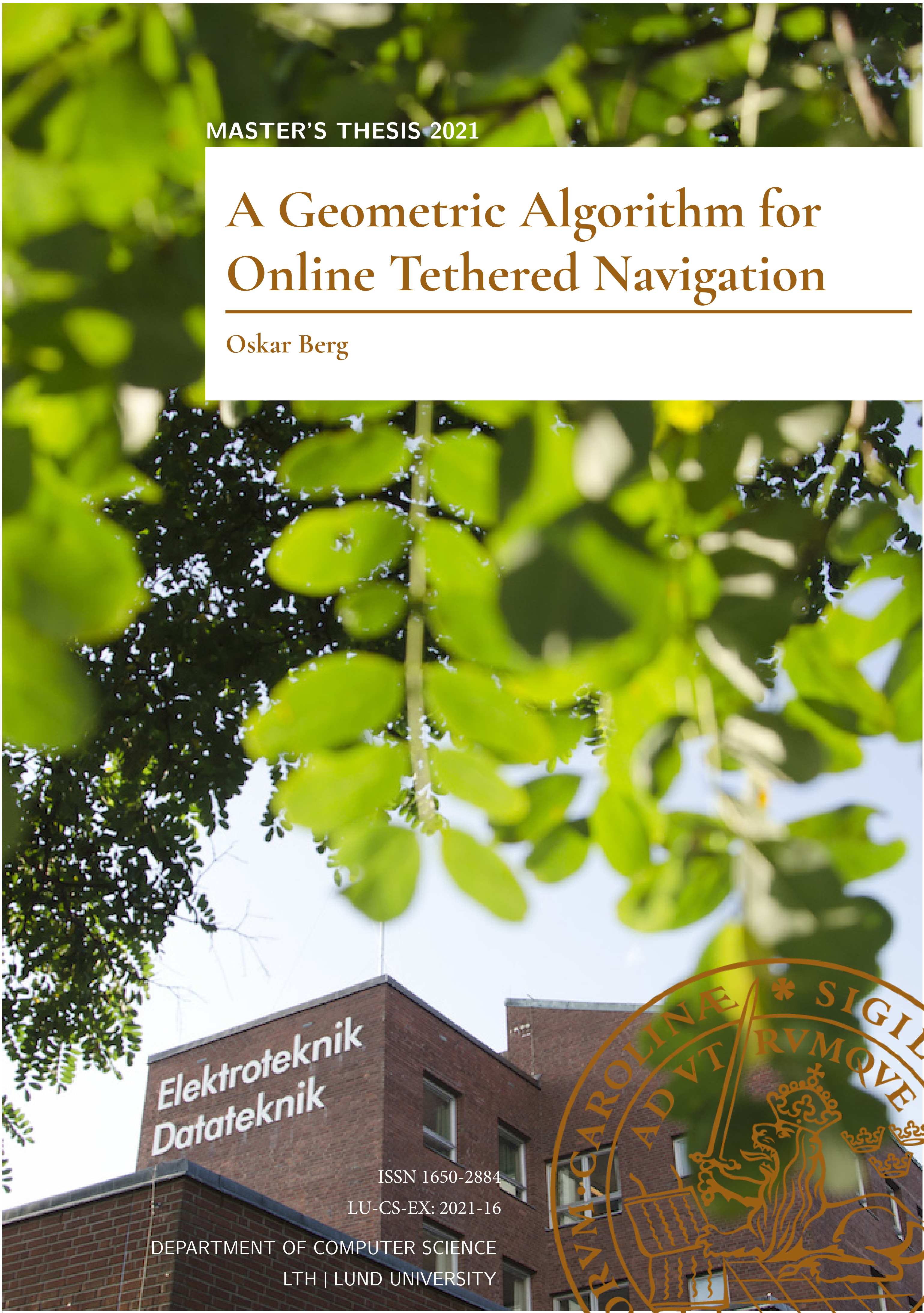
MASTER'S THESIS 2021

A Geometric Algorithm for Online Tethered Navigation

Oskar Berg



ISSN 1650-2884
LU-CS-EX: 2021-16

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2021-16

A Geometric Algorithm for Online Tethered Navigation

En Geometrisk Algoritm för Trådbunden Realtids-navigering

Oskar Berg

A Geometric Algorithm for Online Tethered Navigation

Oskar Berg
dat15obe@cs.lth.se

February 1, 2021

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisor: Elin Anna Topp,Elin_Anna.Topp@cs.lth.se

Examiner: Jacek Malec, Jacek.Malec@cs.lth.se

Abstract

Some robotic applications require a permanent connection to either electricity or a communications base. This can be done by using an attached tether to the robot. This tether adds additional constraints and complexity to the requirements of the navigation of the robot. There exist algorithms for achieving this type of navigation but when implementing these algorithm, several shortcomings are discovered. By implementing my own version, I improve on the existing algorithms and achieve a more accurate coverage of the target area. The new solution I implemented was very efficient but had other shortcomings, such as not being able to cover all corners accurately but deemed as an improvement over the existing algorithm.

Keywords: MSc, Online, Thethered, Navigation, Motion Paths, Wired, Robot, Motion Planning

Acknowledgements

This project was conducted at the Department of Computer Science, Lund University. I wish to thank Elin Anna Topp for her invaluable assistance as my supervisor and Jacob Canbäck for the use of his graphics library.

