DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2018

Dynamic Path Planning for Autonomous Unmanned Aerial Vehicles

URBAN ERIKSSON

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

Dynamic Path Planning for Autonomous Unmanned Aerial Vehicles

URBAN ERIKSSON

Master in Robotics and Autonomous Systems
Date: October 22, 2018
Supervisor: Patric Jensfelt
Examiner: Joakim Gustafson
Swedish title: Dynamisk ruttplanering för autonoma obemannade luftfarkoster
School of Electrical Engineering and Computer Science

Abstract

This thesis project investigates a method for performing dynamic path planning in three dimensions, targeting the application of autonomous unmanned aerial vehicles (UAVs). Three different path planning algorithms are evaluated, based on the framework of rapidly-exploring random trees (RRTs): the original RRT, RRT*, and a proposed variant called RRT-u, which differs from the two other algorithms by considering dynamic constraints and using piecewise constant accelerations for edges in the planning tree. The path planning is furthermore applied for unexplored environments. In order to select a path when there are unexplored parts between the vehicle and the goal, it is proposed to test paths to the goal location from every vertex in the planning graph to get a preliminary estimate of the total cost for each partial path in the planning tree. The path with the lowest cost given the available information can thus be selected, even though it partly goes through unknown space. For cases when no preliminary paths can be obtained due to obstacles, dynamic resizing of the sampling region is implemented increasing the region from which new nodes are sampled. This method using each of the three different algorithms variants, RRT, RRT*, and RRT-u, is tested for performance and the three variants are compared against each other using several test cases in a realistic simulation environment.

Keywords: Path planning, RRT, RRT*, UAV

Sammanfattning

Detta examensarbete undersöker metoder för att utföra dynamisk ruttplanering i tre dimensioner, med applicering på obemannade luftfarkoster. Tre olika ruttplaneringsalgoritmer testas, vilka är baserade på snabbt-utforskande slumpmässiga träd (RRT): den ursprungliga RRT, RRT*, och en föreslagen variant, RRT-u, vilken skiljer sig från dom två första algoritmerna genom att ta hänsyn till dynamiska begränsningar och använda konstanta accelerationer över delar av rutten. Ruttplaneraren används också i okända miljöer. För att välja en rutt när det finns outforskade delar mellan farkosten och målet föreslås det att testa rutten till målpunkten från varje nod i som ingår i planeringsträdet för att erhålla en preliminär total kostnad till målpunkten. Rutten med lägsta kostanden kan då väljas, givet tillgänglig information, även om den delvis går genom outforskade delar. För tillfällen när inga preliminära rutter kan erhållas på grund av hinder har dynamisk storleksjustering av samplingsområdet implementerats för att öka området från vilket nya noder samplas. Den här metoden har testats med dom tre olika algoritmvarianterna, RRT, RRT*, och RRT-u, och dom tre varianterna jämförs med avseende på prestanda i ett flertal testfall i en realistisk simuleringsmiljö.

Acknowledgements

I would like to thank my supervisor Patric Jensfelt for giving me the opportunity to do my thesis work in the drone lab. It was truly an interesting experience and a very positive environment to do the work in. It was also Patric who once accepted me to the system, control and robotics master program, thereby making it possible for me to enjoy two years of study at KTH, which have given me many fantastic experiences and much new knowledge. I would also like to thank my family, supporting me the whole time through the studies, making this whole thing possible.

Contents

1 Introduction

The drone industry has in recent years developed into a multi-billion-dollar business, and that figure is expected to continue to increase to above 100 billion dollars in the coming five to ten years [1]. Currently the main areas of application are recreational (i.e. hobby drones) and photography, and to some extent mapping. In the future, it is expected that areas such as surveillance and local delivery will contribute more to the growth. For this development to take place it is beneficial to reduce the amount of human involvement in the operation of the drones, not only to reduce costs, but also, in the long run to obtain consistent performance, and a concept which easily can be scaled. Therefore, it is very likely we will see a continued and increased interest in research and product development of unmanned aerial vehicles (UAVs), and in particular the autonomous aspect of UAVs.

When comparing UAVs to other types of vehicles and mobile robots, there are several key properties of the UAVs which influence what research questions are reasonable to be considered. First there naturally arise a few specific topics directly related to how the technology can be applied, such as crop analysis, pollution monitoring, and search and rescue operations (where for instance a lost person is to be located), just to take a few examples. From a more fundamental perspective, it is safe to say that a UAV will always have a relatively limited capacity for carrying payload, which makes it always important to reduce the energy consumption and the weight of the control system equipment. The UAV can also move freely and quickly in all three spatial dimensions which puts demands for high speed and low latency of the control algorithm of the vehicle. Furthermore, if the UAV has autonomous capabilities, the vehicle must be able to make plans and decisions by itself using only the limited onboard computing resources. In summary, that puts an incentive on the algorithms to prioritize efficiency and speed, rather than accuracy and completeness. However, with the tremendous development of computational capacity and energy efficiency of new hardware, this picture is changing, and more complex control software can start to be used. It is therefore always important to continuously assess and improve the algorithms used for controlling UAVs, so that maximum utility can be obtained from the vehicles.

Path planning is one of the central functional building blocks for an autonomous UAV. The capability to select a good path to a goal position is very important for saving air time, reduce fuel consumption, and also not to get caught in a deadlock somewhere on the way, which can happen if the planning algorithm is not sophisticated enough. For the autonomous UAV to be able to make correct decisions on which path to use when entering a mission, an analysis is required which takes into account the known constraints. Constraints can for instance be of the kinematic type, such as an obstacle situated between the vehicle and the goal, and other limitations on how the vehicle can move. A classic example of kinematic constraints is present when moving a piano, where it must be rotated in a certain way to be able to pass a doorpost. Another type of constraint is the dynamic constraint, which for instance can be inertia and external forces to take two examples. This constraint limits how fast a vehicle can accelerate, turn, and stop. In the case of a drone moving with high speed in the air, the dynamical constraints are not insignificant, since the steering forces which can be applied are relatively small compared for instance with a vehicle moving on the ground which can have high friction between its tires and the ground.

If the space around the UAV and towards the goal is not entirely known on beforehand the algorithm must be able to produce a preliminary path based on the information available at the planning moment. As the vehicle moves along this preliminary path, more of its environment will gradually be discovered. If an obstacle is discovered in the way of the selected path, re-planning needs to be done, and a new alternative path must to be selected. If re-planning is done continuously and with low latency, the algorithm is said to be online, which is typically what would be required in real world situation when the environment is not completely controlled.

1.1 Contribution of the work

This thesis investigates a method, which can be used by an autonomous UAV for path planning in a three-dimensional unexplored environment. The method is tested with three variants of planning algorithms. Two of the tested variants build on previously published work in path planning, the rapidly exploring random tree (RRT) [2] and the RRT-star (RRT*) [3] algorithm. The third variant, the RRT-u, is proposed in this thesis work and is described in detail in chapter 3. It considers kinematic as well as dynamic constraints, producing smooth, curved trajectories around obstacles, which is anticipated in some cases to simplify path planning and to reduce the time needed to execute the planned path. Analytic formulas have also been derived to obtain the parameters describing the trajectories, thereby using very little computing resources.

The path planning algorithm variants have furthermore been supplemented to handle the case when the space between the vehicle and the goal is partly unknown. This is done by calculating preliminary paths from every node to the goal. Similar to the probabilistic roadmap (PRM) [4], a possibility then exists of choosing between several paths to the goal point. This contributes to making the method suitable for online performance, which is also confirmed by experiments.

For cases when no preliminary paths can be obtained due to obstacles, dynamic resizing of the sampling region increasing the region from which new nodes are sampled is proposed as a remedy and has also been implemented.

The method was implemented in a software library, and finally tested with a realistic drone simulator.

1.2 Outline of the thesis

The remainder of this thesis report is organized into the following sections. In Chapter 2 related work in path planning is reviewed and related to the presented material. Chapter 3 describes in detail the proposed method. In chapter 4 details of the implementation are given. Chapter 5 contains the test setup and the test cases are described and the results from the experiments are presented and discussed. Chapter 6 discusses the social and ethical impact of the technology related to the presented material. Chapter 7 contains the conclusions and some thoughts on future work.

2 Related work

Path planning algorithms is a vast and diverse research area dating back more than 50 years starting with Dijkstras [5] algorithm and the A* algorithm [6]. Today there exist a multitude of algorithms and they usually each have a specific domain of application where they perform the best. For continuous configuration spaces with high dimensionality, it can easily become very computationally expensive to divide the space into a grid, and search for the optimal path, as is done with an algorithm such as the A*. Also, the storage of the search graphs rapidly becomes impractical with increasing problem size. Instead, for such cases, methods using random sampling of the configuration space have successfully been introduced. There exist two dominant classes of random sampling methods, the rapidly-exploring random tree (RRT) algorithm [2] and the probabilistic roadmap (PRM) [4]. RRT works by building a search tree starting from a root node and is suitable for finding a single path from start to goal. PRM builds a graph of connected nodes and can be queried for paths between any two nodes in the graph.

2.1 The RRT algorithm

The RRT algorithm works by building a tree data structure from a root node located for instance where the vehicle is currently situated. Samples are drawn from the configuration space of the vehicle, and if it is a free sample (not inside an obstacle or in unknown space) it will be used to construct a new vertex in the graph. The next step is to search the tree to find the vertex which is nearest to the newly sampled point, using some distance metric. A new configuration is then found by moving an incremental distance from the nearest vertex towards the sampled configuration (steering). This assumes the movement is possible and not limited by some constraint, such as an obstacle for instance. If successful, the new configuration is added as a new node in the search graph. This procedure is repeated for a predetermined number of nodes or some time limit is reached. If a node is near the goal path be constructed by backtracking the vertices in the tree all the way back to the root vertex. An example of an RRT planning graph is presented in figure 1.

It has been shown that there is unity probability to find a path to the goal using RRT, given there is no limit on the number of vertices or time. The algorithm can thus be said to have probabilistic completeness. Even if a path always can be found using RRT in most cases it does not represent an optimal path since it usually contains many sharp turns, even in obstacle free areas. It was later shown that the probability for RRT converging to the optimal path is in fact equal to zero [7].

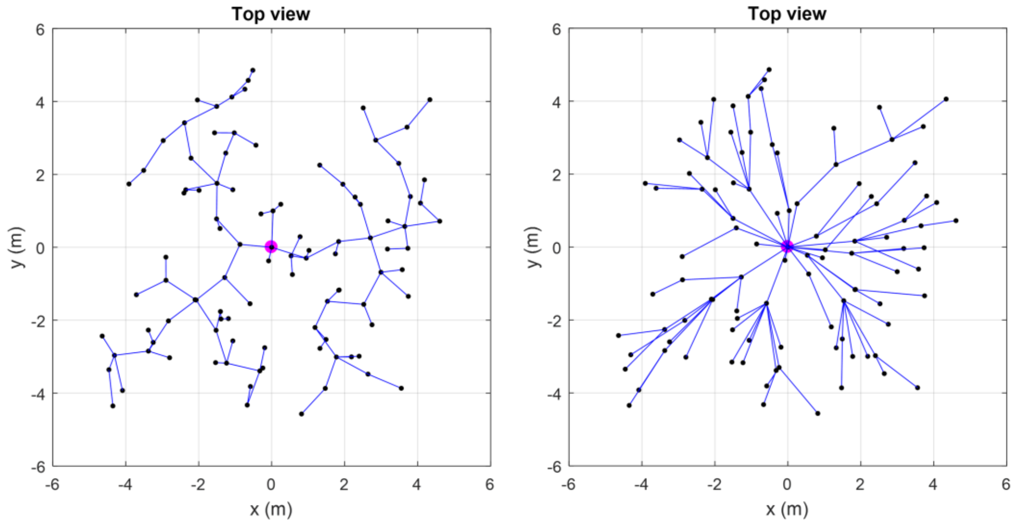

Fig. 1. Two-dimensional projections of 3-D planning graphs consisting of 100 vertices using two different algorithms: (a) RRT (b) RRT*

2.2 The RRT* algorithm

A variant of the RRT algorithm, called RRT*, was proposed by Karaman and Frazzoli [3]. In this method there is an additional rewiring step included, which examines nearby vertices to first find the optimal parent to the newly added vertex, and then reroute edges of neighboring vertices through the newly added vertex if it is found that the resulting path is more optimal by doing so. This can be seen in figure 1, where the RRT* algorithm exhibits paths which have been rewired so that they are straighter and for this case then closer to optimal. The algorithm has been shown to find a path which will in the limit of infinitely many nodes almost surely converge to the optimal solution. At the same time the time complexity is kept the same as for the RRT algorithm (execution time is only increased by a constant factor).

2.3 Dynamic constraints

The RRT and RRT* algorithms, as they were originally proposed, are most straightforward to implement when the vehicle is holonomic and there are no dynamic constraints. Then the vehicle can always change its heading in an arbitrary direction at any point. Plots of the resulting search trees show edges which are perfectly straight lines between vertices, see figure 1, and there is no curvature at the vertices even though the path changes direction. For an UAV this is not a realistic situation since inertia will produce turns which are not very sharp when changing direction, especially when the velocity is high. One approach to mitigate the problem of non-ideal paths, is to post-process a path with sharp corners using smoothening or path optimization, which will result in a closer to optimal solution. If instead the dynamic constraints of the vehicle are included in the path planning algorithm from the start, there will be no need for the post-processing. The downside of this is that the planning algorithm will become more complicated and require more elaborate calculations. A summary how to include dynamic constraints into path planning can be found in chapter 14 in [8]. Normally the full state space is sampled, containing both the position and the velocity components of a point, and the nearest neighbor according to a cost function is evaluated. This evaluation is usually quite cumbersome because an optimal control policy must be found for the trajectory between vertices to be able to compare the cost of connecting a new vertex to different neighbors in a fair way. In practice, many times alternative methods are applied. In [9] and [10] a method is discussed where post-planning tailoring of the velocity profile of the trajectory is done. It is pointed out though in [11] that the path and the velocity profile can be separated when connecting two vertices with for instance a cubic spline. However, it has been reported that there can be a danger when doing parametric interpolation of trajectories because it can be sensitive to noise and exhibit oscillations [12]. To reach an optimal solution for the trajectory when connecting two vertices which both have positions and velocities defined several different approaches has been proposed. One common approach is to use optimal control theory to optimize a desired quantity such as the duration or the fuel consumption [13]. The problem of optimizing a trajectory given the full state vector at the boundary points has also been solved by [14] and they show that in some cases a closed-form solution for optimal trajectories can be derived. Another approach to reduce computation time at the time of path planning is to precompute a large number of "primaries", which are curves that are solutions to the trajectory optimization for different boundary values [15].

To potentially increase the relevance of the solution, the RRT* algorithm can be used including differential constraints [16]. The distance function is then redefined using a more elaborate cost function which better fits the problem.

By sampling on the control input a closed loop RRT variant can be obtained with bounds on the tracking error [17].

2.4 RRT-connect

Several other variants of the RRT and the RRT* algorithm exist. The RRT-connect [18] builds two rapidly-exploring random trees rooted at the start and the goal vertex. In each iteration as the tree is extended, an attempt is made to connect the nearest vertex of the other tree to the new vertex of the first tree. There are also other variants of the algorithm with multiple RRTs [19], [20], which claim improved performance in cluttered environments with many obstacles and narrow passages.

2.5 Online updating

To improve the speed of the algorithm several techniques are used. In [21] a procedure is described where the duration of the initial planning is fixed and limited. Even though no path exists to the goal area the best so far is chosen and the initial part for the trajectory is committed to. Then planning can proceed for the rest of the tree.

