *goalposition*, a place where the most valuable information is gained upon reaching it. While traveling there, the bottom layer keeps checking if it's worth it to diverge from the current path to a *subgoalposition* nearby the agent's position, to obtain extra information in a relatively cheap way.

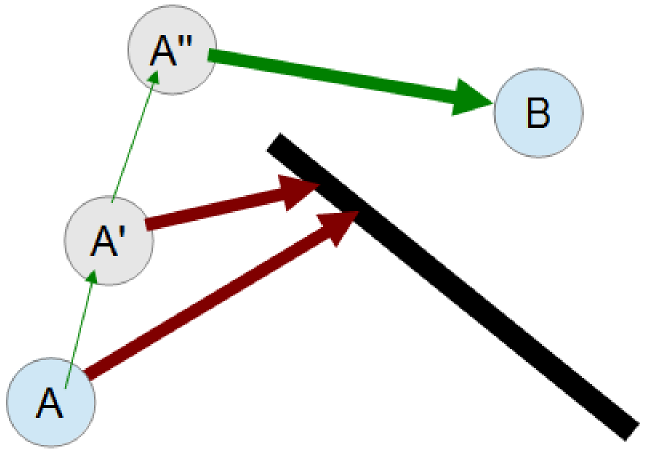

Figure 2: Path planning: The agent starts at position A and sees it can't travel into the direction of B, as it will get too close to a wall. It turns until a clear path is found: A'. From there it still can't reach B directly, so again it turns until a clear path is found: A". From here it can directly go to B, and it will.

2.4.1 Top layer

The goal position is an octree somewhere within the bounds of the room which is regarded as the most valuable place to be at that point in time, in terms of the score of the unexplored area surrounding that point. The score is computed by three factors and is named the $goalscore$.

- Amount of information gained
- Uniqueness of information gained
- Relevancy of information gained

Amount of information

The first item, the amount of information gained, tells us how much of the area surrounding the node still is unexplored. We've chosen this factor, as a point that scores high here will logically provide more information than another point that doesn't. And because part of the objective is to explore as much space as possible, this is relevant.

In our implementation we've iterated through the entire octree. However, instead of checking all the nodes, and performing an operation on all of them, only one thousandth of all the nodes was checked. All of them harbor an equal distance of ten nodes until the following one in any dimension. This is done in order to reduce the complexity.

Because it would also be too complex to simply count how many unexplored nodes would be discovered from that point, an alternative estimation has to be made. In six directions (one in every single dimension incrementing positively, and their decrementing counterparts) it is checked how much distance can be covered until a discovered part or bound is reached. Then, the sum of these distances if obtained for each pair, so for each dimension. These three numbers will then be multiplied with each other for the final 'Amount of information' score.