Contents

Chapter 1
Introduction

While the main trend for electronics is to have more and more battery powered devices, there still exist devices which are not suited for battery power. Among these devices are the units that consume too much power to be able to be powered by battery, devices that require some external material as well as devices that cannot communicate properly without tethered communication such as underwater devices.

A modern solution to this would be more intelligent navigation in order to allow for robots to use tethers. For the subject of this thesis I will focus on navigation to cover a target area. A typical application would be a device that cleans a floor area. A tethered floor cleaner would not be limited by battery power and would be able to use a more powerful cleaning motor.

The problems of using a tether include the issue of not colliding with your own tether, not creating too sharp an angle from the robot to the tether and obtaining an efficient area coverage.

1.1 Research Goal

My goal is to perform a validation of the claims made in the Shnaps and Rimon's article[13] and then improve upon their algorithm. In doing this I want to answer whether their algorithm works, produces useable motion paths and whether it produces an efficient solution to cover a room while attached by a tether? And finally will I be able to make improvements upon their solution?

1.2 Methodology

The first part of my thesis is based upon Shnaps and Rimon's article on online tethered coverage[13] and I started by attempting to verify the algorithm and conclusions in their article. The original approach was to implement their algorithm and then to improve their algorithm to be able to be applied to a more general set of robots.

However, since the result of my verification of Shnaps and Rimon's article was unsatisfactory, I tried to focus more on constructing an alternative algorithm which will be an improvement of their algorithm.

Just as Shnaps and Rimon did in their article, I assume the tether will be under tension and therefore create straight lines between corners. Likewise I also assume that the robot will have a mechanism that will retract the tether when given slack and extend when necessary. The retraction mechanism is assumed to be circular so the tether can extend and retract in any direction from the robot.

1.3 Contributions

This master's thesis contributes by:

- Performing an evaluation of Shnaps and Rimon's algorithm.
- Developing a new algorithm for tethered robots.

Chapter 2
Background

The focus of this thesis will be on covering a space, i.e. traversing all the area in a planar environment. This is very similar to what is done by modern autonomous vacuum cleaner robots. In paper[7] different algorithms are explored for covering a planar environment as fast as possible using an autonomous vacuum cleaner. According to the findings in paper[7] the most optimal route is not produces by a single algorithm but a combination of several.

In the subject of tethered navigation there exists several different approaches to handle the constraints the tether provides. It is described and suggested in the article by Shnaps and Rimon[13] a solution to the same issue I have investigated. In the paper a grid-based approach is used to cover an area with a robot that has an attached tether. The grid's cells that need to be covered are stored in a list. The list is ordered by the cells' distance to the tether origin and then the robot motion is planned from the cells in the list. Split cells are also created, which corresponds to my dividers which will be explained below. The split cells will make sure that the entire area on one side of an obstacle is covered before moving on to cover the other side of that obstacle.

A interesting solution to the tethered problem is presented in paper[9] where a slacked tether is used instead of one under tension. This makes it difficult to know where the tether is located but does not require the robot to have a tensioning device. Instead the tether is avoided by making a model of how the tether will behave when moved, and the robot can then react to its predicted position in order to not run over the tether or get tangled. This is very difficult to simulate so I opted to not use a slacked tether.

In my thesis I will use general pathfinding. One of the articles I based my own pathfinding function on is in article[5] where among other things, A* pathfinding is explained. By using the A* algorithm in paper[5] in combination with the algorithm presented in the article[14], I managed to create a pathfinder for flat polygonal environments.

In [15] an algorithm is presented to find the shortest path for a tethered robot between two points in a planar environment. This is done by first untangling the tether by reversing the robot from the current position until it can reach the other point.

Similarly in paper [4] an algorithm is explored for pathfinding with a tether. However, the possibility of the robot only being able to retract the tether when reversing towards the tether is also considered.

Shortest paths when connected with a tether is also investigated in paper[12]. However, the execution time was shortened by optimizing the algorithm by preprocessing the environment.

In the area of tethered robotics there exists several interesting applications. As an example, in paper[2] it is proposed to use a tethered robot on the surface of Mars. In the paper it is explored the possibility of using a smaller robot which is using a tether attached to a bigger robot rover, in order to ascend and descend steep terrain. In order to achieve this safely a geometric model was created of how the tether will behave when extended across the uneven terrain of the slopes.

Another common application of tethered robotics are underwater rovers. Since water blocks almost all forms of communication, there are no other options for controlling robots deep under water than using some sort of tether. An example of such a rover is the Ocean One rover[8]. It is a human-like robot which is tethered to a large base unit, where the smaller robot can move almost freely in order do explore underwater environments.

Some robots uses a wire as a fail safe[3], e.g. when inspecting pipes or air ducts, it can be difficult to navigate properly around those narrow spaces and in case the robot gets stuck, it can always be extracted with its attached wire.

In my thesis I used the ROS[1] library as inspiration for many components in my environment. The ROS library is an open source library and platform for robotics. In ROS, GMapping is used as one of the options for SLAM (simultaneous localization and mapping) where it makes use of a Gaussian distribution of particles to estimate the position of the robot from the scan data. In my implementation I never implemented a full SLAM system, but used algorithms found in papers on the ROS homepage for inspiration to my simplified version.