What [22] are describing is an exploring scenario where two RRT search trees are used. One is for detecting the frontier points of the map which can be described as point lying on the border between explored and unexplored area. The second RRT is used to guide the robot toward a selected frontier point.

A new variant of RRT called RRTx [23] claims to be able to handle unpredictable obstacles in real time, by cascading rewiring downwards in the tree when an obstacle is detected.

2.6 Obstacle avoidance

Dynamic obstacle avoidance and rerouting can be problematic with RRT as this is a single query algorithm. It implies that re-planning needs to be done when an obstacle is detected. For PRM the situation is better since there already exists a family of possible paths to the goal [24], from which a new path non-colliding path can be selected. In the RRT case, to trigger a re-planning event, a method has been presented where the angle towards a moving object can be used to determine if the robot is on a colliding course with an obstacle or not [25]. If that is the case a detour is planned around the obstacle back to the original optimal path again. The reason for going back to the original path is to avoid excessive computations whenever an obstacle is appearing.

2.7 Related work built on in this thesis

- The work in this thesis is based partly on the algorithms for RRT and RRT* as described in [2] and [3].
- How to include dynamic constraints into the path planning can be found for instance in [10]. However, the boundary constraints have been applied differently for the proposed RRT-u method, which is described in more detail in chapter 3.
- The idea to include goal trajectories into the planning graph comes from the RRT-connect [18], but since the whole map is not known, and especially so around the goal, it makes no sense to plan and maintain two trees, so if one would make an analogy to the RRT-connect method, one tree only contains the goal vertex. Another way to view this, is to see the result of the presented method as a PRM planning graph, but only between the root node and the goal node. The multi query functionality comes in when an obstacle is detected and blocking the currently optimal path. Then a new optimal path can directly be selected.

- To keep a constant number of nodes and use the best solution so far was described in [21].
- The procedure to commit to path segments and to move the root node one step up in the planning tree was found in [26]. The motivation for using this method instead of the RRTx [23] is that the latter method has the root of the planning tree at the goal point and therefore must plan the whole unexplored volume of the map in front of the vehicle, instead of only the volume near the vehicle.

These concepts were implemented and combined in the presented work. The dynamic resizing of the sampling region was not found yet in any work. It was added later during the work to make in order to make the RRT and the RRT* path planning algorithms and to some extent the RRT-u algorithm work when no preliminary paths to goal existed.

3 Method

The proposed method can be divided into three major subareas:

- The path planning algorithm (RRT, RRT*, or RRT-u)
- How to choose the size of the sampling region
- Online path selection and pruning of the planning tree

The path planning algorithm describes how to build a planning tree consisting of possible paths the vehicle can take. Here only the novel RRT-u algorithms is described since the other two can be found in the literature ([2] and [3]).

The dynamic resizing of the sampling region is introduced to optimize the performance of the path planning, since the efficiency of the path planning is related to the size of the sampling region. At the start of the planning the sampling region has a nominal size. When no preliminary paths leading to the goal point exist the sampling region needs to be increased.

The path selection section describes the procedure of selecting which path to follow, and some dynamic aspects such as ongoing collision detection of paths. Also pruning of the tree is described, which is advantageous for making the tree more compact and reducing the total number of vertices in the tree.

3.1 The RRT-u path planning algorithm

The novel path planning algorithm presented in this thesis is called RRT-u and builds on the original formulation for creating a rapidly exploring random tree (RRT), with the addition of applying dynamic constraints on the local motion planning. Optimal parameters for the dynamic behavior are found by minimizing a cost function, for the local motion planning between two vertices, and analytical formulas for the parameters are derived.

The general algorithm for building the planning trees according to the RRT-u recipe is given as pseudocode in figure 2. The main function of the planner, MAIN_RRT-u takes a start vertex and a goal position as input parameters, together with the maximum number of nodes which can be present at the same time in the planning tree. Also, a timeout limit can be given so that planning is only taking place for a specified duration, leaving the actual number of vertices in the graph less than the maximum, but giving better control of execution time.

```
MAIN_RRT_u(q_start, p_goal, N)
----------------------------------------------
q_root ← q_start
n ← 0
while n < N
        p_rand ← RandomSampling()
        success ← ExtendTree(q_root, p_rand, p_goal)
        if success then
                n ← n + 1
```

```
EXTEND_TREE(q_root, p, p_goal)
----------------------------------------------
q_parent ← GetNearestNoCollidePathVertex(q_root, p)
if exist(q_parent)
        q_new ← Steer(q_parent, p)
        AddRelation(q_parent, q_new)
        q_goal ← NewNoCollidePathVertex(q_new, p_goal)
        if exist(q_goal)
                AddRelation(q_new, q_goal)
        return true
else
        return false
```

Fig. 2. Pseudocode for the RRT-u planning algorithm.

The tree is building from a root node, which contains the position and the velocities of the vehicle when planning begins. In the main function there is an iteration performed where new vertices are created by sampling random locations and extending the tree until the maximum number of vertices is reached. Note that the random sampler only returns locations, not the full state space, and that the dynamical parameters determining the velocity profile and the exact shape of the path between vertices come later as the result of the cost minimizer function.

The EXTEND_TREE function takes the root node, the new sampled point, and the goal point as parameters. The function starts by searching the whole of the planning tree for the lowest cost parent, the q_{parent}, which is defined as the node which produces the lowest cost when going from the parent node to the newly sampled location. Only parents which have an entirely collision free trajectory all the way to the new position can be considered as optimal parents. Sometimes there are no nodes fulfilling this condition, giving an empty result of the search. After that the steer function is applied, and a new node is created part way or all the way in the direction of the random point.

The relationships between the nodes are stored so that it is later possible to for instance enumerate all the nodes by just knowing the root node and traversing its descendants. Also, the reverse relationship is stored which enables a path to be back-tracked all the way to the root node from any node in the planning tree.

Finally, it is investigated if it is possible from the new node, q_{new}, to reach the goal without collisions with obstacles. If so, a new node is created at the goal point, and a relation is added from the new node to the goal node. The total cost is the accumulated cost of all edges from the root to the goal and is stored together with the goal node and later used to determine which of the possible paths to the goal position is the best path in terms of cost.

3.1.1 Dynamical constraints

To incorporate dynamical constraints, a node is specified by adding the parameters governing the dynamical behavior for the vehicle. They are the velocities and accelerations for each dimension, in addition to the position coordinates. The accelerations are not directly related to the vertex itself. Instead, they are related to the edge upwards in the tree, specifying the accelerations for the trajectory spanning from the vertex to its parent.

$$q = \begin{pmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \\ a_x \\ a_y \\ a_z \end{pmatrix} \tag{1}$$

When calculating a trajectory between two vertices, the duration Δt is the time it takes to go from one vertex to the other. The position and the velocities of the vehicle will gradually change, while the accelerations are assumed to be piecewise constant, and only change in discrete steps when moving from one edge to the next in the planning tree. The accumulated cost is also stored together with the rest of the parameters at a vertex to facilitate path selection.

3.1.2 Cost calculations

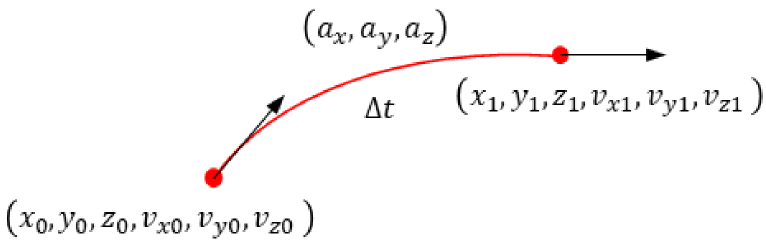

Fig. 3 The quantities involved in the local motion planning between two vertices. The position and the velocities of the first state and the position of the second vertex are given. The accelerations and the velocities at the second vertex are a result of minimizing the Δt.

The cost function for the RRT-u method is based on calculating the time it takes for the vehicle to travel between two nodes, so that when looking at a planning tree, the nearest neighbor is then simply defined as the node from which it takes the shortest time to travel. It does not necessarily have to be the node closest in space. The exact implementation of what counts as a cost is actually quite arbitrary and can, if desired, be changed to something else, such as for instance energy consumption. The quantities which are used in the cost function are shown in figure 3. The algorithm is defined so that the accelerations (a_x, a_y, a_z) are the free parameters and the velocities (v_{x1}, v_{y1}, v_{z1}) are functions of the accelerations.

There are a few limiting conditions which are given by the dynamical model used by this method, which must be considered when doing the cost minimization. To start with, the vehicle is approximated with a point mass which can be freely moved in any direction of three-dimensional space. One of the limiting conditions is that the acceleration in every dimension is constant throughout the entire trajectory between two nodes. There is also an upper limit for the absolute value of the acceleration. Then, finally, the absolute value of the velocity for each dimension is always required to be below a maximum value.

Analyzing the problem further, given the above limitations of the model, there will only be a limited number of cases to consider. Either the acceleration for some dimension is at its maximum value during the whole course of the trajectory, or the velocity of the same dimension reaches its maximum value at the end of the trajectory, while the acceleration is at a constant value lower than the maximum. There are three dimensions to investigate and consequently three distances that needs to be traversed when travelling from one node to another, which are equal to the differences between the end positions and the start positions:

$$\Delta x = x_1 - x_0 \tag{2}$$

$$\Delta y = y_1 - y_0 \tag{3}$$

$$\Delta z = z_1 - z_0 \tag{4}$$

The optimal case will occur when selecting the dimension which takes the shortest time to complete its delta distance, using either maximum velocity or maximum acceleration, while the other dimensions are fulfilling the conditions at the same time which is to be within their acceleration and velocity limits. Once the Δt is known the acceleration parameters for all dimensions can be calculated and stored. Using Δt and the acceleration parameters for all three dimensions, the trajectory between two nodes can then easily be calculated.

How to calculate two possible cases giving a minimum Δt will now be gone through in detail below.

Maximum velocity case:

We will just be considering the x-dimension here because the other dimensions can be treated in an identical way. The objective here is to traverse the distance Δx from x_0 to x_1 in the shortest time, using the highest possible value for the acceleration, but without exceeding the maximum velocity limit. The velocity at point x_0 is given as an input and is equal to v_{x0}, which is less than or equal to the maximum allowed velocity. The solution to this problem is to let the velocity at x_1 be equal to the maximum allowed velocity.

$$v_{x1} = v_{max} \tag{5}$$

The time it takes in to travel the distance is then simply the distance between the two points divided by the average velocity.

$$\Delta t = \frac{\Delta x}{(v_{x0} + v_{x1})/2} = \frac{2\Delta x}{v_{x0} + v_{max}} \tag{6}$$

The acceleration in the x-direction is further obtained by dividing the change in velocity with the elapsed time.

$$a_x = \frac{v_{max} - v_{x0}}{\Delta t} \tag{7}$$

The accelerations for the two other dimensions are now obtained from solving for the acceleration in the expression for calculating the delta distance as a function of the acceleration, delta time, and the initial velocity.

$$a_y = \frac{2}{\Delta t^2}\left(\Delta y - v_{y0}\Delta t\right) \tag{8}$$

$$a_z = \frac{2}{\Delta t^2}\left(\Delta z - v_{z0}\Delta t\right) \tag{9}$$

Having calculated the acceleration components, the two resulting velocity components at the end location of the trajectory can be calculated in a straightforward way:

$$v_{y1} = v_{y0} + a_y \Delta t \tag{10}$$

$$v_{z1} = v_{z0} + a_z \Delta t \tag{11}$$

There can be a range of different outcomes from these calculations. For instance, the Δt from eq. (6) can be positive, undefined, or negative, depending on the values of the velocities which are input to the equation. Only a positive Δt is a valid solution. The absolute values of the acceleration components, a_x, a_y, and a_z can be larger than the allowed values, making the calculated cost, Δt, not a valid option to consider. The absolute value of the velocity components v_{y1} and v_{z1} can also be larger than the maximum allowed velocity, making the obtained Δt invalid.

The Δt calculation is done for the two cases when $v_{x1} = v_{max}$ and $v_{x1} = -v_{max}$, and for the three different dimensions, adding to a total of six different cases for the above calculations.

<u>Maximum acceleration case:</u>

The other main case is when the acceleration is at a maximum during the whole course of the trajectory. Then, consequently, either $a_x = a_{max}$ or $a_x = -a_{max}$. The time it takes to traverse the delta distance in the x-direction is equal to:

$$\Delta t = -\frac{v_{x0}}{a_x} \pm \sqrt{\left(\frac{v_{x0}}{a_x}\right)^2 + 2\frac{\Delta x}{a_x}} \tag{12}$$

The two acceleration components which are not given are calculated in the same way as in eq. (8) and (9):

$$a_y = \frac{2}{\Delta t^2}(\Delta y - v_{y0}\Delta t) \tag{13}$$

$$a_z = \frac{2}{\Delta t^2}(\Delta z - v_{z0}\Delta t) \tag{14}$$

The three velocity components at the end of the trajectory can be calculated, similar to eq. (10), as follows:

$$v_{x1} = v_{x0} + a_x \Delta t \tag{15}$$

$$v_{y1} = v_{y0} + a_y \Delta t \tag{16}$$

$$v_{z1} = v_{z0} + a_z \Delta t \tag{17}$$

Once again only a positive Δt is a valid solution. The acceleration a_x is given but a_y, and a_z need to be checked so that their absolute values are not larger than the maximum allowed values. The absolute value of the velocity components v_{x1}, v_{y1}

and v_{z1} must also be checked so that they are not larger than their maximum allowed value.

The Δt calculation is repeated for the two cases when $a_x = a_{max}$ and $a_x = -a_{max}$. The two solutions of the second-degree polynomial equation (12) add another two rounds of calculation. Combining that with three different dimensions amounts to a total of twelve different cases for the above calculations.

<u>Summary of cost calculation:</u>

A total of 18 different cases are all tried and the optimal solution (minimal value) for Δt is obtained by taking the minimum of all cases. If no path between the start and the end position can be found, which can be the case if the new point is completely blocked by an obstacle, the algorithm will give a null result, and a new attempt must be made with another sampled point. If an optimal Δt is obtained, the corresponding parameters can be obtained (the velocities and the acceleration components) determining shape and the velocity profile of the edge between the parent node and the new location. A new vertex can then be created in the graph where relevant information such as the accelerations and the Δt is stored. An example of how the RRT-u planning tree can look like is shown in figure 4.

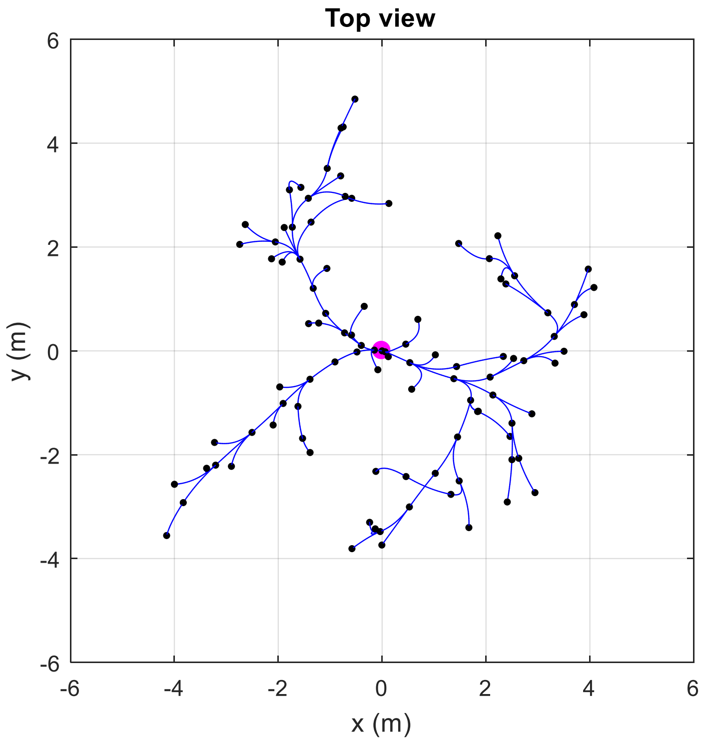

Fig. 4. Planning tree using the RRT-u algorithm, consisting of 100 vertices.