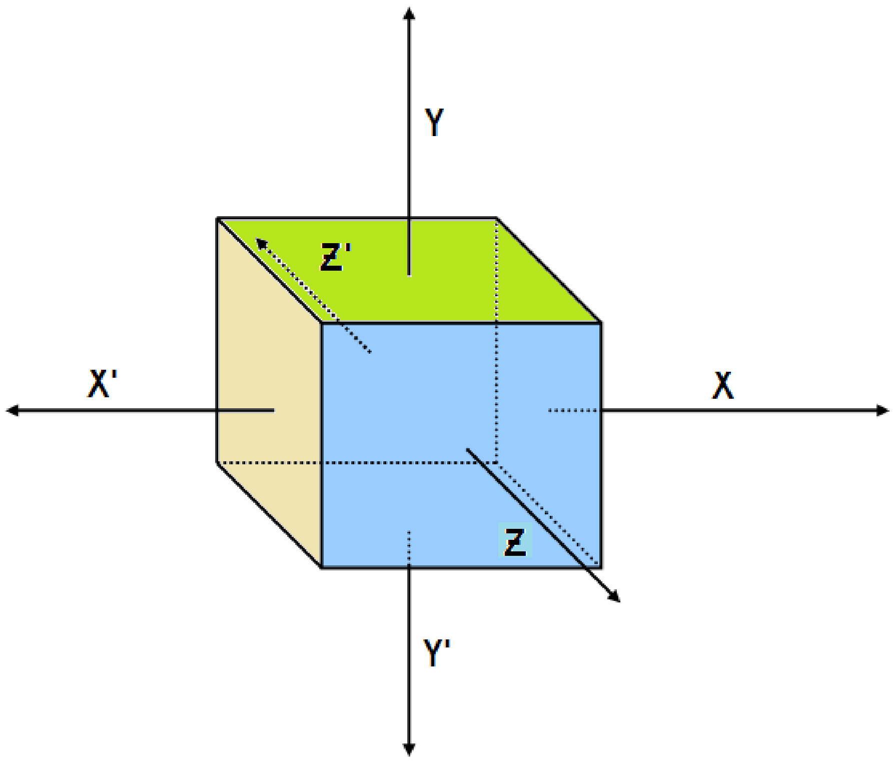

Figure 3: Amount of information gain $A = (x + x') + (y + y') + (z + z')$

Uniqueness of information

Secondly, the uniqueness of the information. This is an indicator about how 'special' this information is. In practice: from how many other possible goal positions can this information be retrieved? This is valuable, because nodes that are exploreable from quite a few other positions have a higher chance of being explored along the way anyway.

To calculate the uniqueness value of a potential *goalpositionG* in an ideal world, the following elements should be computed:

1. *count*(G), Amount of nodes n discovered from G (amount of information

gained)

2. For each of those nodes n_i, the amount of *goalpositions* besides G that would explore it: $score(n_i)$

Then the following formula can be used:

$$\frac{\sum_{i=0}^{count(G)} \frac{1}{score(n_i)}}{count(G)} \tag{1}$$

From a potential goal position G, the uniqueness score is the average of the inverse of its node scores.

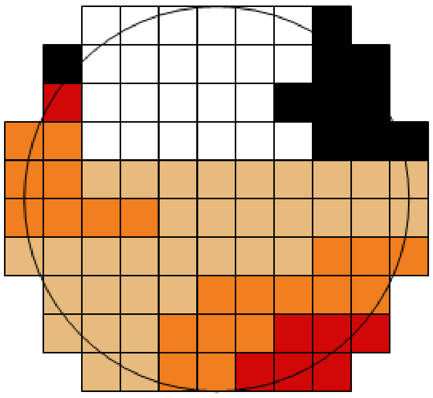

Figure 4: Uniqueness example. Every node has it's value:
- Red: reached by < 5 other potential goal positions
- Orange: reached by $6 - 10$ other potential goal positions
- Brown: reached by > 10 other potential goal positions
- White: already explored
- Black: unreachable

The uniqueness value then is: $\frac{\frac{1}{2} + \frac{1}{7} + ... + \frac{1}{1}}{61}$

Unfortunately, this method is too complex to use in reality. We couldn't implement it for that reason, as the compilation time became too high. We couldn't come up with a simplification of this computation either, so we had to leave it out in the final implementation.

2.4.2 Relevancy

Altough not implemented during this project, but of great importance to exploration algorithm is the relevancy of information.

When an UAS is searching for a next location to navigate to, it might be the case that multiple locations have similar values for information gain, meaning the locations are equally unknown. In that case priorities should be set on what kind of places are more important to explore. When dealing with different situations regarding the purpose of the exploration, the type of area to be mapped and the available prior knowledge, it might be handy to have parameters which could be adjusted to meet the requirements of the user.
For future improvements of the described framework we will discuss several strategies to categorize different viewpoints and we will try to emphasize the importance of each strategy in different situations and parameterize the properties of these strategies.

Different situations

Areas

To describe different situations two types of indoor areas will be distinguished. The first type is a big open space with little to no subspaces (closed spaces within the main space). An example would be an empty warehouse or a large hall. The second type of area is the opposite of the first, so one with lots of subspaces, for example an office/apartment building or a school.

Purposes

The purpose of the exploration of an unknown area can also differ. One can have the goal to simply map the inside of a building in order to furnish it or make efficient use of the available space. For example mapping the inside of a warehouse to decide how to store the goods in an efficient way. Another exploring purpose could be search and rescue. When for example an office building has caught fire and the purpose of the exploration is to provide information for rescuing forces to safely enter and leave different areas of the building. Our strategies will mainly focus on the search and rescue purpose of exploration.

Prior knowledge

In most situations the dimensions of a building will be known. Most indoor areas have blueprints available and clear borders can be set to the total space to be explored. However, in some situations this information could be unavailable. For example, when exploring an underground structure for military purposes, it could be necessary to map the outside contour of the structure.
When general information about the building is available one could focus the exploration on finding paths between different places inside the building and the available entrances/exits. Also the need for finding out which (emergency) exits are available and which are blocked from inside could make the exploration task somewhat goal-based.