Another basis for SLAM is scan matching which is explained in [11]. In the article it is explained how to compute the likelihood that a scan was made from all nearby positions close to the robot. From the those calculation it is then possible to determine the location and orientation of the robot.

A more modern example is the use of robots for charging other robots. The electric vehicle company Tesla presented a prototype charger capable of finding and connecting to one of their vehicles. This robot is intended to automatically charge a car without user interaction. Tesla have not disclosed how the robot will carry energy to the car but one solution is probably to let the robot navigate with an attached power cord. Similar concepts of robotic car charging have been researched before with the robots carrying small batteries[10]. This application could also be improved with tethered navigation since the robots could use power cords instead of batteries.

Chapter 3
Approach

3.1 Algorithm

When designing my algorithm, I based it upon Shnaps and Rimon's algorithm[13]. In order to achieve an efficient area coverage I want the robot not to unnecessarily cover the same area several times. Therefore I want the motion path to first prioritize the areas closest to the origin point and then move on to increasingly more distant areas. However when an obstacle is encountered, the motion path should cover all areas on one side of the obstacle before it moves on to the other side of the obstacle. Otherwise the robot would have to move back and forth around the obstacle many times.

The algorithm I created is a combination of a function for finding points of a given distance from a center point and an algorithm to partition these distances in a hierarchical tree structure.

To find all the points at a given distance from the origin point, we begin by choosing a starting point. If there is no obstacle nearby the start, the boundary is a circle. See Figure 3.1.

If an obstacle is close, we need to calculate how the boundary expands beyond it. This is done by finding all convex corners within range and if the convex corner has an occluded side we create a new boundary from this corner with the range decreased by the distance from the origin. This corner is then a node. This new boundary is not circular, instead it only ranges from the occluded side of the convex corner to the opposite direction to the origin. See Figure 3.2. This new boundary is then expanded recursively until no other points are inside the range of the boundary. See Figure 3.3.

To construct the boundary and nodes into a tree structure we need to use dividers, where the tree structure will be split. To explain a divider we can imagine a graph, see Figure 3.6, of the distance to the wall from the current point (a node) as function of its angle. Then we can find the local minima in this graph. These are the points we will use as dividers. These points are points where the robot will have to come closer to the current point before continuing on the next boundary, see Figure 3.4 and 3.5.

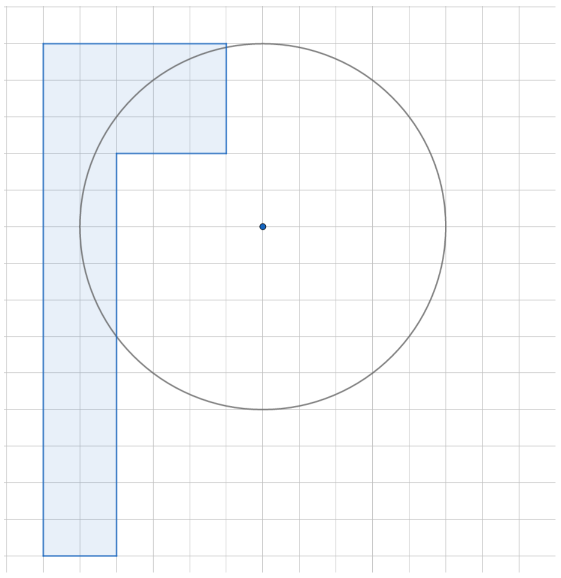


Figure 3.1: When no outside edges of obstacles have been encountered, the boundary is circular.

In order to find the dividers we need to translate the map to a polygonal representation where all walls and obstacles are described as polygons. We can then get all the points which are candidates to be dividers. For each node in the boundary, all divider candidates have to be checked whether they are a divider to that node. Each line in the polygonal representation has a divider candidate which is the point closest to the node. In some cases these are the end points of the lines and sometimes they are the orthogonal projection of the node on the line. The candidates can then be refined by excluding the points which are occluded. These points are dividers that branch the tree structure in order of distance to the node.

One of the main difficulties in implementing this algorithm is the management of numerical errors. The algorithm uses both a discrete and continues coordinate system, i.e. the robot is positioned in a continues coordinate system while the map is stored in a discrete system. This creates several situations where rounding errors causes wrong results if we do not include these situations as exceptions in the implementation. Let us look at a few of these situations.

Such a situation arises when checking whether a point can be observed from another point on the map and one of the points is located on a corner of an obstacle. If we just use the default way of finding occlusions the corner may occlude itself. We therefore have to remove the closest cells to the corner in the occlusion algorithm.

Another situation occurs when checking whether a point is inside a circle sector. Several cases occurs where the point will be situated perfectly on the edge of the arc and depending on rounding errors it will sometimes be considered inside and sometimes not.