3.1.3 Collision checking

Collision checking is performed for the RRT-u algorithm, as well as for the RRT and RRT* algorithms, before adding new vertices to the planning tree

Before a new node can be added, the path segment between the nearest vertex and the new vertex is checked for collisions. The path segment does not have to be entirely classified as free, it can contain unknown parts also, but there is a requirement that there are no parts of the curve going through space which is classified as occupied.

Also, the entire path leading up to the random sampled point needs to be collision free, even if only a part of the distance of that to the new point would be used (limited by the steer function). Otherwise it is for instance possible to get depleted areas behind walls because the closest points can consistently be on the other side of the wall, and the paths are then extended on the “wrong” side of the obstacle.

If extra margins to obstacles is desired, checking for closeness to occupied areas can be included in the collision checking, and paths which are close to obstacles can be disregarded. The exact implementation of how to achieve that may vary, but at least some method should always be used since the planned path just consists of line segments which do not occupy any space, but a real vehicle always has some finite size in all dimensions.

3.2 Dynamic resizing of the sampling region

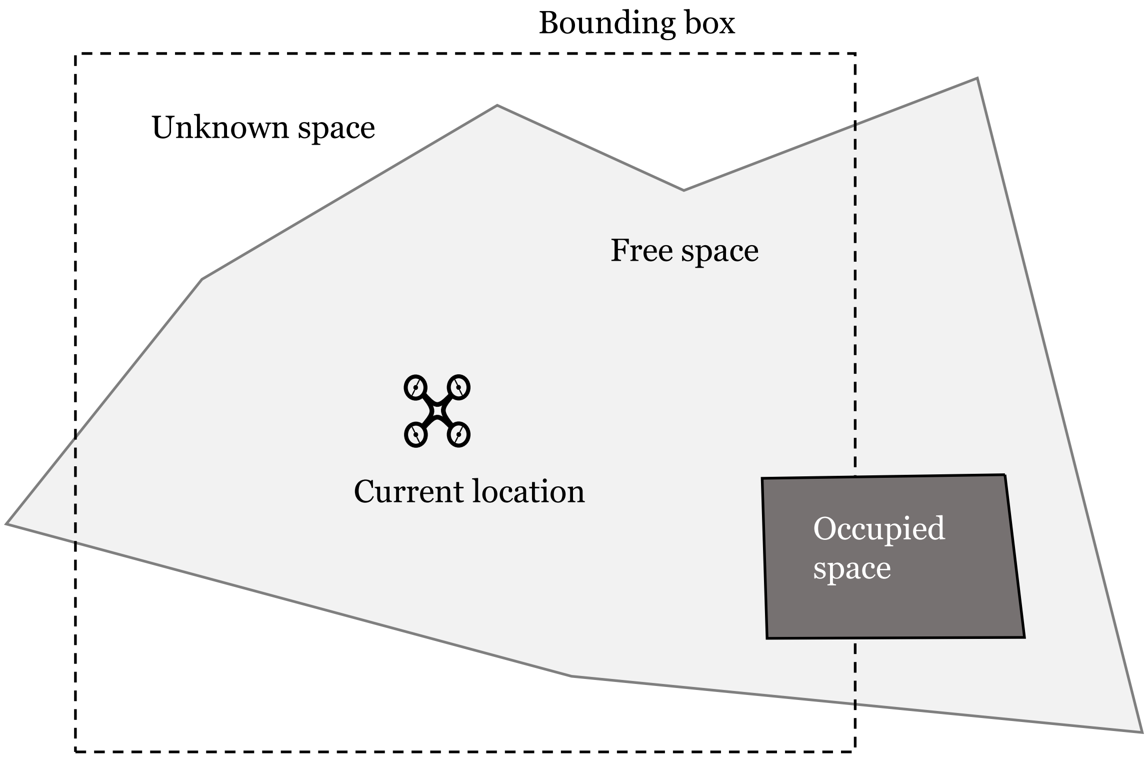

Fig. 5. The sampling region is limited by a bounding box. New random samples can be placed in both free space and unknown space, but not in occupied space.

Random sampling is the process which gives the locations of new vertices which are later added to the planning tree (given that they fulfill certain required conditions). Random sampled points are only valid if they are not in occupied space, see figure 5. In this work random points are sampled inside a bounding box situated around the vehicle. The bounding box moves with the vehicle, and the reason behind this is for the sampling density not to be diluted, as more and more volume is covered by the vehicle. If the bounding box instead would be expanded, lots of samples would fall into areas visited previously in the planning process and many of the new vertices would most likely not be very relevant in the search for a good path to the goal point, simply since they would be sampled much further away from the goal than the vehicle is at that moment. On the other hand, using a limited sized bounding box for sampling can cause other problems, such as getting caught in deadlocks from where not a single potential path to the goal can be found, due to the goal being screened by an obstacle. It is therefore in this work tested to increase the size of the bounding box dynamically in situations when no potential paths to goal exist and then reduce it again to the nominal size when potential goal paths are found again.

How and when to change the size of the region which is used to randomly sample new vertices has not been very much discussed in the literature. Partly, the reason for that is probably that much of the published work concerns planning when the entire map is known, and this naturally gives the range of configurations for the vehicle. Also, one can argue that sampling, and thus planning in unknown parts of the space is not a very interesting problem. For instance, planning a path in totally unknown space for a holonomic robot would simply give a straight line from the start to the goal as the most optimal path. However, when some parts of space are known, and other parts are unknown, there can appear situations when sampling in unknown space can be useful.

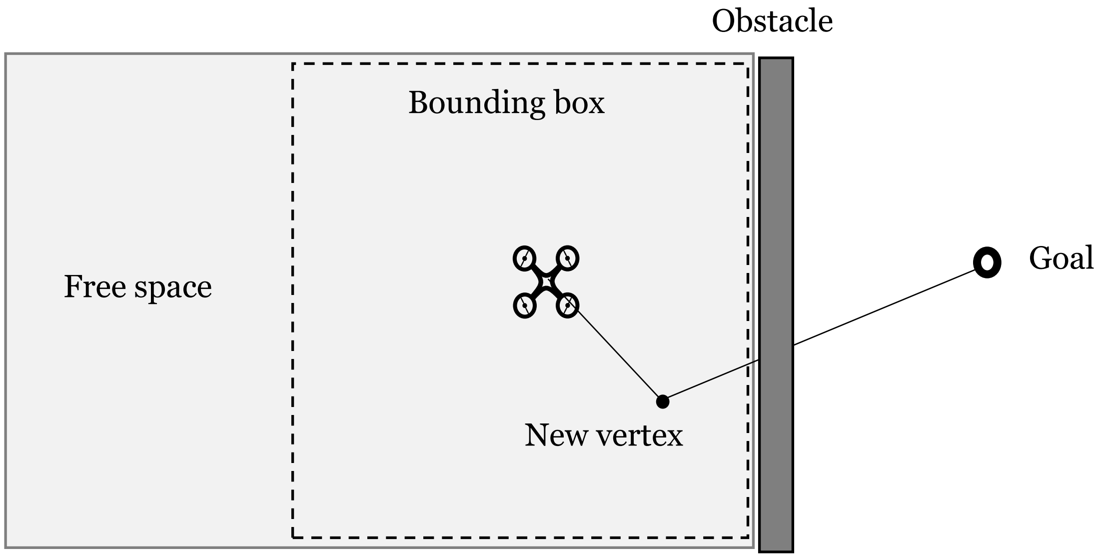

Fig. 6. When using a bounding box of a small size, e.g. only of free space, the goal region can be completely blocked by an obstacle.

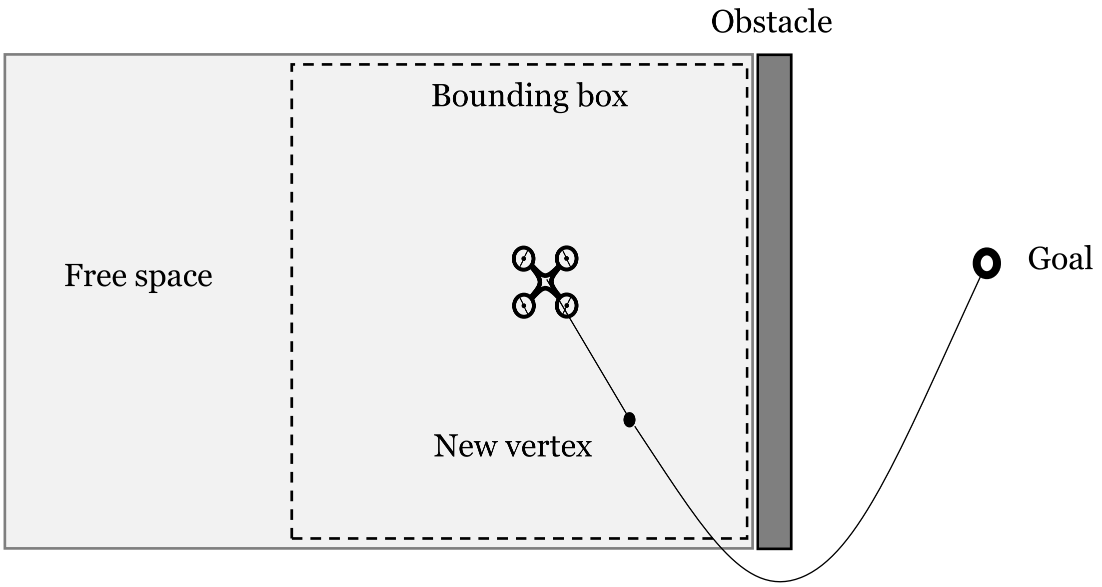

Fig. 7. By using a large sampling region size, and placing new vertices in unknown space, the chance of finding a potentially non-colliding path to goal increases.

In this work, when a vertex is created it will always be investigated to see what potential it has for creating a good path to the goal. One requirement for such a path to the goal is for example that it does not go through any obstacles. There can be other constraints as well, both kinematic and dynamic. Looking at the RRT and the RRT* algorithms first, they are here implemented assuming holonomic motion, which means the vehicle can maneuver in an arbitrary direction at any point. For this case it is quite possible that there is not a single vertex in the planning tree that has the possibility to create a valid path to the goal point, especially if the new vertices are sampled only from known free space, see figure 6. When this situation occurs, the bounding box can be increased, and points can be sampled in unknown space as well. Then it is more likely to sample a point that produces a vertex with an approved potential path to the goal, see figure 7.

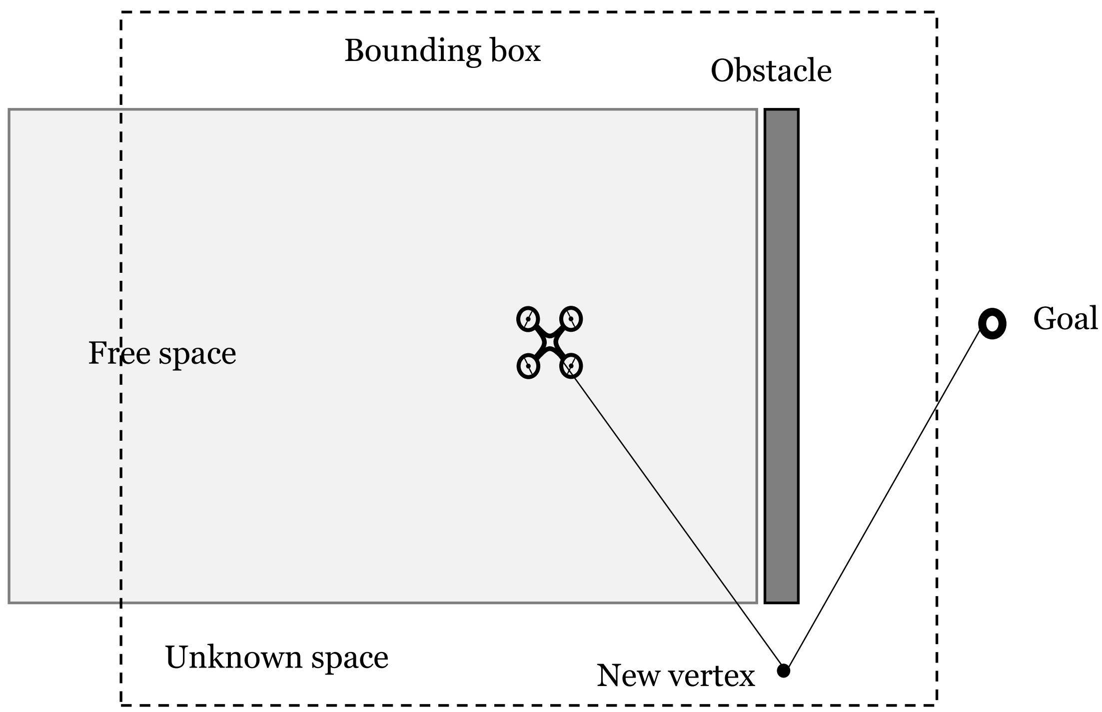

Fig. 8. The RRT-u planning algorithm, using dynamic constraints, has the ability to plan around corners of obstacles.

The RRT-u algorithm on the other hand has a different local planning strategy than the RRT and the RRT* algorithms. The RRT-u algorithm takes dynamic constraints into account and therefore produces smooth curved paths, since constant accelerations for the different spatial dimensions are allowed over a path segment from one vertex to another. This gives an advantage for the RTT-u algorithm because it can make path planning around corners, see figure 8, and thus the bounding box does not need to be as big as for the RRT and the RRT* cases.

3.3 Path selection and tree pruning

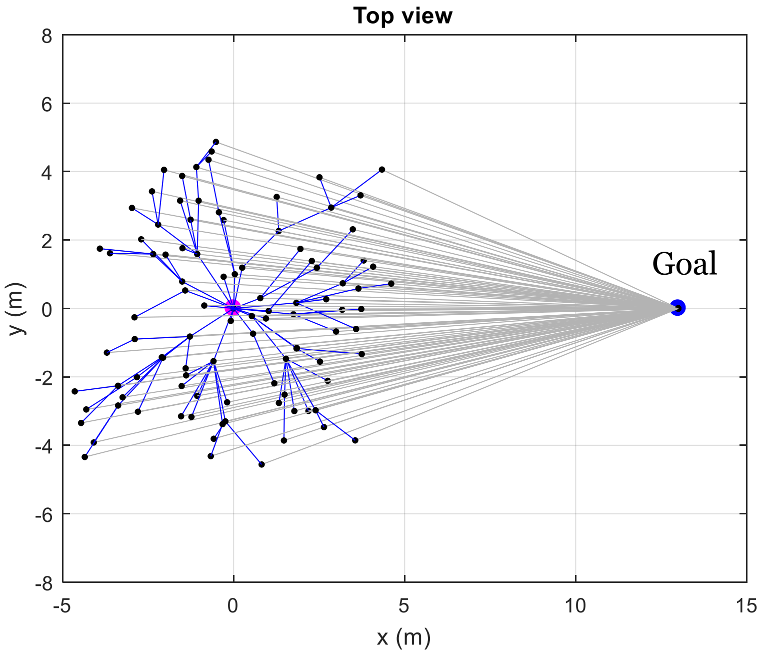

Fig. 9. An example of a planning tree using the RRT* algorithm shown in blue and the preliminary paths leading to the goal point shown in gray.