Strategy parameters

Height

One obvious parameter that can influence the relevance of the gained information is the height at which the UAS is traveling. When the goal of the exploration is making a complete map of the area, the height is only bounded by the height of the area, meaning the height of the ceiling.
In the case of search and rescue, the main purpose of the area-mapping would be to provide navigating information for ground-agents (human or ground robot with more battery capacity than UAS). In this case there is little need for information about the area above a certain height. Of course the desired height to map from can vary within the same area at places where the ground elevates or lowers, for example above stairs. It could be convenient to provide the UAS with a fixed height value which is then, if possible, reached by measuring the height of the ground directly underneath it at the current location and adjusting the flying height.

Accessibility

When performing a search and rescue exploration task in an area with lots of subspaces, the user might want to set a higher priority to mapping ways between the subspaces than mapping the inside of a subspace. This strategy can be intuitively justified by stating that a rescue agent, once inside a subspace, can manually "map" the inside of it and that it would rather have ready information about how to get to another subspace (Figure 5). Of course a distinction between subspaces should be made first. This can be done by using mapped vertical surfaces above a certain height that act as borders between subspaces. The accessibility of a given viewpoint could for example be expressed by the amount of rigid surfaces that stand in a straight line between the current position and a calculated next viewpoint. This value could be used as a parameter and adjusted by the user, depending on the capabilities of the rescue-agent.

Redundancy

Indoor search and rescue operations are sometimes performed in situations where the environment changes dynamically. For example, in a building that has been exposed to a big fire or an earthquake, it could happen that large objects collapse and fall down on pathways that were mapped by the UAS and considered accessible. This calls for an exploration strategy which guarantees multiple ways to and from every known location. This type of redundancy can be implemented as a parameter that represents a maximum distance. As soon as this distance between the known areas and the UAS has been reached, it starts to look for an alternative way back to a known area (Figure 6). This narrows down the choices of next viewpoints, since the ones too far away will not be explored first. A minimum distance between the forward and the back-

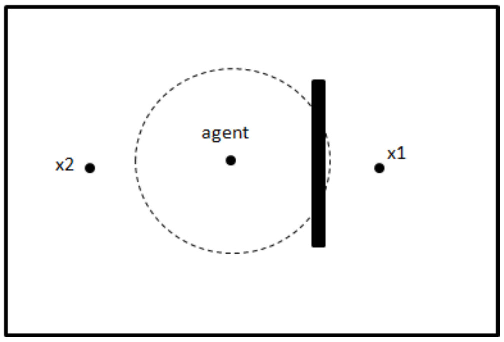

Figure 5: The probability of the x points being visible from the agent position is: $P(x_1) = 0$ and $P(x_2) \geq 0$. Therefore it is more relevant for the UAS to explore x_1

ward paths should be retained to prevent both paths from being blocked after a single collapsing event.
The parameter described above can also be used to represent the maximum distance to travel before looking for an alternative way back to an exit point. The location of such a point can be attained from the explorations or provided beforehand if blueprints or similar information about the building are available.

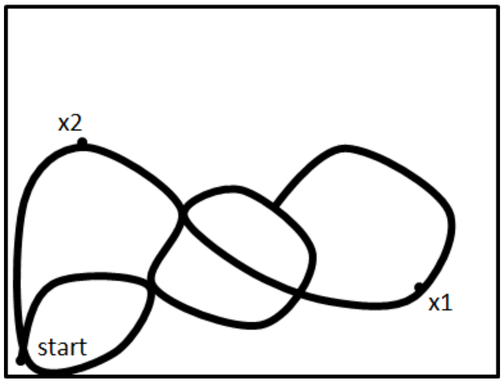

Figure 6: Making redundant backways to a known position may result in circle-like shapes of the explored path, with the maximum-distance parameter as diameter

2.4.3 Bottom layer

As stated before, the bottom layer in the hierarchy looks whether or not it's worth it to diverge from the current path. The current path leads to the goal position, which is computed by the top layer. The point to which might be diverged, is called a *subgoalposition*. The user will provide the agent with a double, the *distractionvalue*. This number will decide how quickly the agent is 'distracted' into subgoal positions.