3.2 Environment

When implementing the project I decided to not use existing tools such as ROS[1] or other robotics libraries because I am more familiar with the Java language and my own already developed libraries were deemed to be sufficient to implement the proposed project. To

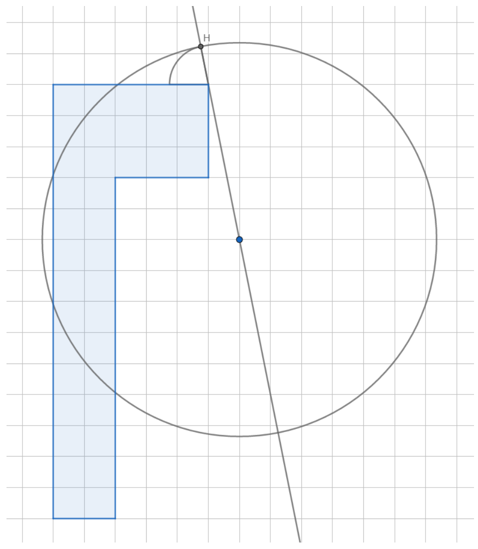


Figure 3.2: When an outside edge is encountered a new circular segment is produced to continue the boundary.

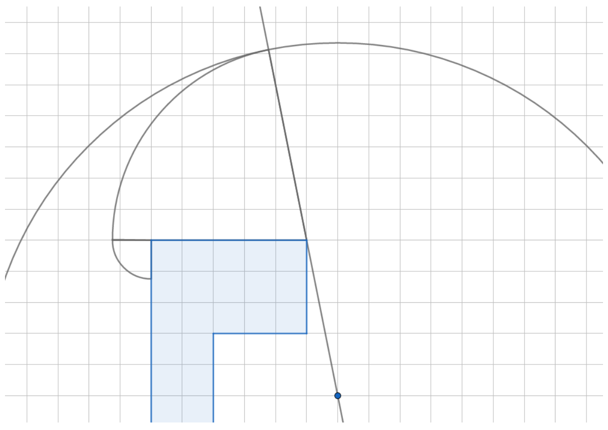


Figure 3.3: This is then done recursively to find the entire boundary.

have better insight into the internals of the algorithms and supporting functions, I decided to use my own implementations. As an example, a shortest path algorithm in a polygonal environment uses functions to extract convex corners and tethered navigation is dependent on using convex corners. By implementing my own version of a shortest path algorithm enables me to reuse the part of the algorithm that extracts convex corners with a much higher degree of understanding than if I had used existing tools.

For the implementation, I used Java and a graphics library supplied by Jacob Canbäck.

The simulation consists of a 3D environment, see Figure 3.7, and a robot able to view the environment using a 2D lidar. The lidar was used to create a map of the environment.

I did not implement any functions in order to see if the tether imposed a problem. Instead I simulated the tether and could visually verify if this tether would impose any issues in the real world and in that way verify my algorithm.

One shortcoming of my environment was the lack of a proper SLAM (Simultaneous localization and mapping) implementation. I implemented the mapping part and a few basics of the localization using the algorithm found in [11]. I decided to use the exact location of

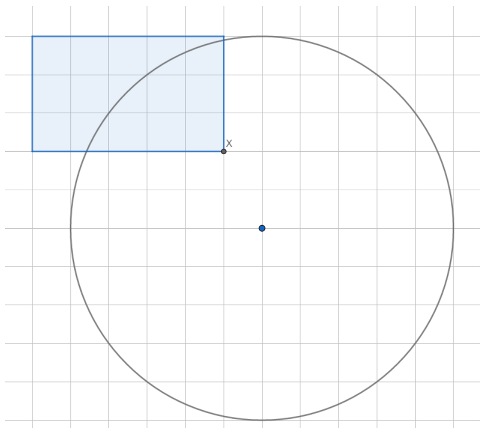


Figure 3.4: The simplest case of a divider. The corner X prohibits the robot from moving from one side of the rectangle to the other without moving closer to the center point.

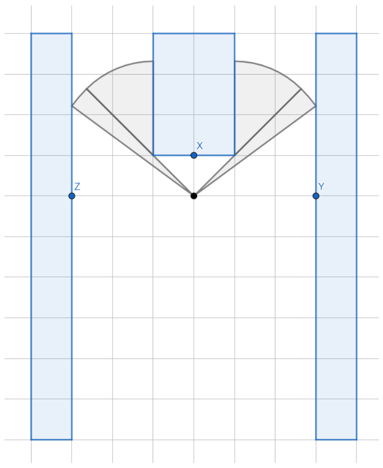


Figure 3.5: Since it is impossible to move from one side of the obstacle to the other without shortening the tether, we have encountered a divider. The divider is the marked point X. The points Y and Z are also dividers.

the robot in my simulation since I did not have a fully optimized SLAM stack such as GMapping found in [6] and as this was not deemed necessary for exploring the tethered navigation problem.

Since I did not use ROS or any other robotics library and therefore do not have an existing environment in which to implement my control algorithm, I had to construct my own. So many of the already well known algorithms had to be implemented in order to create my own algorithms. My algorithm is very complex and thus it would have been difficult to implement it in an unfamiliar environment.

To begin with I had to implement a map building system. Since I did not need any advanced mapping techniques I created my own mapping system from scratch. On top of this system I built functions for finding shortest paths in my map. Using [14] and [5] I implemented a shortest path algorithm which could be used in my environment after translating my map to a polygonal representation.