After path planning has been done (in this work for a maximum specified amount of time or a maximum node count), the next step for the vehicle is to commit to a path segment and start following the committed trajectory towards the next waypoint. In the planning tree there usually exist several different leaves which have the goal point as its location, see figure 9. In fact, for every node which is created by the sampling process an attempt is also done to connect it by a path segment to the goal point, and if successful create a vertex there. Therefore, there are usually many nodes at the same spot at the goal position, but each of them has a different parent, and they also each have a different total accumulated cost associated with them, determined by the specific path leading up all the way from the root node to the goal node. The estimated total cost to reach the goal is what determines how good the different paths are in comparison.

(a)

(b)

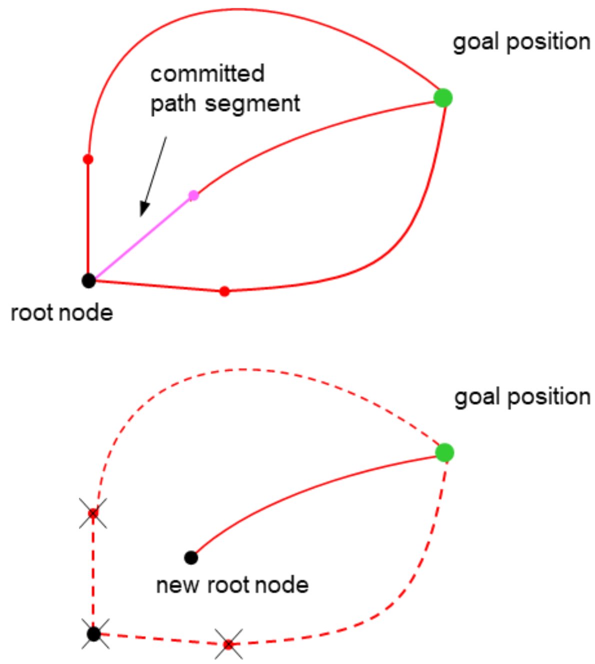

Fig. 10. (a) A path segment is committed on the path which potentially will give the lowest cost for reaching the goal. (b) The new root node will be the vertex at the end of the committed path segment. Branches from the old root node become obsolete and can be pruned and vertices downstream can be removed from the planning graph.

The optimal path, which is the path with the lowest estimated cost for reaching the goal, should thus be the path ending at the goal position with the smallest accumulated cost. This is the candidate path which will be investigated further to discover if it fulfills every requirement to be selected as the path to follow.

For a path segment to be committed to it needs to be checked that it travels through free area only. Unknown area is not allowed for the path segment closest to the vehicle since no further collision checking will be done for this path segment by the path planner. Also, the other path segments following the first path segment, leading up to the goal are checked again not to have become blocked by new obstacles, for example discovered by sensor measurements done after the path was originally planned.

If the collision check is passed and the path segment can be committed to, the root node is moved to the vertex at the end of the committed path segment, see figure 10 for an illustration. When doing so a number of nodes, and sometimes entire sub-trees branching from the old root node become obsolete. If the method has a limit on the maximum number of vertices, the number of new vertices available for planning is then increased.

If the path does not pass the collision check, nodes are instead deleted which are situated between the obstacle and the goal, including the goal nodes. After that the remaining goal nodes are re-examined to find the node with the lowest total cost, and a new candidate path segment to commit to is selected.

3.3.1 Continuous re-planning

During the time the vehicle is following a committed path segment the situation can occur that a new obstacle is discovered which blocks the selected path at some stage later than the currently committed path segment. What will happen then is simply that nodes downstream (closer to the goal) of the discovered obstacle in the planning tree will be deleted, and the number of vertices available for planning will be increased. The re-planning procedure is continuously performed so that the tree is always up to date with regards to new information from the sensors.

3.3.2 Special cases

There exist several special cases which the path selection must be able to handle. First there is the case when there are no potential path candidates leading to the goal point. Then a complete re-planning is performed and the bounding box for the sampling regions is increased in size in all dimensions by a factor. In this work 1.2 was used.

If the path planner tries to commit to a final path segment to the goal it is not uncommon with a quite long path segment, especially in the beginning of the planning. See for example figure 9 where the path segments leading to goal are much longer than the path segments in the original planning tree. Therefore, the path planner is allowed to split a path segment into two, when a final path segment is selected. It is split so that the committed path segment covers the distance up to right before the known and free space ends, and the new root node is placed there. The rest of the final path segment which then will become the only path segment belonging to the planning tree. Re-planning can then be done as usual.

4 Implementation

To test the different path planning methods, simulations are run using the robot operating system (ROS) integrated with the robotics simulation software Gazebo. A ROS node is created which continuously controls the three-dimensional position of a simulated drone model implementing realistic dynamic properties. This model was provided in advance to the thesis project. The set positions are obtained from the path planning module, which contains the main contributions of this work. The main ROS node begins to operate the drone at a starting point and is calling the path planner for new path segments to follow, until it reaches a goal point. Data from the run are stored in text files for further analysis.

4.1 High-level code blocks

The path planning software is implemented in C++, and the code has been divided up into a number of high-level code blocks, see figure 11. The main routine uses the path planner to obtain a path segment which it can follow. This path segment is passed on to the path follower which in turn will give out set point locations for the control of the drone.

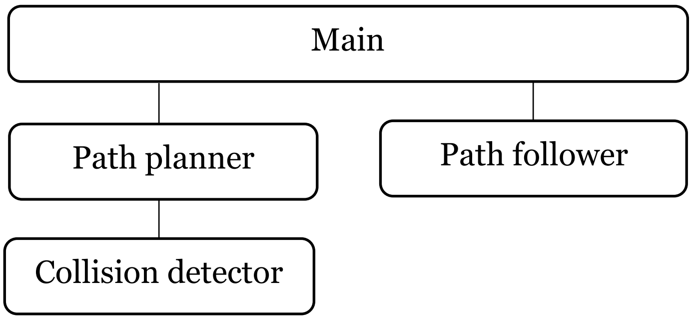

Fig. 11. Block diagram showing the hierarchy for the main building blocks of the code.

4.1.1 Path planner

The path planner is responsible for sampling new nodes in space and connecting them according to the algorithm which is used, and updating the tree if nodes become obsolete. The path planner also investigates if it is possible to reach the goal from each node which is added to the planning tree. If that is the case an additional vertex is created at the goal point with the associated total cost for the whole path.

The path planner further takes care of committing to path segments which consist of trajectories between the root node and the following node which the vehicle will follow. After committing to a path segment there will be a new root node assigned to the planning tree. Obsolete paths from the old root node are then deleted and a number of node objects are removed from the planning tree.

4.1.2 Collision detector

The collision detector class contains an instance of an OctoMap (see section 4.3) and can be used to check if path segments are colliding with obstacles, if path segments are entirely free, or if they contain parts which goes through unknown space but are otherwise collision free. To increase margins to obstacles offsets from the planned path are tested for collisions.

4.1.3 Path follower

The path follower is a supplementary class which is needed to run simulations. Path segments can be sent to this object together with continuous updates of the vehicle position. It then provides functionality for calculating setpoints for the control of the vehicle.

4.2 Nodes

The higher level blocks use lower level classes such as the Node class and its derived classes as building blocks. The path planner is implemented using a template class. Therefore, it is possible to use polymorphism in the type definitions of nodes in the planning graph and let them implement different functionality in their member methods.

For this study three different variants of nodes have been implemented, see figure 12. First a basic variant of node, which is used for vertices for the original variant of RRT, and then two derived classes which implement functionality for the RRT* algorithm and the RRT-u algorithm (see chapter 3).

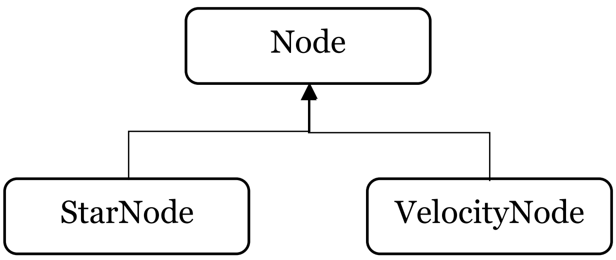

Fig. 12. Class dependencies for the basic datatypes

4.2.1 Node base class

The Node base class contains variables for a 3D position, the total accumulated cost for following the path from the root vertex to the current vertex. It also contains pointers to the parent and the children of the Node instance in order to describe the tree structure. The cost function implemented is simply the calculation of the

Euclidian distance (which is proportional to time given a constant velocity) from one vertex to another vertex in the planning tree. This will produce planning graphs which follow the original RRT formulation.

4.2.2 Star node class

The StarNode class is the node class that implements RRT* specific code. The RRT* algorithm can be formulated in an identical way as the RRT algorithm, except that it has a rewiring step which finds the nearest parent to the new node, and which also rewires nearby nodes to the new node if a lower total cost can be achieved. To accomplish that an empty virtual rewire method is included in the Node base class. This method is then overridden by the StarNode class to provide the rewiring functionality.

4.2.3 Velocity node class

The VelocityNode class contains in addition to a position, velocities and accelerations. The velocities are the values exactly at the position of the vertex. The accelerations are constant values used for the whole path segment leading up to the vertex. The cost function is overridden and implements the cost function as described in chapter 3, which is the time it takes to travel the path segment between the parent node and the node in question.

4.3 Classification of space using OctoMap

Space is divided up in cells consisting of cubic voxels with a uniform and preset size. The individual voxels are further classified into one of the three categories, free, occupied and unknown. Unknown space is the space which has been unexplored or where it cannot be determined from measurements if the space is free or occupied. This means that from the very beginning of the planning algorithm all space is unknown. After some time when the vehicle has been active, sensor data is gathered, and the state of the surrounding space is continuously updated. Occupied space is where the measurements are mainly confirming there is some obstacle, and free space is consequently where measurements mainly confirm there are no obstacles. This process can be handled in a probabilistic way where measurements increase or decrease the probability for the cells in space to be free or occupied. In this thesis the OctoMap [27] library has been used to keep track of measurement information and update the probabilities. The library also provides a query function with which points in space can be queried for their log odds of being occupied, providing a tool for detecting if a trajectory for a vehicle would lead to a collision with an obstacle or not. Voxels with log odds of below -0.4 are considered free space, above 0.7 they are classified as occupied space, and between -0.4 and 0.7 are unknown space. These values are the same as proposed by the authors of OctoMap in [27].

5 Experiments and Results

Four different test maps have been defined to test how well the method performs. For each map, the three different algorithm variants RRT, RRT*, and RRT-u are tested and compared. The test cases consist of predefined indoor environments with a specific layout and obstacle positions, designed to test different aspects of the planning algorithms. The task for the vehicle is to travel from a given start point to a given goal point in the most efficient way. The simulated UAV is not provided with any information about the environment at the start, so it will therefore need to explore the map and react to obstacles which are discovered on the way to the goal. Since the methodology is probabilistic each experiment was repeated ten times per algorithm.

The dynamic resizing functionality was furthermore tested by running each algorithm three times on each test case without the resizing functionality turned on and observing the result, to check if the efficiency of the algorithms is affected by this feature.

The only sensor which is currently used to detect the surroundings is a simulated realsense RGB-D camera from which depth data can be obtained up to around 5 meters in front of the drone.

All the environments have a floor and a ceiling. Since the camera has a limited field of view, mapping of space straight above and straight below the drone is not done very well, in contrast to in-plane space which is more easily mapped. The result is that paths are commonly proposed to go up into the ceiling since that is unexplored environment with no obstacles being detected there. The solution chosen here, for simplicity is not to let the drone go above and below specified vertical limits, roughly corresponding to the floor and ceiling.

5.1 Performance Metrics

Several different performance metrics are used to monitor the performance and to find differences between the different variants.

- **Planned path length**. This shows how efficient the path planner is to find a path close to the optimal path. It can for instance be expected that the RRT* will have a shorter planned path length than the RRT.
- **Total elapsed time**. This metric is the time it takes to complete the whole path from start to goal. It is correlated to the path length since the velocity profile of the path segments has been designed to be the same for the different planning methods. However, the actual dynamics of the simulated drone may cause some difference between the two metrics, if for instance there are more sharp turns in the planned path of one algorithm than another.
- **Number of resizes**. The number resizes shows how many times the planning algorithm has not found any potential path leading to the goal, and

thus it must restart the planning algorithm using a larger bounding box to sample from.

- **Accumulated planning time**. This gives the total planning time including collision checking for the whole run.
- **Planning time per vertex**. This is calculated as the average over the whole run. It will give a higher value the more complex the algorithm is. The smaller this time is the higher number of vertices in the planning can be used.

5.2 Simulation Parameters

There are several parameters which need to be fixed before the simulation starts. The maximum number of vertices is 100 for all test cases. The maximum planning time is set to 100 ms, except for the first planning occasion or when a reset of the planning tree is needed. Then a planning time of 1000 ms is allowed. The “q” parameter which determines the maximum length has a value of 1.0 m, and the “eta” which is the neighborhood radius of the RRT* algorithm is set to 2.0 m. Maximum velocity is 0.3 m/s and max acceleration is 0.2 m/s^2.

5.3 Test case 1: Empty map

The empty map contains just a start point and a goal point and no obstacles, see figure 13. The purpose of this test case is to be a sanity check for the different path planning algorithms. They should roughly have the same performance and they should produce an approximately straight line from the start to the goal.

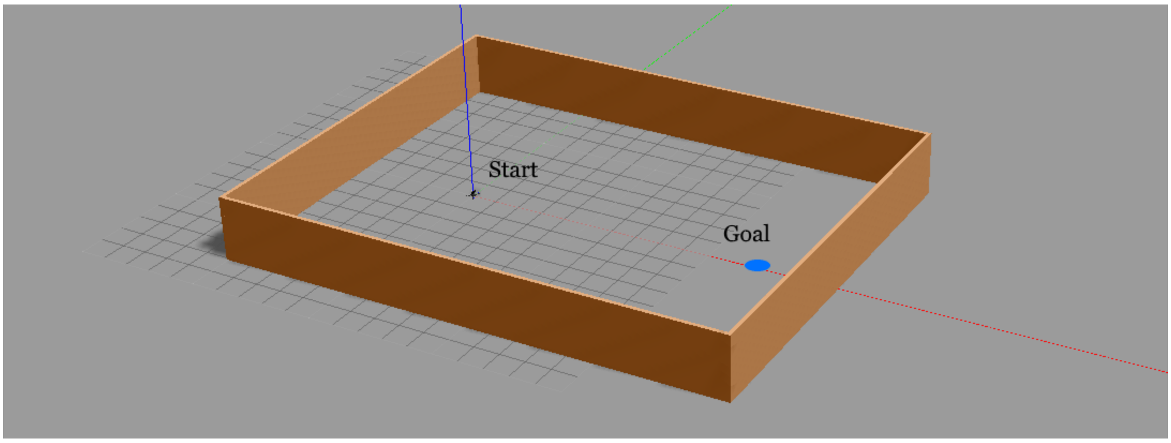

Fig. 13. The empty map test case environment with the top layer of the construction removed.

5.3.1 Results RRT

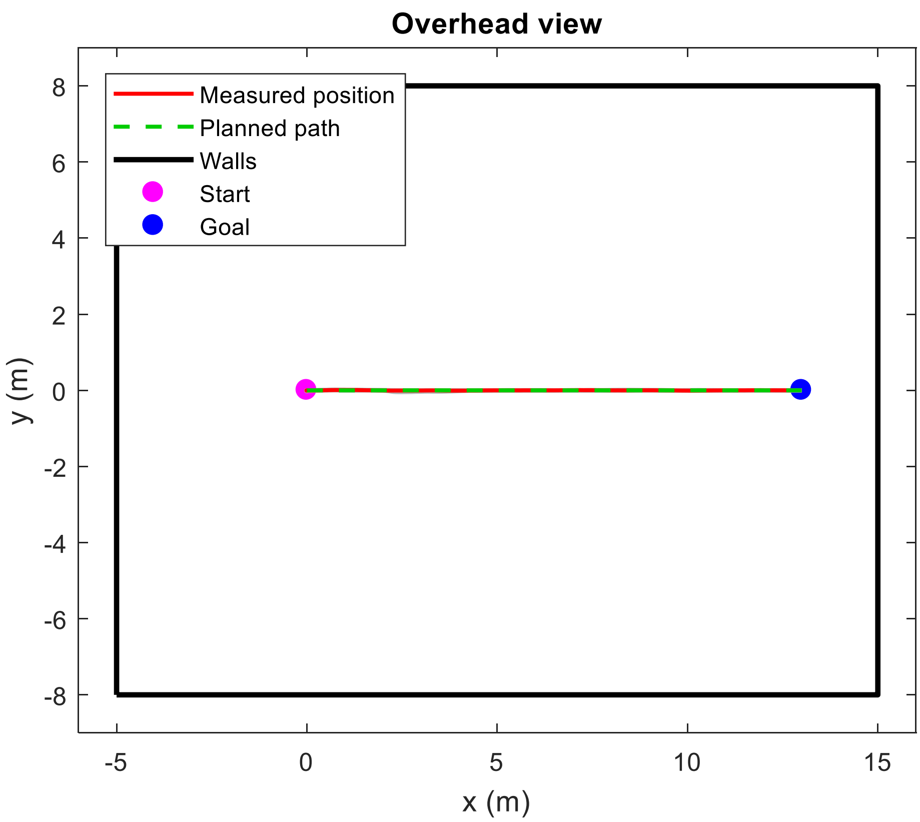

Fig. 14. The path using the RRT algorithm in the empty map.

The resulting ten paths of the RRT run are shown in figure 14. As can be expected they are all virtually identical to a straight line. That is because the vehicle enters a goal path directly at the start point since the goal path in this case represents the fastest way to reach the goal location (see the gray path segments of figure 9).

5.3.2 Results RRT*

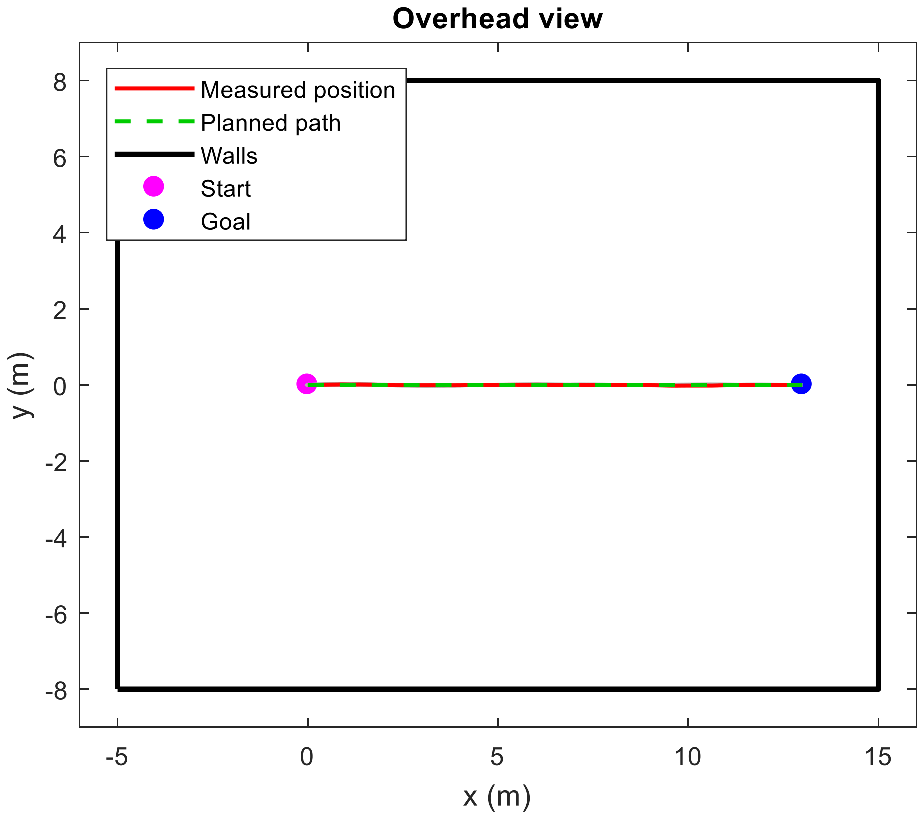

Fig. 15. The results for the RRT* in an empty map.

The paths for the RRT*, shown in figure 15, look very similar as for the RRT case, which can be expected since there is no need to do rewiring going for this test case.

5.3.3 Results RRT-u

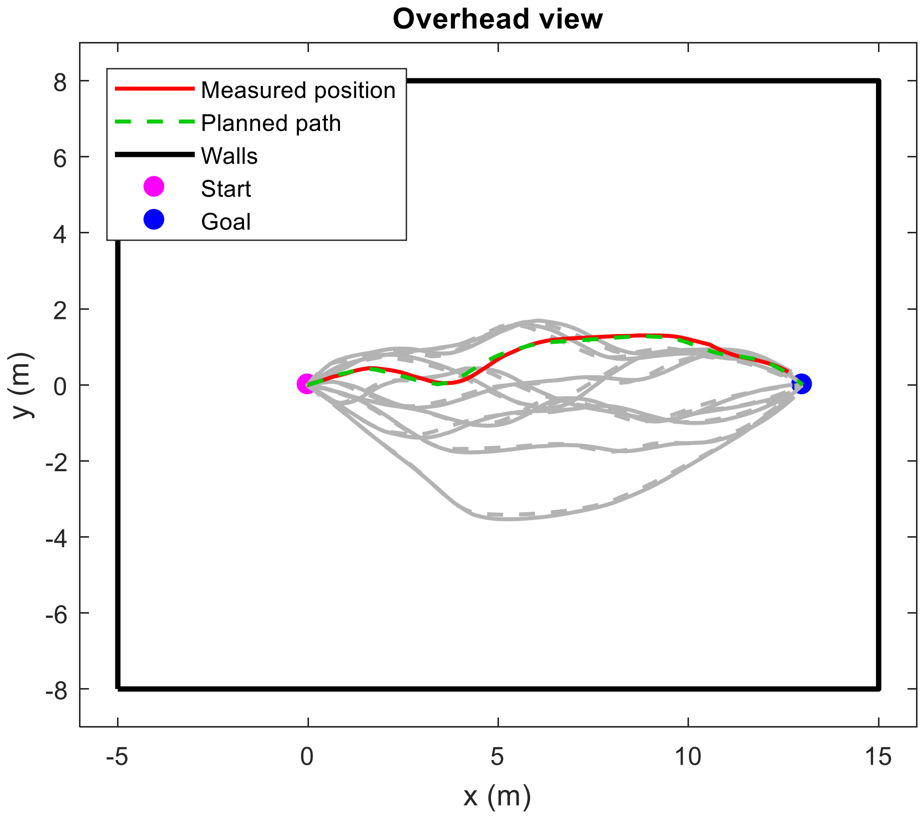

Fig. 16. Resulting paths for the RRT-u case in the empty map.

The planned and the measured paths for the RRT-u, shown in figure 16, are not as straight as for the RRT and RRT* case. This is because a goal path is usually curved, due to the direction of the velocity vector of the vertex it is connected to. To go straight to the goal without passing any nodes is not a very good solution for the RRT-u, because if the distance to the goal node is sufficiently large, the average velocity will then become half of the maximum velocity, and the acceleration to achieve that will be rather small. This is different compared to the RRT and RRT* algorithms, which have constant velocity over the whole path.

5.3.4 Summary for all methods for the empty map test case

Parameter	RRT	RRT*	RRT-u
Planned path length [m]	13	13	14.21
Total elapsed time [s]	39.378	39.431	39.664
Number of resizes	0	0	0
Accumulated planning time [s]	1.582	1.589	4.769
Per vertex planning time [ms]	3.195	3.209	4.884

Table 1. Summary of the metrics for the empty map, taking the average over ten runs.

The summary of the metrics for the empty map test case is shown in table 1. The RRT and the RRT* show very similar performance, when it comes to planned path length and total elapsed time. The RRT-u algorithm has slightly worse performance on most parameters, which can be expected from seeing the curved paths in figure 16. It is a consequence of the dynamic constraints applied. No resizes are needed for any of the algorithms since there are no obstacles in the empty map.

5.4 Test case 2: Double slit

The double slit map, see figure 17, consists of two large compartments divided by a wall with two slits in it, and the drone can choose to pass either one of the openings on its way to the goal. The wall is far enough away from the starting position so that the drone cannot sense that there is a wall in front of it without first moving some distance forward.

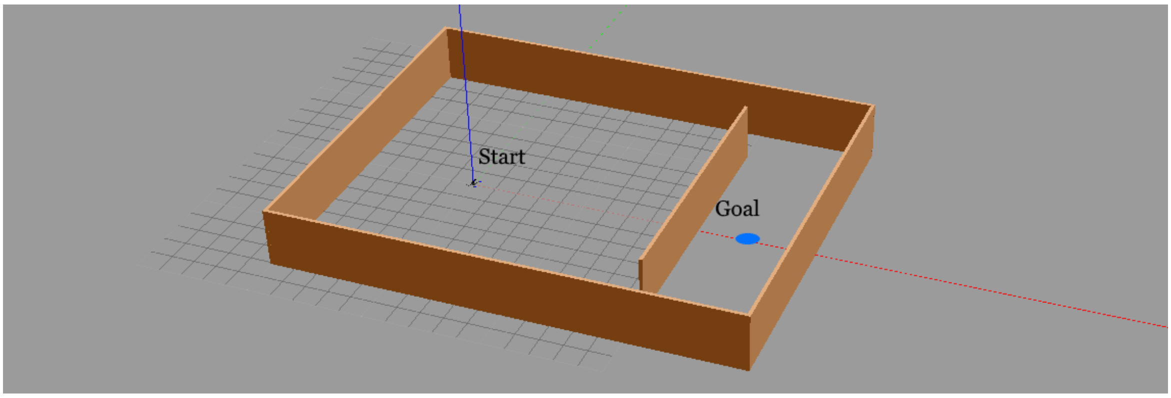

Fig. 17. Building layout for the double slit test case.

5.4.1 Results RRT

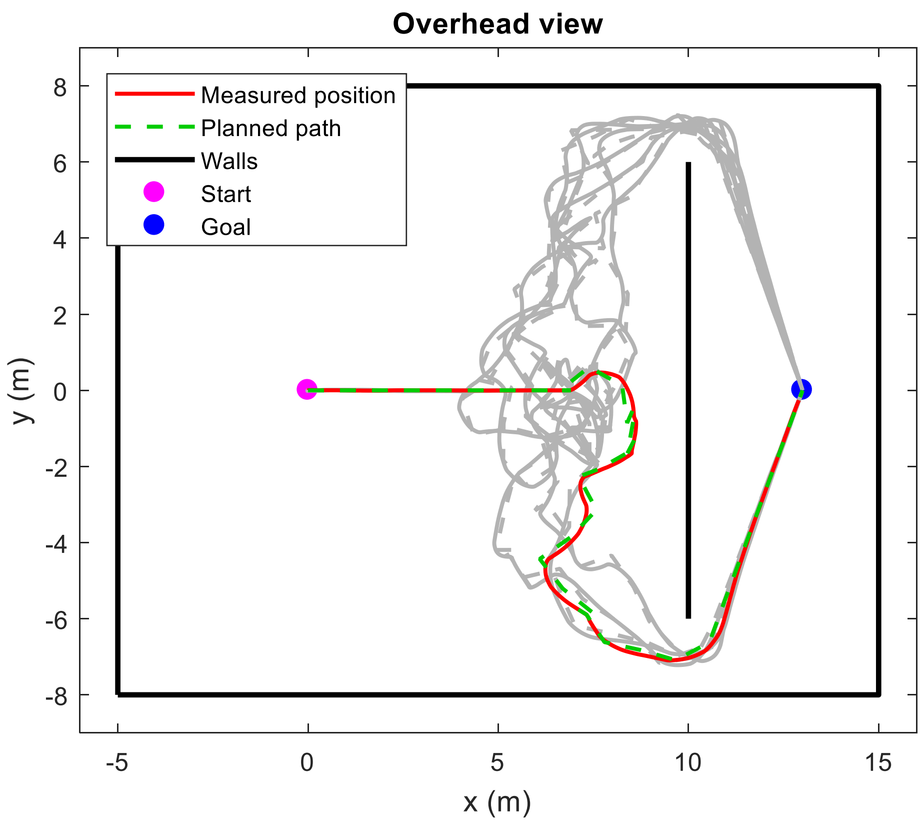

Fig. 18. Results for the double slit experiment using RRT.

The RRT algorithm produces paths which first move straight to the goal, see figure 18. This is before any obstacle has been detected. After the wall is discovered by the camera, the vehicle gradually finds potential paths further and further away from the center, until it finds a path which does not eventually become blocked by the wall from further sensor measurements, and thus can be used to go through a slit and towards the goal.

5.4.2 Results RRT*

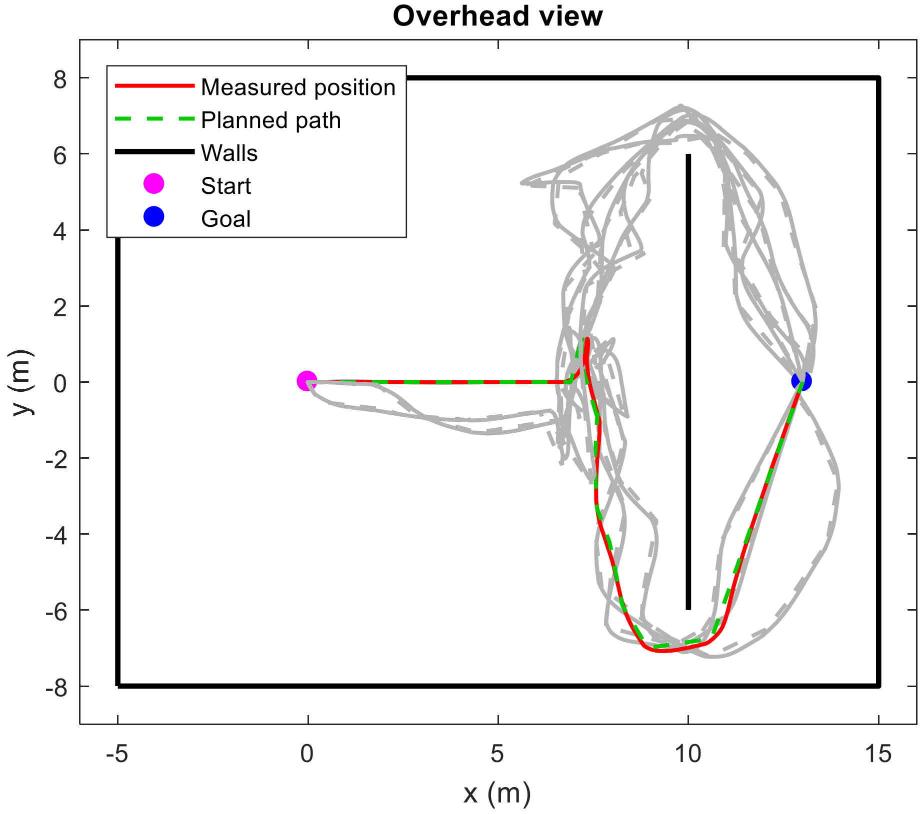

Fig. 19. Results for the double slit experiment using RRT*

In figure 19 the results for the RRT* algorithm are shown. The results are slightly different due to the rewiring efforts of the RRT* algorithm. It is possible that with more careful selection of the algorithm parameters even better results would have been obtained.