The area in which is searched for a subgoal position is a subspace around the agent. In our implementation this exists out of two directions perpendicular to the current direction of the path, and perpendicular each other. Along those directions, and their negative counterparts, a handful of points is checked for their information gain, uniqueness and relevancy score. The point with the highest combined score will be the potential subgoal position, with its subgoal score. The agent will actually travel towards the subgoal if and only if $subgoalscore * distractionvalue > goalscore$.

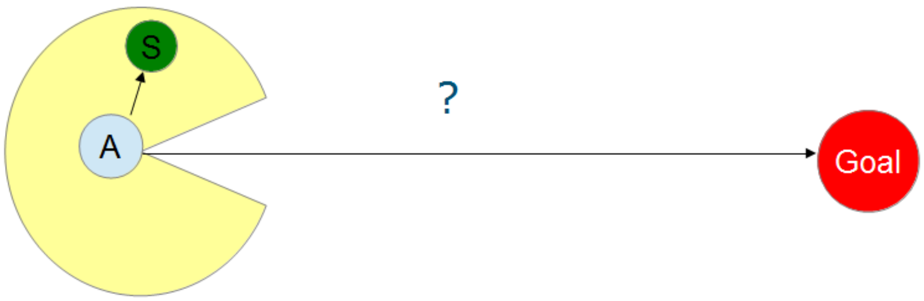

Figure 7: The bottom layer of the hierarchy looks for potential subgoals in transit to its destination. Whether or not it will go to the subgoal, depends on the ratio between the product of the subgoal score and the distraction value, and the goal score. (A is the current position, S is the subgoal and the yellow area is the searchspace for the subgoal.

The hierarchical structure, and the distraction value were suggested by our academic mentor, Leo Dorst. The strength of it, is that it mainly focusses on unique and relevant places, as the amount of information gained will not be very high most of the time (as the area directly around the agent is mostly explored). The advantage of this is, that this kind of information is typically difficult to obtain from a random place in the room. Therefore taking a small detour can differ a lot in the end.

2.4.4 Pseudo code

Algorithm 1 Exploration strategy

```
$distractionValue = 5$
$goalScore = 0$
**for all** goalPosition as g **do**
  $score = g.informationGain * g.uniqueness * g.relevancy$
  **if** $score > goalscore$ **then**
    $goalScore = score$
    $goalPosition = g$
  **end if**
**end for**
**while not** at g **do**
  $s = currentPos.getBestSubGoal$
  $subScore = s.informationGain * s.uniqueness * s.relevancy$
  **if** $subScore * distractionValue < goalScore$ **then**
    $s.travel$
  **else**
    $planNextNode(currentGoal)$
  **end if**
**end while**
```

3 Future improvements

3.1 Path-planning

The path-planning algorithm used in this implementation only works in very basic environments without any subspaces. In order to be able to find a path in more advanced environments, one could use a more sophisticated path-planning. One of these path-finding algorithms is D*.

3.2 Uniqueness of information

An implementation which can determine the uniqueness of information will lead to a more efficient algorithm.

3.3 Relevancy of information

As mentioned in the previous section, the relevancy of information was not included in the implentation. By including this in future implemenations, the exploration algorithm will be more adjustable to the needs that arise in different situations.

Also, different parameters of relevancy can be thought of. In many cases the height parameter will sufficiently express the relevancy but exceptions can be thought of. Ladders and ropes present in the area will not be mapped in this way.
Using an accessibility parameter that divides a space into subspaces using ver-

tical obstacle-surfaces will bring with it the risk of seeing accessible obstacle (stairs or climbable fences) as not accessible. Finally, a redundant strategy that makes the map more robust will also require more time and thus battery power. This will have a negative effect on the efficiency of the mapping.

3.4 Performance

The writers of this report and of the software code had no prior experience with the $OcTree$ data-structure or with the C++ language. The goal of the software is conceptual, meaning it implements a framework for addressing the indoor exploration problem. The implementations of the functions used have a high complexity (often $O(n)$) and require much runtime. In future developments, where this kind of framework is linked with real world sensors or actuators or used to make real-time simulations, the performance issue could be addressed by computer-programmers with more experience.

4 Conclusion

4.1 Metrics

In order to evaluate the performance of our exploration-algorithm we used one main metric that could express its efficiency. Since the agent is relocated step by step, using a fixed step-size, the number of steps and thus the distance covered can be used to rate the number of nodes discovered. The metric unit used to evaluate the performance is thus the number of discovered nodes per step.