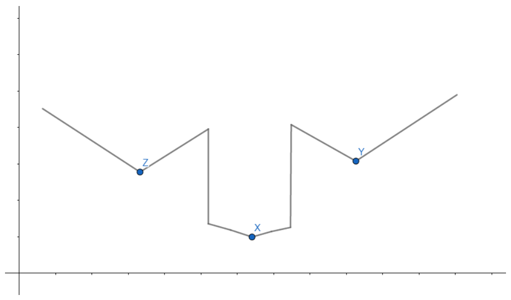
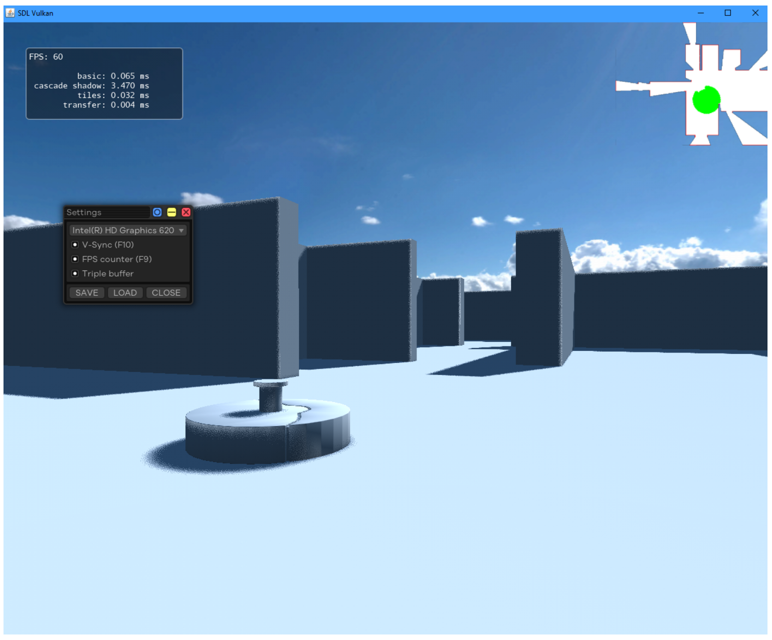


Figure 3.6: A sketch of a graph showing the distance from walls as a function of the angle. Here we can clearly see the divider points X, Y and Z are the local minima in the graph.



```
FPS: 60

           basic: 0.065 ms
  cascade shadow: 3.470 ms
           tiles: 0.032 ms
        transfer: 0.004 ms
```

Figure 3.7: A view of the 3D environment when fully rendered

3.3 Implementation

When the boundary expansion and divider finder algorithm is finalized, I have an expanded boundary (a geometrical description of how far the tether can reach with a certain length) which is divided in a tree structure. Now points can be sampled from the constructed boundary to construct a path, and the points that are occluded from the nodes are removed since they are either inside or behind an obstacle.

To connect each path, I used a pathfinding algorithm to find the shortest trajectory between the end of the first to the start of the next path. The algorithm I used was a simple polygonal pathfinding algorithm as it was deemed enough for the purpose. A more specialized shortest path algorithm[15] could have been used to decrease the likelihood of issues, such as colliding with the tether.

Chapter 4
Results

4.1 Shnaps and Rimon's algorithm

When attempting to implement Shnaps and Rimon's algorithm[13], as described in Chapter 2, several problems occurred. The algorithm is based on a numerical approach and divides the environment in cells in order to calculate motion paths. The crucial part is planning a new motion path for a new radius. The algorithm sorts these cells by the distance from the tether attachment point. This results in a path that will prioritize the new cells in the new radius in an unstructured way which looks almost random. This is due to that the grid layout of the cells will make some cells in the new radius closer than others. The algorithm works on a theoretical basis, the execution of it will make the robot cover the entirety of the space but will use an unstructured and less than efficient path to cover the space. See Fig 4.1

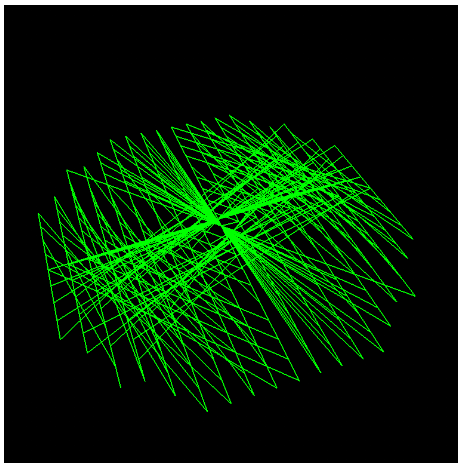


Figure 4.1: A simple example where Shnaps and Rimon's algorithm produces useless motion paths

This issue was too large to ignore. The planned path covered the same areas several times and was insufficient to build upon. I did attempt to fix this problem by making the algorithm first sort cells in chunks of similar distance to the attachment point and then secondarily on angle to latest tether bend. While this did improve the paths a little bit, it did however still produced bad and jagged motion paths. See Fig 4.2. And the sought after motion of contouring the distance to the attachment point was still missing.

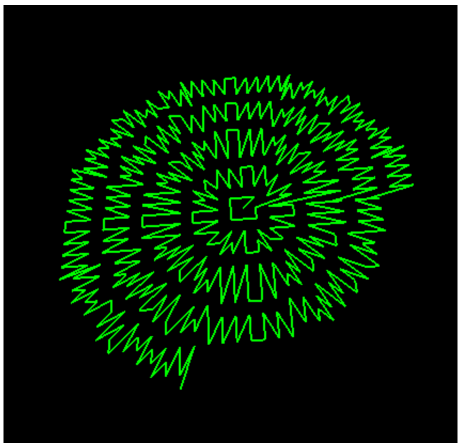


Figure 4.2: A simple example where Shnaps and Rimon's algorithm with my improvements produces jagged motion paths

4.2 My algorithm

In the end, I succeeded in creating an alternative algorithm to the one proposed in Shnaps and Rimon's paper. The algorithm produces a path contouring the distance to the attachment point and manages to cover most of the area in an efficient way. By looking at the map of traversed space, see Figure 4.5, recorded after the simulated robot has run, we can see that it covers most of the area. The spots that it has issues with are the edges and the corners. This is because according to the algorithm it will increment it search area one robot radius at a time. If the edge is not directly a factor of the radii away from the tethered point, it will not plan a path next to that wall. However, much of the space next to the walls still get traversed since the robot will pass next to them to get to the next part of the room.

My algorithm has been tested on simple spaces such as a rectangle room, see Figure 4.3, as well as complex spaces with many connecting paths and small branching rooms, see Figure 4.4. While it is not perfect, it does seem to handle most cases very well. A comparison of the coverage between Shnaps and Rimon's algorithm and my algorithm for a rectangular room can be seen in Figure 4.1

My algorithm increases the boundary in steps, and when a corner is passed but not included it is never considered for the algorithm again. In my algorithm I assume that the radius of the robot is small compared to the lengths of the walls. This means that when we expand our boundary one robot radius, we assume it will not expand it through an obstacle. If we expand the radius through or beyond an obstacle in one expansion step the algorithm will expand the boundary beyond corners on the opposite side of the obstacle and in practice these corners have not been observed and we do not know that they exist. This can cause

	Shnaps and Rimon's	Mine
Coverage	98%	94.1%
Path length	4743.68	380.27

Table 4.1: A comparison of the coverage and path lengths (unitless) of algorithms when covering the rectangle room

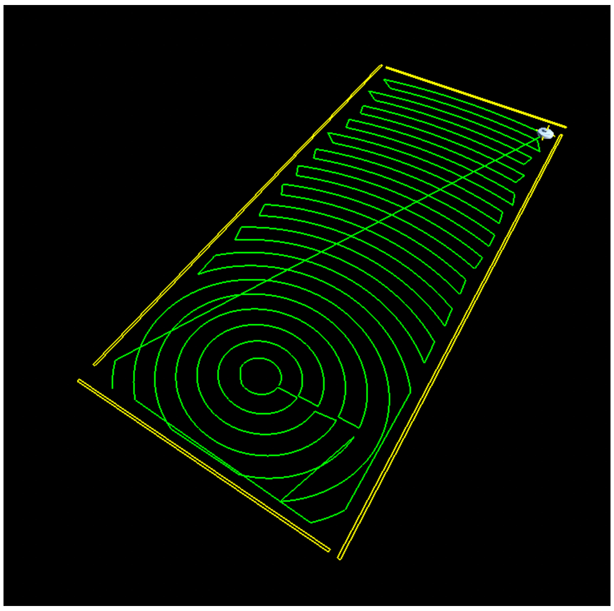


Figure 4.3: A simple example where my algorithm covers the entire area

areas of the rooms to be missed. I have not yet encountered this problem in a simulation since i use a reasonable size for my robot compared to the size of obstacles.

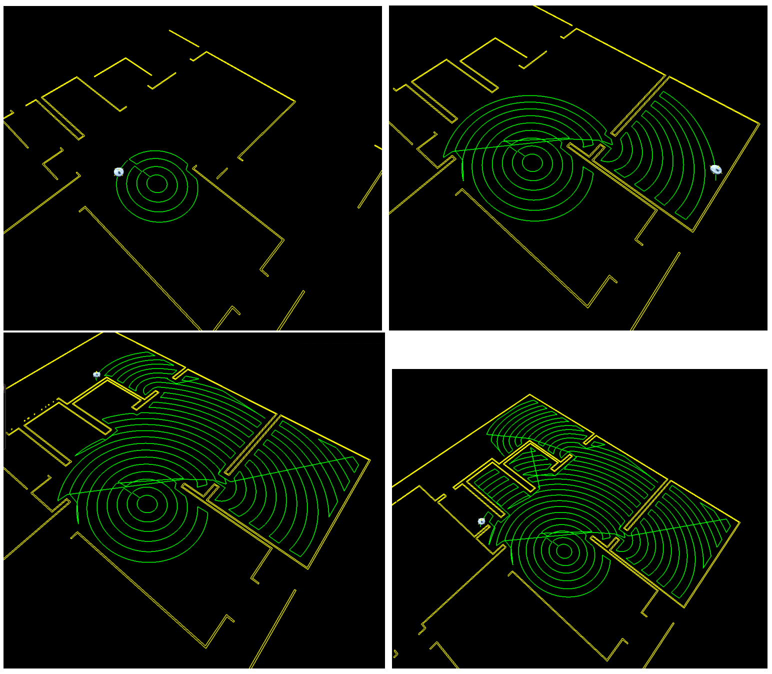


Figure 4.4: The motion plan generated from my algorithm in progress in a complex environment.

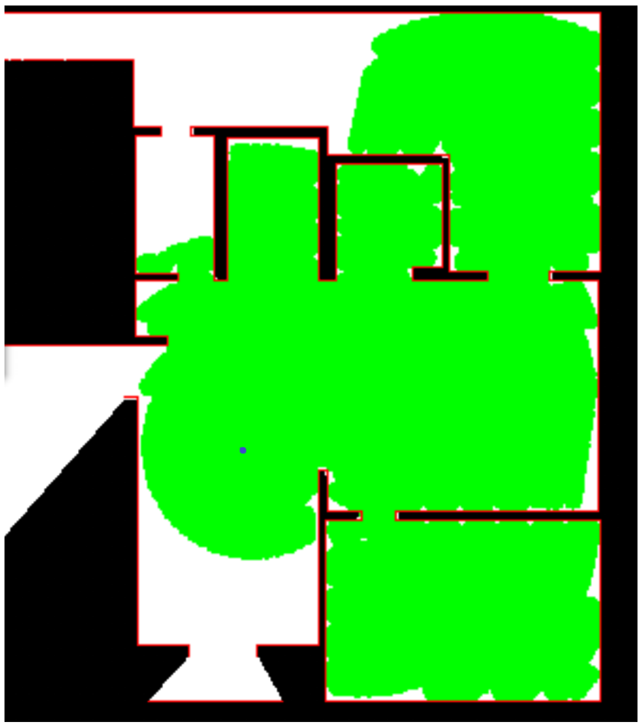


Figure 4.5: The green area shows the traversed regions by the robot using my algorithm in a complex room. The image is a snapshot while the robot is running. The tether origin is marked by the blue dot.

Chapter 5

Discussions and conclusion

5.1 Comparison

If we compare the two solutions for navigation, we can see some clear differences. The solution proposed by Shnaps and Rimon[13] is not as efficient as my solution since their path lengths are much longer. Their motion paths take many unnecessary small turns and traverse the same spots more than once, thus taking more time. However their paths are better at reaching corners and get closer to the walls as they will make sure to cover all cells the round robot can reach. Their solution is also more robust and can handle edge cases better.

A known issue of my solution is that it can miss large parts of the area since it cannot manage all cases of online motion planning. This can occur when a corner can not be seen when we try to extend our boundary beyond it. Then when it can be seen we do not retry to calculate its influence.

Another issue with my solution is that it is not time optimized since it was not important to the purpose of this thesis. Compared to the algorithm in[13] my algorithm is very slow and does not scale well with large environments. The algorithm works, but since some parts are computed relative to all corners within the reaching distance of the tether, it scales poorly when used for large distances in environments with many corners.

5.2 Real world applications

If using a tethered robot with no constraint on the tether position or direction, my algorithm should work very well according to my simulation. However, if studying the paths taken by the robot, there are cases when the robot faces towards the tether. This would cause issues if the algorithm was used on a robot which had a directional winding mechanism for its tether.

It is also not proven that the algorithm would work in a real world scenario. I did some basic assumptions when constructing my implementations, such as a wall can never disappear. If run on a real robot in a room it might not be able to cope with a door being opened.

5.3 Further Development

The most interesting parts to further research are the implementations of more dynamic systems. For example, can this algorithm be used for a different motion path? My way of using the algorithm is doing a breadth first traversal of the area graph, but if the motion paths are adapted it might be possible to do a depth first traversal and find a more efficient way of traversing the area.

The problem with a corner within a boundary not being seen, and therefore not being computed properly is an important issue to solve. Without this problem the algorithm would be as good online as offline and a solution to this problem would improve the reliability of the algorithm.

The time complexity problem is also one of the issues that would require further research. It might be possible to optimize the code e.g. by using an acceleration structure to speed up execution.

If used in a real world application, the directional constraints would be an important part to improve. If we could ensure the robot always extends and retracts the tether from one direction, this would enable the tether retracting mechanism to be mounted on one side of the robot, and be a lot less complicated. As mentioned, the motion path will face both towards and away from the tether. However, since most of the planning is done with boundary segments which are equidistant from the tethering point, the only part left to change to achieve these constraints are the paths between the boundary segments.

5.4 Conclusion

In conclusion, the purpose of the thesis has been reached. The new algorithm, despite its shortcomings, is an improvement in solving the problem compared to Shnaps and Rimon's algorithm and with a little further development it could be applied to a physical robot.

References

[1] Robot Operating System (ROS) homepage. http://www.ros.org.

[2] Pablo Abad-Manterola, Issa AD Nesnas, and Joel W Burdick. Motion planning on steep terrain for the tethered axel rover. In *2011 IEEE International Conference on Robotics and Automation*, pages 4188–4195. IEEE, 2011.

[3] Syed Ajwad and Jamshed Iqbal. Recent advances and applications of tethered robotic systems. *Science International*, 26:2045–2051, 01 2014.

[4] Peter Brass, Ivo Vigan, and Ning Xu. Shortest path planning for a tethered robot. *Computational Geometry*, 48(9):732–742, 2015.

[5] SD Goodwin, S Menon, and RG Price. Pathfinding in open terrain. In *Proceedings of International Academic Conference on the Future of Game Design and Technology*, page 8, 2006.

[6] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with Rao-Blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, 2007.

[7] Kazi Mahmud Hasan, Khondker Jahid Reza, et al. Path planning algorithm development for autonomous vacuum cleaner robots. In *2014 International Conference on Informatics, Electronics & Vision (ICIEV)*, pages 1–6. IEEE, 2014.

[8] Oussama Khatib, Xiyang Yeh, Gerald Brantner, Brian Soe, Boyeon Kim, Shameek Ganguly, Hannah Stuart, Shiquan Wang, Mark Cutkosky, Aaron Edsinger, et al. Ocean one: A robotic avatar for oceanic discovery. *IEEE Robotics & Automation Magazine*, 23(4):20–29, 2016.

[9] Soonkyum Kim, Subhrajit Bhattacharya, and Vijay Kumar. Path planning for a tethered mobile robot. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1132–1139. IEEE, 2014.

[10] P. Kong. Autonomous robot-like mobile chargers for electric vehicles at public parking facilities. *IEEE Transactions on Smart Grid*, 10(6):5952–5963, 2019.

[11] Edwin B Olson. Real-time correlative scan matching. In *2009 IEEE International Conference on Robotics and Automation*, pages 4387–4393. IEEE, 2009.

[12] Oren Salzman and Dan Halperin. Optimal motion planning for a tethered robot: Efficient preprocessing for fast shortest paths queries. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4161–4166. IEEE, 2015.

[13] Iddo Shnaps and Elon Rimon. Online coverage by a tethered autonomous mobile robot in planar unknown environments. *IEEE Transactions on Robotics*, 30(4):966–974, 2014.

[14] Anders Strand-Holm Vinther and P Afshani. Pathfinding in two dimesional worlds. *Master Thesis, Aarhus University*, 2015.

[15] Patrick G Xavier. Shortest path planning for a tethered robot or an anchored cable. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, volume 2, pages 1011–1017. IEEE, 1999.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2020-12-17

EXAMENSARBETE A Geometric Algorithm for Online Tethered Navigation
STUDENT Oskar Berg
HANDLEDARE Elin Anna Topp (LTH)
EXAMINATOR Jacek Malec (LTH)

En Geometrisk Algoritm för Trådbunden Realtidsnavigering

POPULÄRVETENSKAPLIG SAMMANFATTNING **Oskar Berg**

Även med dagens effektiva batterier så finns det situationer då robotar behöver mer ström än vad de batterier idag klarar av. I vissa av dessa situationer kan man med hjälp utav smart vägplanering byta ut batteriet mot en sladd. En lämplig tillämpning är robotdammsugare, då de idag endast borstar golvet för att en riktig dammsugare drar för mycket ström för att köras på batteri.

Jag valde i mitt examensarbete att fokusera på hur man täcker en yta effektivt när man är kopplad med sladd. Jag gjorde detta genom att först utvärdera en existerande algoritm med avsikt att utöka den. Men då den existerande algoritmen inte gav det förväntade resultatet, fick jag göra en alternativ algoritm till den existerande. För att göra detta använde jag mig utav rörelsemönstret som föreslogs ifrån artikeln men valde att åstadkomma detta mönster på ett annat sätt. I artikeln skapas en väg för roboten genom att använda ett rutnät som fylls i efterhand som roboten rör sig över rutorna. Genom att sortera de icke ifyllda rutorna enligt en viss algoritm kan roboten beräkna en rutt genom att gå till nästa icke ifylld ruta. Jag valde istället att skapa robotens väg genom att på ett geometriskt sätt räkna ut hur lång tråden blir om man flyttar roboten till olika positioner. Denna informationen användes sedan till att räkna ut hur roboten skulle röra sig.

Min algoritm är en kombination som består utav två delar. Den första är till för att hitta kurvor som roboten kan följa och fortfarande ha samma sladdlängd. Den andra används för att hitta ställen där man kan dela upp rörelsemönstret för att inte behöva gå fram och tillbaks runt hinder för mycket. Resultaten utav dessa två delar är en trädstruktur med kurvor som i tur och ordning ska följas för att köra över all yta i rummet.

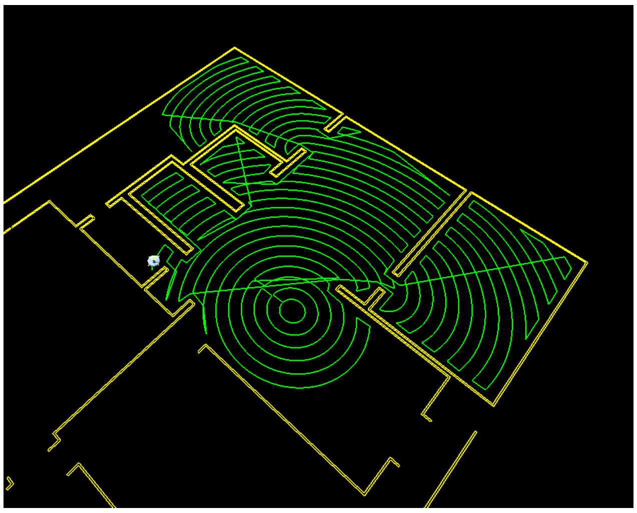


Figure 1: Vägen robotten tar när min algoritm används i ett rum.

I slutändan uppnåddes målet med att skapa en effektiv väg i rummet och förutom några få situationer klarade min nya algoritm att täcka området bättre än den algoritm som jag först testade.