5.4.3 Results RRT-u

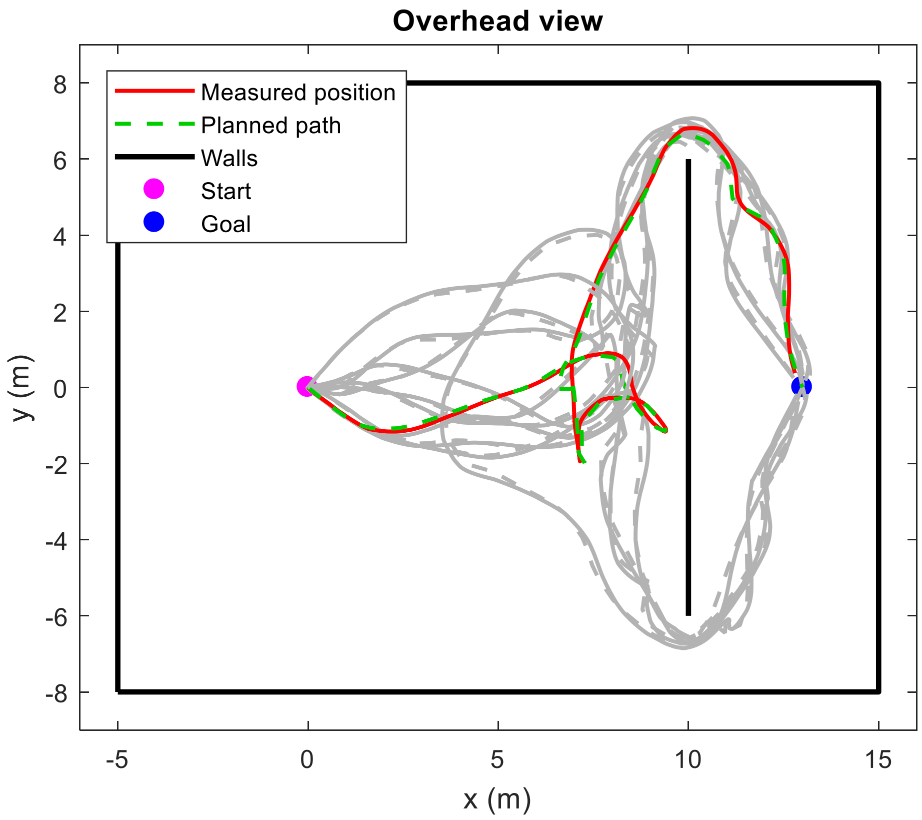

Fig. 20. Results for RRT-u in the double slit test case.

For the RRT-u path planner in the double slit test case, shown in figure 20, the paths make loops and many turns. It is because as the sensor discovers more and more of the wall in one direction, the path planner believes it can find a better route in the other direction and then makes a turn. It could possibly even make more turns back and forth if the blocking wall would be even wider, but here it eventually finds one of the slits and can continue towards the goal. This type of behavior is not present for the RRT and the RRT* algorithms. It is because the sampling region in these simulations is typically small compared to the size of the discovered obstacle. Therefore, the RRT and RRT* algorithms don't turn back, simply because they cannot find a potentially shorter path in path backward direction. The RRT-u has an ability to plan paths around obstacles and has therefore easier to find a path in the other direction, and then try that.

5.4.4 Summary of metrics for the double slit test case

Parameter	RRT	RRT*	RRT-u
Planned path length [m]	31.959	29.277	27.956
Total elapsed time [s]	94.395	88.48	81.35
Number of resizes	4.8	4	1
Accumulated planning time [s]	11.933	10.836	15.936
Per vertex planning time [ms]	6.928	6.261	6.975

Table 2. Summary of the metrics for the double slit experiment, taking the average over ten runs.

In table 2 the values for the different methods of the double slit are summarized. For this test case the RRT-u is the fastest to reach goal and the planned path length is the shortest on average. RRT and RRT* does resizing of the sampling area several times while the RRT-u algorithm does that only one time, which confirms that the RRT-u algorithm is better to plan around corners of obstacles. The differences in planning times are rather small even though it can be noted that the RRT* has the shortest planning time, even though it is not the simplest algorithm. The reason for that is not clear, but possibly it can have something to do with how the algorithm scales, and how many nodes there are in the graph on average.

5.5 Test case 3: Vertical Up & Down

The Up & Down map, see figure 21, consists of a series of obstacles which either are slabs standing up 1.5 m from the floor or 1 m slots near the floor. The total height of the construction is 2.5 m. The planning algorithms will therefore need to plan in the vertical dimension here to construct a path from the start to the goal.

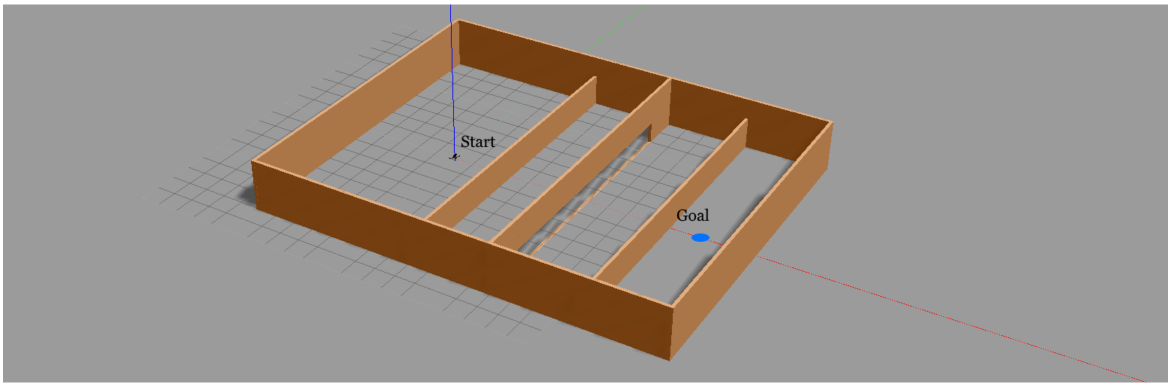

Fig.21. The layout for the vertical Up & Down test case.

5.5.1 Results RRT

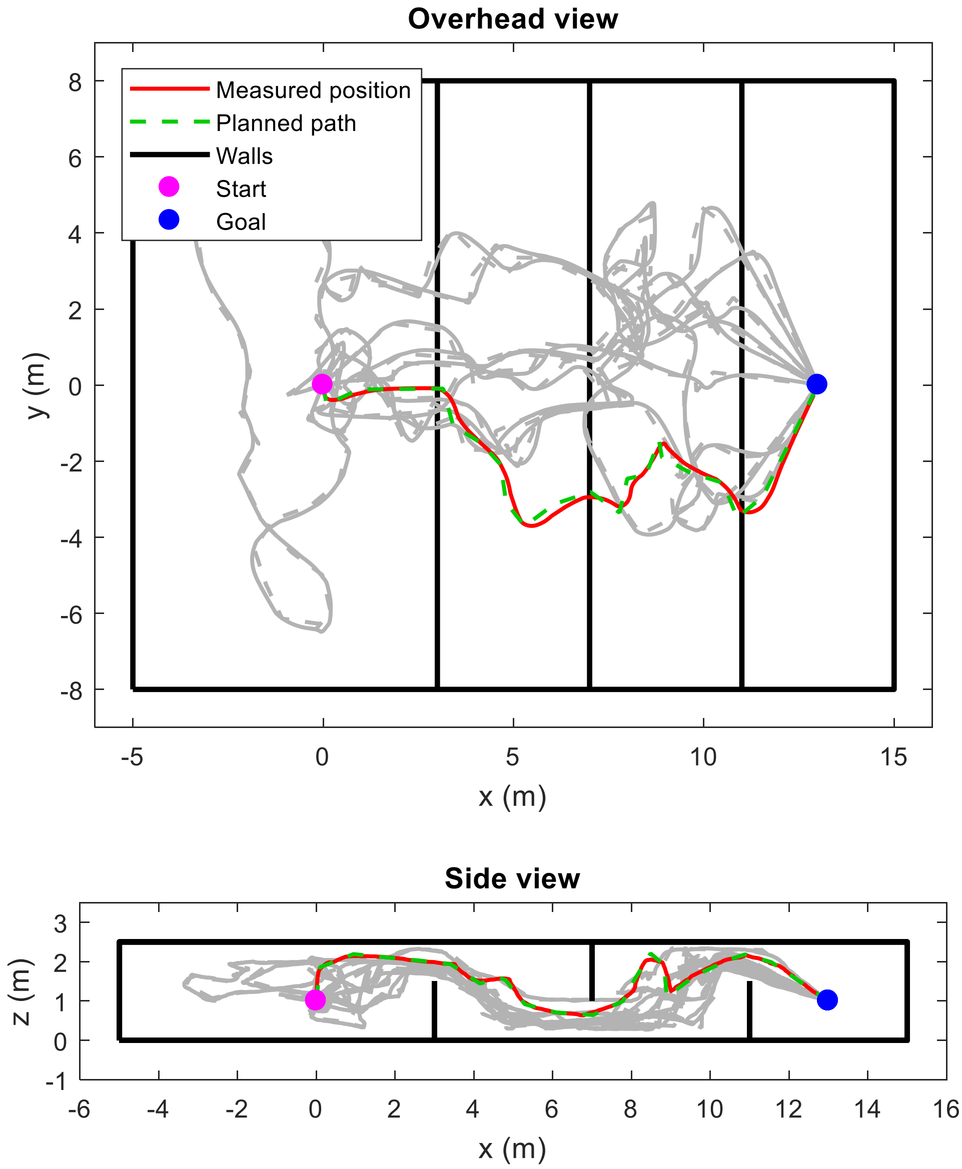

Fig. 22. Results for the RRT in the Up & Down test.

The resulting paths for the RRT algorithm are shown in figure 22. The algorithm manages to produce paths from start to goal, but the paths are not very smooth and therefore significantly longer than for the optimal case. The RRT does not do any rewiring, so the sampling procedure becomes rather important in determining the actual paths. This type of construction is many times wider than it is high, why the samples spreads out more in the lateral direction than in the vertical direction.

5.5.2 Results RRT*

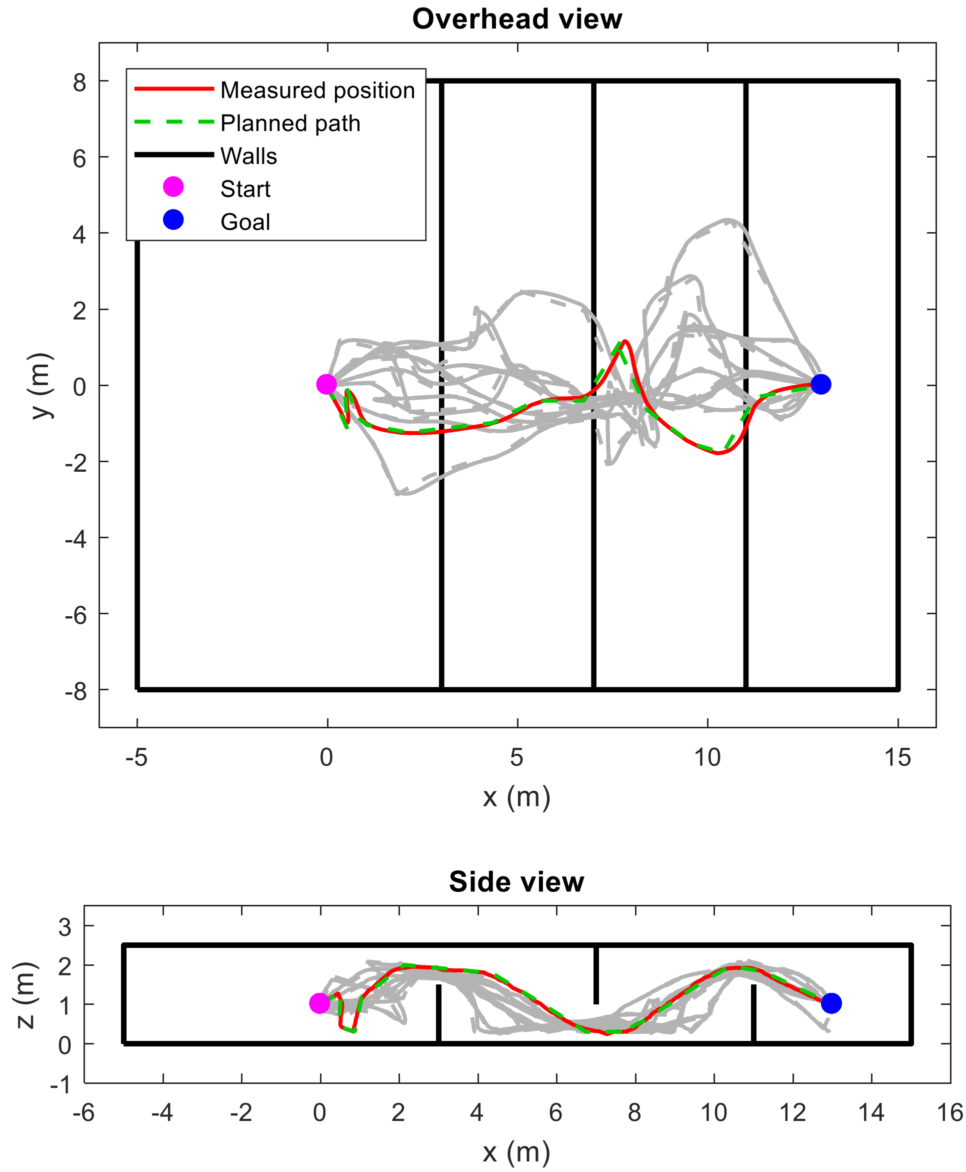

Fig. 23. Results for the RRT* in the Up & Down test.

The results for the RRT* algorithm, see figure 23, shows a smoother and closer to optimal curve than for the RRT algorithm. This is because the rewiring step removes unnecessary movement in the y-direction when possible.

5.5.3 Results RRT-u

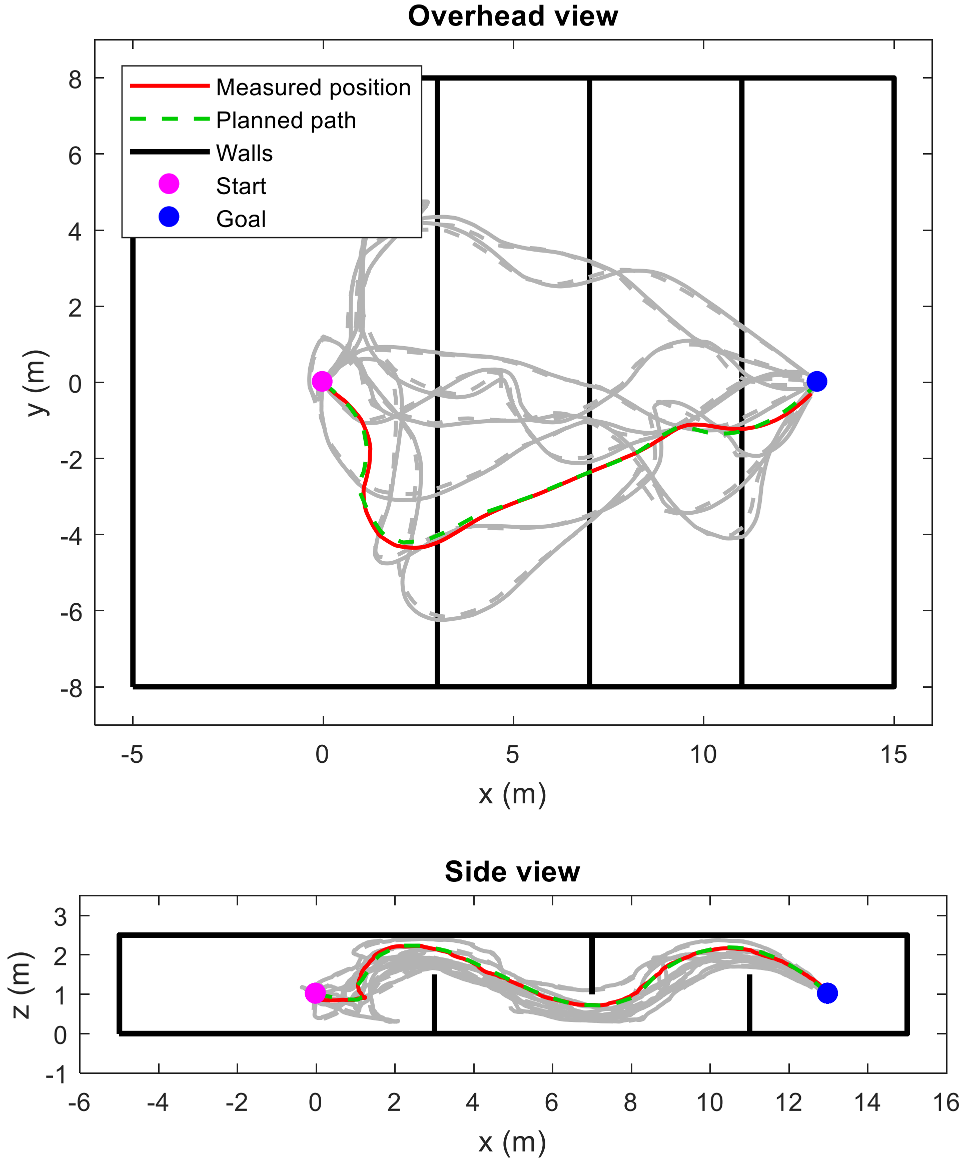

Fig. 24. Results for the RRT-u in the Up & Down test.

Figure 24 plots the results for the RRT-u algorithm. This algorithm suffers here because it does not have a rewiring step as the RRT* has. The curves are rather smooth though and perhaps with other parameter settings allowing sharper turns a better result can be obtained, since then the vehicle can the turn faster towards the goal.

5.5.4 Summary of metrics for the Up & Down test case

Parameter	RRT	RRT*	RRT-u
Planned path length [m]	29.097	20.269	20.213
Total elapsed time [s]	82.894	60.433	56.791
Number of resizes	3.7	0.9	0.4
Accumulated planning time [s]	12.517	5.62	14.368
Per vertex planning time [ms]	6.372	3.952	7.422

Table 3. Summary of the metrics for the vertical Up & Down test, taking the average over ten runs.

The results for the Up & Down map are listed in table 3. Once again the RRT-u has the shortest planned path length and the shortest average time to reach the goal. The resizing of the sampling area is close to zero, since the obstacles are not very large for this test case. When it comes to planning time, the RRT* has the shortest overall planning time while and the shortest planning time when normalized with the total number of vertices used for completing the test case.

5.6 Test case 4: The trap

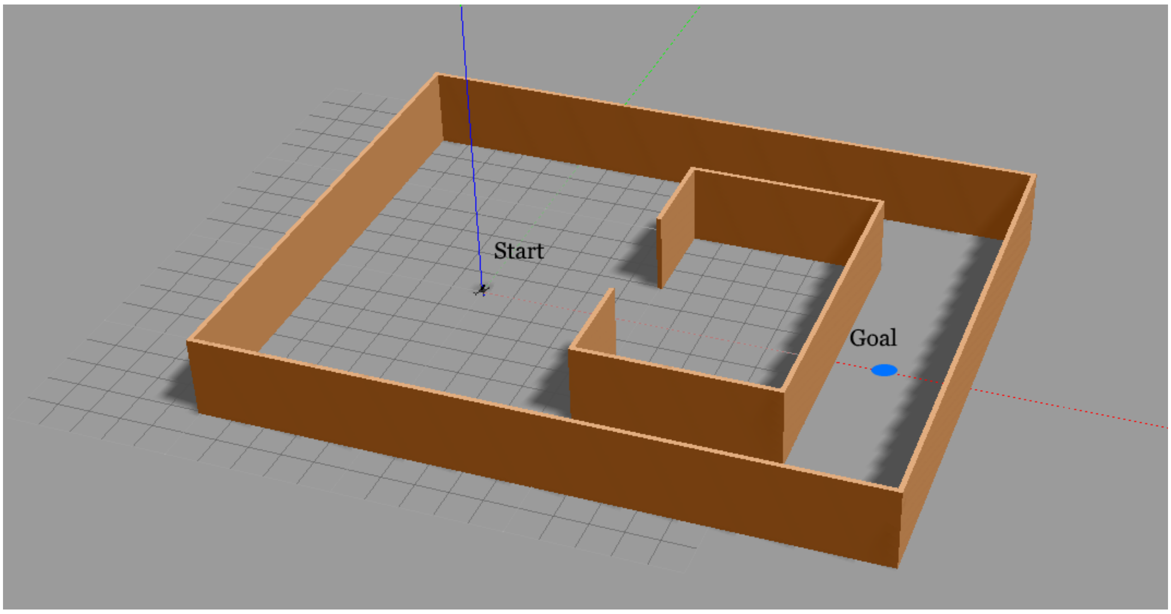

Fig. 25. The setup for the “The trap” experiment.

In figure 25, the “The trap” setup is shown. It is designed to be the most difficult map to complete for the different algorithms. The heuristics will guide the vehicle into the enclosing, and the drone will there need to detect the different walls to understand that there are no potential paths to the goal from within the enclosing. Then the algorithms will need to find a path out from the enclosing, and around towards the goal, which usually is possible with the default size of the sampling region. After it has exited the enclosing the vehicle is rather far from the goal and there is a detected obstacle in front of the goal hindering straight goal paths to be created from far away. That is usually when the majority of the dynamic resizing occurs.

5.6.1 Results RRT

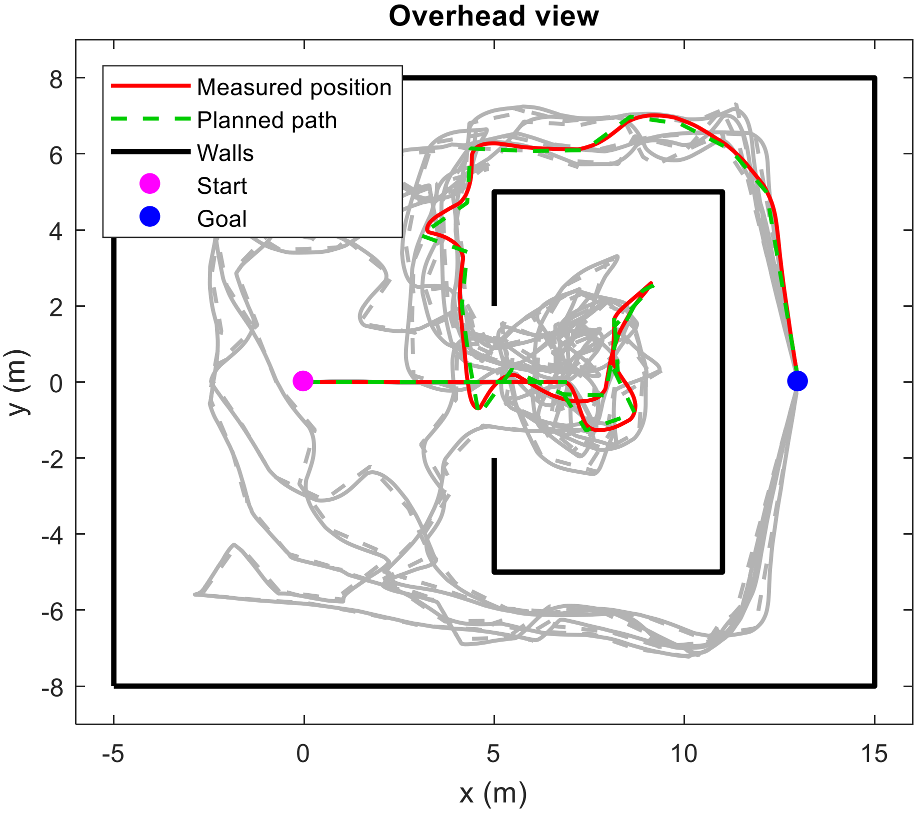

Fig. 26. Results for the RRT in the “The trap” test.

The results for the RRT algorithm are shown in figure 26. After discovering no possible paths to the goal from within the enclosing the vehicle manages to find paths on the outside and navigate itself towards the goal.

5.6.2 Results RRT*

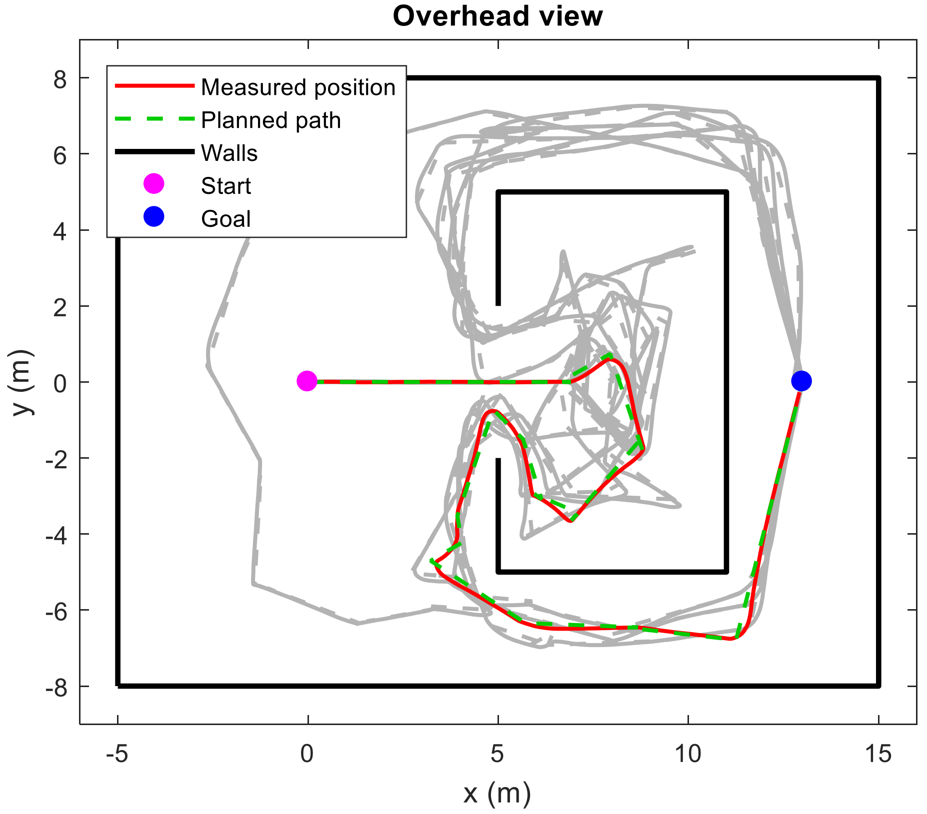

Fig. 27. Results for the RRT* in the “The trap” test.

The results for the RRT* algorithm, see figure 27, are in many ways similar to the ones for the RRT algorithm, except the path looks slightly more efficient.

5.6.3 Results RRT-u

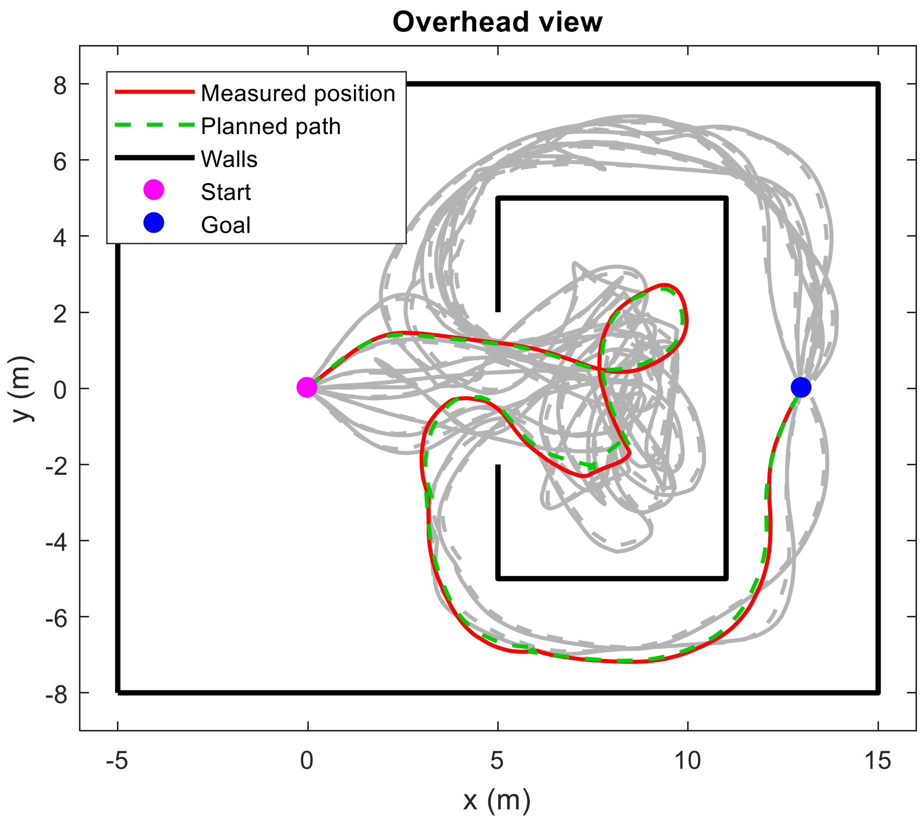

Fig. 28. Results for the RRT-u in the “The trap” test.

The results for the RRT-u algorithm is shown in figure 28. The general impression is that the planning looks to be a little bit more curved than the RRT and the RRT* algorithms. Overall it has a rather consistent performance of producing paths which lead to goal.

5.6.4 Summary of metrics for the “The trap” test case

Parameter	RRT	RRT*	RRT-u
Planned path length [m]	54.95	45.704	46.251
Total elapsed time [s]	175.235	143.765	146.025
Number of resizes	19.6	11.7	4.4
Accumulated planning time [s]	26.88	19.447	33.551
Per vertex planning time [ms]	6.835	6.213	12.518

Table 4. Summary of the metrics for the “The trap” test, taking average over ten runs.

For the trap test case, see table 4, the RRT* has the shortest path length on average and the shortest time to complete the task. For this test case the number of resizes of the sampling are is rather high. Here the RRT-u has the lowest number, most likely because it can plan by using curved path segments which can reach the goal point by passing obstacles on either side. Furthermore, the RRT* also has the lowest planning time, both total and per vertex.

5.6.5 Bar diagrams showing total elapsed time for the different test cases

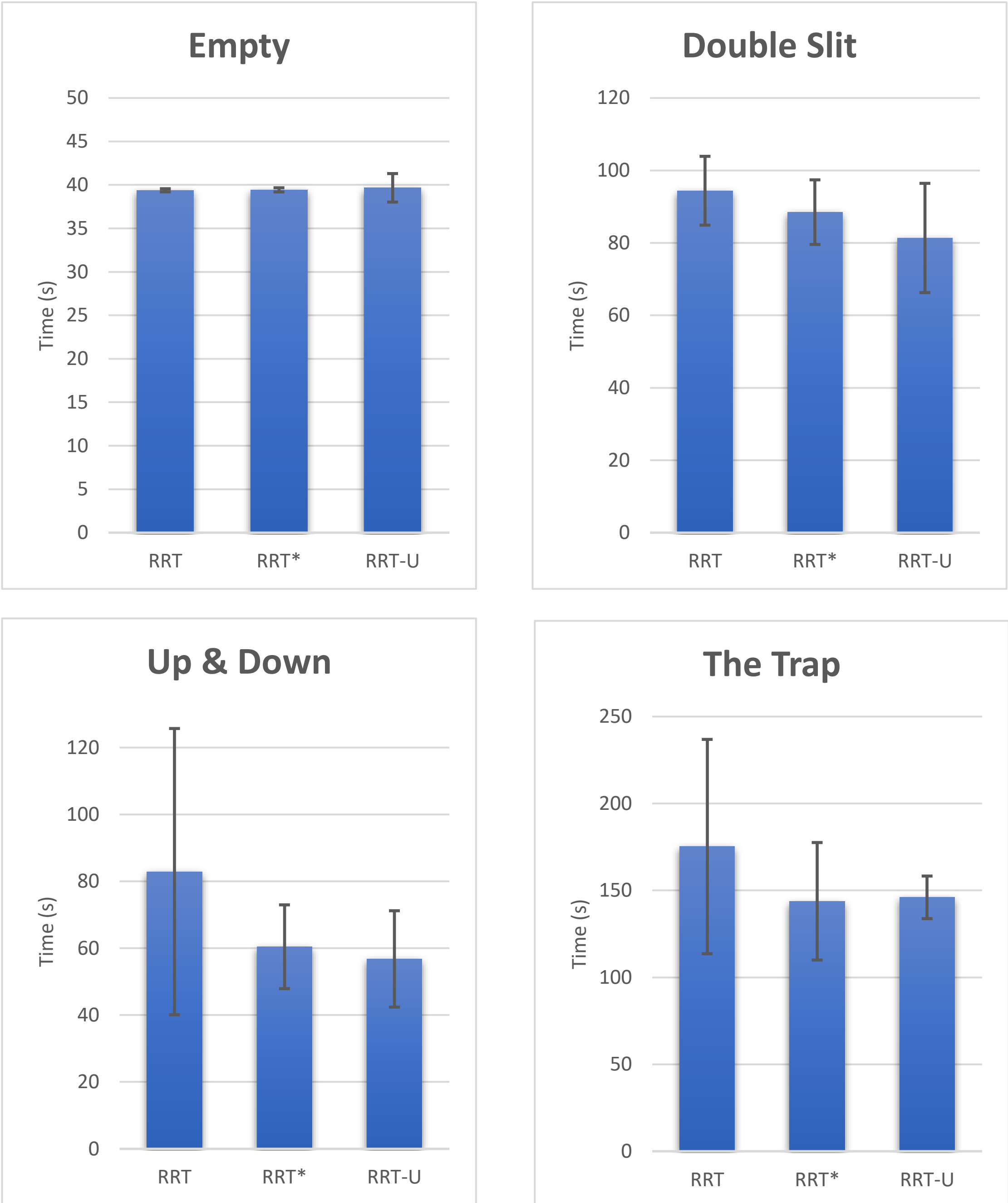

Fig. 29 Bar diagrams showing the average time to travel from start to goal for the different test cases. The standard deviation for 10 runs is shown in the error bar.

The total elapsed time for all four test cases are shown in in figure 29. The average time it takes to complete the map is shown using blue bars. The standard deviation of all 10 runs for each algorithm variant is displayed in the error bars. For the empty test case the time is very similar, and the standard deviation is very small except for the RRT-u algorithm, because that variant did not result in paths that went straight from start to goal as the other two algorithms did. The double slit and the up & down test cases show the RRT-u algorithms as the fastest with the RRT* on second place. The up&down test case has quite large difference between RRT and the two other variants and the standard deviation is larger because there is a larger lateral uncertainty in the chosen paths. For the trap test case the RRT* is slightly faster than the RRT-u algorithm. To summarize, the difference between the algorithm variants is not very large and the best algorithms are the RRT* and the RRT-u.

5.7 Testing the effect of dynamic resizing

Dynamic resizing of the sampling region is proposed to handle the case when no preliminary paths to the goal exist. At the start of the path planning a nominal size for the sampling region is assumed. When the case of no preliminary paths occurs the sampling volume is increased gradually until the algorithm is able to find a path to travel towards goal again. The effect of resizing, or rather the effect of not having the resizing functionality turned on, is tested here by running each algorithm for each of the prepared test cases three times and observe if and why problems occur. In table 5 the results are summarized, and there are three cases where the lack of dynamic resizing causes problems. First there are some minor problems for the RRT algorithm running the double slit test case. A few times the planning graph needs to be completely re-planned because no potential paths are found leading to the goal. This is likely due to the rather wide obstacle which is in front of the goal in the double slit test case. Eventually, after several re-plannings, the algorithm was able continue anyway, and the vehicle reached the goal point. But for "The trap" test case both the RRT and the RRT* variants completely fail repeatedly. This is most likely because there is a detected obstacle in front and on the sides obscuring the goal point and when the simulated vehicle needs to go back out from the enclosing, the sampling region is not large enough to reach around the obstacle. No preliminary paths can then be planned to the goal, and if there are no potential paths the vehicle does not know what to do. The RRT-u algorithm can handle the lack of resizing better, since it can plan around obstacles, using curved path segments.

Test case	RRT	RRT*	RRT-U
Empty	+	+	+
Double slit	-	+	+
Up & down	+	+	+
The trap	-	-	+

Table 5. Summary of the results for running test cases without dynamic resizing turned on. A plus means the execution went without problem, and a minus that some problem was encountered.

6 Sustainability, Ethics and Societal impacts

This is a thesis in the field of robotics and it deals with the subject of how to improve the autonomous capabilities of a vehicle. As autonomous technology improves, it will become more common with machines carrying out advanced tasks entirely without human assistance. This will among other things help to make certain services to become economically feasible so that more people can use them. Take for instance autonomous cars as an example. Then a machine is driving a vehicle and one or more humans are riding the vehicle, or some goods is transported. This service is hardly new. For almost as long as there have been cars there have been humans working as taxi drivers, bus drivers, and truck drivers, which probably are the closest job descriptions one could find to what an autonomous car provides. So, what will happen when these jobs get competition from autonomous technology. If cars driven by machines become cheap enough the value proposition will be more attractive to many persons, and more people end up enjoying the service of not having to drive the car themselves, and the cost of transporting goods can be decreased. However, there is another aspect of this, which is that the people employed in these types of jobs will possibly all lose their source of income, and perhaps will have a hard time finding another job. This will affect their overall life situation for the worse, and perhaps also the family members of them will also suffer from this development. It is thus clearly not easy to judge if it is good or bad from an ethical point of view to promote the development of autonomous technology. It is even more so because it depends on so many other things in society that will be affected as well, such as the distribution of wealth to take an example. If society gives economical compensation to people who lose their jobs due to autonomous technology and creates opportunities for free education with the goal for unemployed to learn a new profession, then the equation looks more favorable for the case of autonomous technology. Also, if this support program is successful, after some time the pendulum might be swinging the other way. Autonomous technology will not be seen as something that takes jobs away. Instead it will be viewed upon as a liberator, which frees people which are otherwise caught in a servant type of employment, or jobs which are otherwise not very stimulating. The future will tell the answer for this particular question. There are of course other challenges related to jobs and autonomous technology. The impact of the new technology can for instance be so enormously big that it will be a problem for society as a whole to cope with. In that case it would probably be wise for the society to impose regulations that will slow down and facilitate the introduction of autonomous technology.

7 Conclusions

A method for dynamic path planning in unexplored environments in three dimensions has been presented and investigated. The method has been tested with tree different path planning algorithm variants, the RRT, the RRT* and the novel RRT-u algorithm. The different path planning algorithms were further extended by adding to the planning tree, if possible, a preliminary connection to the goal point from each new vertex. This simplifies the selection of an optimal path when unexplored regions exist between the vehicle and the goal point, since there is an estimated total cost associated with each branch of the planning tree, and the path with the potentially lowest cost can be selected as the currently most optimal path.

All three path planning algorithms were found to work well in a variety of circumstances, and no problems were found in terms of execution speed. From the results of running four different test cases it could be concluded that the RRT-u and the RRT* performed the best, compared to the RRT. The RRT-u method has the benefit of including dynamic constraints, even though it is a rather simple dynamic model. A big advantage with the presented RRT-u algorithm is that it is very fast to compute the parameters for a trajectory, and the planning time difference compared to the other methods is very small.

Dynamic resizing of the sampling region was proposed to keep the sampling region at the nominal size during normal operation for performance reasons, and to increase the size of the sampling region when problems in the planning occurred such as when no preliminary path to the goal could be found. It was found that for certain types of map configuration where there was a big obstacle in front of the goal, especially the RRT and the RRT* algorithm benefitted from using dynamic resizing. Without resizing the vehicles were unable to find a path to the goal, but with the resizing functionality turned on, paths were found.

It is believed that with further optimization of the software and with sensor hardware that is better at mapping on all sides of the vehicle, the proposed methods could work even better and be useful in a product where autonomous navigation is required. Furthermore, the RRT-u algorithm seems to be an interesting concept and there are probably situations where it could be useful, even though some more development and optimization is probably needed.

7.1 Future work

There are several different items which could be subject to further studies. Many of them concern the study of performance of the presented algorithms. For instance, varying the maximum number of nodes in the planning tree would be interesting to look at, and there are other parameters which can be varied.

The dynamic resizing could perhaps be compared against or combined with other approaches. Another possible approach which was not investigated is to remember the traveled path and to back-track that path when no preliminary paths to the goal exist.

Also, in the presented study there are only static obstacles in the environment. The addition of dynamic obstacles would be a valuable improvement.

As a potential improvement to the method, one could perhaps continuously refine the planning tree. If there is time left in a planning iteration, leaf nodes from which it is not possible to plan a path to the goal location could possibly be removed from the graph to allow for the addition of more new vertices.

The convergence properties and dynamics of the different algorithms are also of interest to find the algorithm which has the best potential to be developed further.

Finally, it would of course be interesting to verify how well this method works in real world tests.

8 References

[1] F. Giones and A. Brem, "Fom toys to tools: The co-evolution of technological and entrepeneurial developments in the drone industry," *Business Horizons,* pp. 875-884, 2017.

[2] S. M. Lavalle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," *TR 98-11, Computer Science Dept., Iowa State University,* 1998.

[3] S. Karaman and E. Frazzoli, "Incremental Sampling-based Algorithms for Optimal Motion Planning," *http://arxiv.org/abs/1005.0416,* pp. 1-20, 2010.

[4] L. E. Kavraki, P. L. J.-C. Svstka and M. H. Overmars, "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces," *IEEE Transactions on robotics and automation,* pp. 566-580, August 1996.

[5] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs," *Numerische Mathematik,* pp. 269-271, 1959.

[6] P. E. Hart, N. J. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transaction of systems science and cybernetics,* pp. 100-107, July 1968.

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

[8] S. LaValle, Planning Algorithms, Cambridge University Press, 2006.

[9] S. LaValle and J. Kuffner, "Randomized Kindodynamic Planning," *The International Journal of Robotics Research,* pp. 378-400, 2001.

[10] S. LaValle and J. Kuffner, "Randomized Kinodynamcic Planning," *IEEE International Conference on Robotics and Automation,* pp. 473-479, 1999.

[11] C. Sprunk, "Planning Motion Trajectories for Mobile Robots Using Splines," Albert-Ludwigs-Universität , Freiburg, 2008.

[12] D. A. Dolgov, S. Thrun, M. Montemerlo and J. Diebel, "Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments," *The International Journal of Robotics Research,* vol. 29, no. 5, pp. 485-501, 2010.

[13] M. Altaher and O. Nomir, "Euler-Lagrange as Pseudo-metric of the RRT algorithm for optimal-time trajectory of flight simulation model in high-density obstacle environment," *Robotica, 2017, Vol.35(5),* pp. 1138-1156, 2017.

[14] D. J. Webb and J. v. d. Berg, "Kinodynamic RRT*: Optimal Motion Planning for Systems with Linear Differential Constraints," *arXiv: Robotics,* vol. , no. , p. , 2012.

[15] R. E. Allen and M. Pavone, "Toward a real-time framework for solving the kinodynamic motion planning problem," *IEEE International Conference on Robotics and Automation,* pp. 928-934, 2015.

[16] S. Karaman and E. Frazzoli, "Optimal Kinodynamic Motion Planning using Incremental Sampling-based Methods," *49th IEEE Conference on Decision and Control,* pp. 7681-7687, 2010.

[17] B. Luders, S. Karaman, E. Frazzoli and J. How, "Bounds on Tracking Error using Closed-Loop Rapidly-Exploring Random Trees," *American Control Conference,* pp. 5406-5412, 2010.

[18] J. Kuffner and S. LaValle, "RRT-Connect : An Efficient Approach to Single-Query Path Planning," *IEEE International Conference on Robotics & Automation* , pp. 995-1001, 2000.

[19] M. Clifton, G. Paul, N. Kwok, D. Liu and D.-L. Wang, "Evaluating Performance of Multiple RRTs," *IEEE/ASME International Conference on Mechtronic and Embedded Systems and Applications,* pp. 564-569, 2008.

[20] D. Li, Q. Li, N. Cheng and J. Song, "Extended RRT-based Path Planning for Flying Robots in Complex 3D Environments with Narrow Passages," *IEEE International Conference on Automation Science and Engineering,* pp. 1173-1178, 2012.

[21] S. Karaman, M. R. A. Walter, E. Frazzoli and S. Teller, "Anytime Motion Planning using the RRT∗," *International Conference on Robotics and Automation (ICRA)* , pp. 1478-1483, 2011.

[22] H. Umari and S. Mukhopadhyay, "Autonomous Robotic Exploration Based on Multiple Rapidly-exploring Randomized Trees," *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) September 24–28, 2017, Vancouver, BC, Canada,* 2017.

[23] M. Otte and E. Frazzoli, "RRTX: Real-Time Motion Planning/Replanning for Environments with Unpredictable Obstacles," in *Algorithmic Foundations of Robotics XI*, 2015.

[24] K. Hauser, "Lazy Collision Checking in Asymptotically-Optimal Motion Planning," *IEEE International Conference on Robotics and Automation (ICRA)* , pp. 2951-2957, 2015.

[25] D. Connell and H. M. La, "Dynamic Path Planning and Replanning for Mobile Robots," *IEEE International Conference on Systems, Man, and Cybernetics (SMC),* pp. 1429-1434, 2017.

[26] Y. Kuwata, S. Karaman, J. Teo, E. Frazzoli, J. P. How and G. A. Fiore, "Real-Time Motion Planning With Applications to Autonomous Urban Driving," *IEEE Transactions on Control Systems and Technology,* vol. 17, no. 5, pp. 1105-1118, 2009.

[27] A. Hornung, K. M. Wurm, M. Bennewits, C. Stachniss and W. Burgard, "OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees," in *Autonomous Robots*, 2013.

TRITA EECS-EX-2018:780