Desired results would be high values of mapped *octree*-nodes per step, since in real life situations time is valuable. By varying the distraction-value different ratios were measured.

Besides the information gain, other metrics that could be used to determine the efficiency of the exploration-algorithm like:

- Time spend
- Distance covered

Also tweaking different parameters can be used to see the effect on the metrics. Examples of these parameters are:

- Sensor range
- The world map, the performance of the exploration algorithm can differ depending the environment
- Relevancy, different useages of the system require different kinds of information. Whereas rescue workers would only be interested in a global overview map of an environment, military forces may require very detailed information.

4.2 Results

As can be seen in Figure 8, $the distraction value$ seems to have an influence on the information gain. Within the range of the $distraction value$ between $1 - 5$ the optimum lies at values > 3. This means tweaking the distraction value is important assure efficiency of the algorithm.

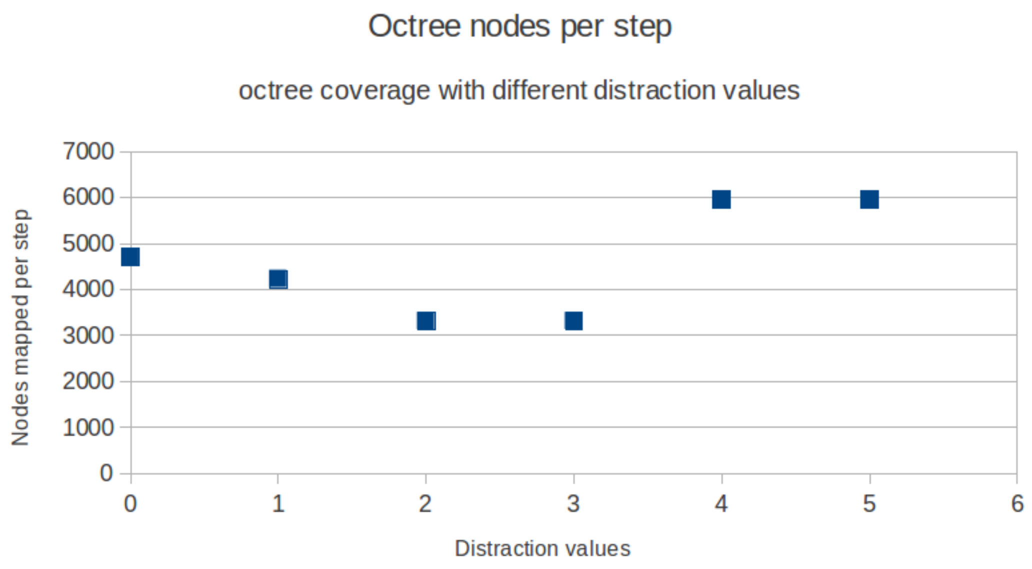

Figure 8: Nodes discovered per distraction value

4.3 Summary of developed product

During this project one method of autonomous exloration was implemented. This implementation works in basic room without much subspaces. During the implentation insight was gathered on concepts that can influence the performance of exploration algorithms in different situations.

Also a platform was created on which future implementations of exploration algorithms can be tested and benchmarked.

Thanks
We would like following people that have helped us during the project and have made this project possible:

- Gerald Poppinga, our client at the NLR
- Leo Dorst, our academic mentor from the UvA
- Robbert Proost, intern at the NLR
- Raquel Fernandez, coordinator of the project

5 References

References

[1] The National Aerospace Laboratory website: www.nlr.nl

[2] The wikipedia entry for NLR: http://en.wikipedia.org/wiki/National_Aerospace_Laboratory

[3] Proost, R. (2013). Simultaneous Localisation and Mapping in 3D environments using a mini-UAS

[4] Harris, C., Hants, R. (2007). Strategies for Visual Exploration of Buildings

[5] Stenz, A. (1994). Optimal and Efficient Path Planning for Partially-Known Environments

[6] Surmann, H., Andreas, N., Herzberg, J. (2003) An autonomous mobile robot with a 3D laser range finder for 3D exploration and digitalization of indoor environments

[7] Joho, D., Stachniss, C., Pfaff, P., Burgard, W. (2007) Autonomous Exploration for 3D Map Learning

[8] Hornung, A. et al (2012) OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees