Comparing the motion planning methods Hybrid A* and RRT for autonomous off-road driving of bicycle vehicles

Song Luu

Thesis submitted for the degree of
Master in Robotics and Intelligent Systems
30 credits

Department of Informatics
Faculty of Mathematics and Natural Sciences

UNIVERSITY OF OSLO

Spring 2021

Comparing the motion planning methods Hybrid A* and RRT for autonomous off-road driving of bicycle vehicles

Song Luu

Abstract

Motion planning for autonomous vehicles have become a popular topic over the last three decades, with the aim of enhancing safety and efficiency while planning in unfamiliar and demanding environments. When it comes to preparing to drive in rough terrain, nothing is simple. Due to characteristics such as inclination and terrain roughness, separating the environment into obstacles space and free space is insufficient when driving in a challenging environment. In this thesis, the environment is created using a combination of terrain knowledge and an obstacle map. This allows the proposed methods to investigate optimality conditions in terrain directly utilizing terrain characteristics.

This thesis focuses on comparing RRT and Hybrid A* using a simple bicycle model, given the same conditions and environment. The experiments were carried out in two different ecosystems, Rena and Hardangervidda. Especially in a open and flat place like Rena, the Hybrid A* algorithm with higher δ values struggled to find a path, due to the minor elevation difference in the landscape. Here, RRT and Hybrid A* algorithm with $\delta = 0$, both yielded great results with a success rate of 100 % after 10 iterations. In terms of examining optimality conditions, one may argue that RRT has a tendency to find paths that are more safe and likely to be what a person would prefer to drive. Hybrid A* with $\delta = 0$, on the other hand, takes significantly longer to approximate the shortest path. Although the paths obtained in RRT may differ significantly between iterations, RRT and Hybrid A* with $\delta = 0$ might be the best choice in a flat surface such as in Rena.

In Hardangervidda, on the other hand, all algorithms were able to develop a solution path with a success rate of 100 % after 10 iterations. The impact of choosing an incline-based cost is significantly more emphasized due to the large elevation difference in the landscape. In this case, choosing the path with the least amount of incline is more important than finding the shortest path. For this reason, it is reasonable to argue that Hybrid A* with $\delta = 1$ is the optimal planner in Hardangervidda among the presented algorithms in the thesis.

Acknowledgements

First and foremost, I would like to extend my sincere gratitude to my supervisors, Kim Mathiassen and Marius Thoresen, for their helpful guidance, support and enthusiasm throughout the entire thesis. Their knowledge of the field were extremely valuable.

Secondly, I would like to express a special thanks to our closest friends and family who helped me getting through the program, especially this semester.

In addition, a special thanks and appreciation to my fellow students, especially Daniel Woldegiorgis, Eivind Brastad Dammen, Sjamil Gazimagamaev and Øyvind Soma. I could not have completed this program without the support of you guys.

Finally, I want to thank University of Oslo for giving me the opportunity to take part in the Robotics and Intelligent Systems program and I want to offer my heartfelt gratitude to all of the professors for their outstanding work.

Oslo, May 2021

Song Luu

Contents

List of Figures

List of Tables

Abbreviations

AUV	Autonomous Underwater Vehicle
DARPA	Defense Advanced Research Projects Agency
DSM	Digital Surface Model
DTM	Digital Terrain Model
FFI	Forsvarets forskningsinstitutt
GPS	Global Positioning System
LIDAR	Laser Imaging Detection and Ranging
NASA	National Aeronautics and Space Administration
PRM	Probabilistic Roadmap
RRT	Rapidly-exploring Random Tree
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle

Chapter 1

Introduction

This chapter will give a brief overview of the motivation and the problem statement of this thesis. In addition, some related work, as well as state-of-the-art motion planning algorithms for autonomous off-road vehicles, will be presented here to provide an overview of the area. Finally, a description outlining the remainder of the thesis will be given.

1.1 Motivation

In the autonomous self-driving space, the robotic motion planning issue has seen a steady increase in terms of research and implementation over the last three decades. Vehicles are now capable of operating autonomously to some degree, thanks to recent advances in sensing and computing technology. However, research is still ongoing with the aim of improving safety and efficiency by exchanging knowledge and coordinating vehicle behavior on the road.

The Norwegian Defence Research Establishment (in Norwegian: Forsvarets forskningsinstitutt (FFI) has for a number of years been researching Autonomous Underwater Vehicle (AUV) and Unmanned Aerial Vehicle (UAV). In recent years, FFI has begun to look into Unmanned Ground Vehicle (UGV) to research autonomous driving in terrain [25] [38] [26].

UGV is often used in applications where its purpose is to replace humans in dangerous, high-pressure situations. In defence, one may use UGV as combat vehicles, but it also has other applications including firefighting, public safety, surveillance, and supply distribution in remote areas [40]. Outside of military applications, one may see UGVs are being used in space applications, i.e. the National Aeronautics and Space Administration (NASA)'s Mars Exploration Rover project [24]. Moreover, in commercial applications like in agriculture or manufacturing purposes.

Motion planning, also known as path planning, is the method of determining a path through an obstacle-filled area. The motion planning problem is to find a sequence of control inputs which drives the robot from its initial state to one of the goal states given a robot and its dynamics, a representation of the environment, an initial state, and a set of goal states [16]. The vehicle must be able to prepare and arrive at a destination with ease.

However, for off-road driving, the surroundings are complex and it may be ineffective to divide the world into obstacles/non-obstacles. The optimal path may not be the shortest one. Hence, there are multiple variables to considerate, e.g. finding the path with least incline, the least unevenness, the least vegetation, or the one that is considered the safest path.

1.2 Problem statement

The research into motion planning for autonomous off-road driving is still ongoing, with the aim of enhancing safety and efficiency while planning in unfamiliar and demanding environments [14]. The ability to autonomously navigate to a destination in a safe manner is one of the key issues when preparing in challenging terrain conditions. Based on this issue, the aim of the thesis is to investigate two appropriate motion planning algorithms for simulating off-road driving. The following research questions are addressed in this thesis:

1. Are there any motion planning methods that are particularly suitable for this specific task?
2. Investigate adaptation possibilities and utilize this for finding close to realistic and optimal paths.
3. Identify performance metrics for the two motion planning algorithms for evaluation.

1.3 Previous work

In 1959, Edsger W. Dijkstra published a paper about an algorithm for finding the shortest paths between nodes in a weighted graph [5]. Dijkstra's algorithm calculates the path of least cost/weight, also known as the shortest path, by incrementally constructing a collection of nodes with the shortest distance from the source. The collection of nodes are expanded by selecting the edge with least cost, ensuring that the shortest path property is maintained for each new node. The search continues until the goal is reach, and the optimal path is found.

One of the most known motion planning algorithms is the so-called A* search algorithm which was first published in 1968 by researchers at Stanford Research Institute, namely Peter Hart, Nils Nilsson and Bertram Raphael [11]. This algorithm is inspired by Dijkstra's algorithm, but achieves faster performance due to its ability of using heuristics to guide its search.

These searching algorithms inspired many researchers, especially the extension of the well-known A* search algorithm to operate in a known environment. The Defense Advanced Research Projects Agency (DARPA) organized the DARPA Urban Challenge [2], a driverless car competition in 2007. The Stanford Racing Team [4] participated, using a hybrid solution of A*. This A* searching algorithm were expanded with a kinematic state space of the vehicle in $SE(2)$, defined by x, y and θ. In this paper, the state-update rule operated and updated in continuous time, while the search were discretized accordingly. Combining this hybrid-state A* search with Reed-Shepp paths, the vehicle were able to move forward in the state space while it taken in consideration of the obstacle grid's Voronoi graph, yielding great results and took second place in that competition.

In the following years, more trajectory planners were introduced mainly focusing in unknown, unstructured environments with many obstacles. In [6], a unique variant of A* were presented to find a feasible path even with badly placed way-points due to poor Global Positioning System (GPS) signals. This robust solution utilizes Laser Imaging Detection and Ranging (LIDAR) and camera data to calculate cost, while taking in consideration to obstacles, curvature and slope. In similar to [4], this planner also considers the obstacle grid's Voronoi graph. This way, the vehicle will safely avoid obstacles. This autonomous vehicle took first place in the "Autonomous Navigation" scenario at euRathlon in 2013 [7].

Another approach was presented in [15] where an efficient path planning method was designed for real-time operations using short range sensors. By using a A*-like searching algorithm, along with detailed vehicle model and greatly accurate pose estimation, the vehicle could safely preform local path planning. In this research, two real world experiments were performed with exceptional performance.

While A* search methods gained a lot of attraction, an important method in path planning were introduced, i.e. sampling based algorithms. In the early 90s, one of the most popular methods that was gaining a lot of momentum was the Probabilistic Roadmap (PRM) approach, which was developed by researchers at Utrecht University and Stanford University [18]. The idea of the PRM was to generate many samples in the space. Those samples will lie everywhere on the space, for instance on obstacles or in the free space. Afterwards, the roadmap is constructed by trying to connect pairs of samples that are close to each other. After many repetitions, for instance a few thousand samples, one may eventually obtain a dense graph known as a roadmap. In order to connect point A to point B, one should look onto the roadmap and find a nearby node that is in the roadmap from your point A, correspondingly for point B. The last step is to use a standard graph search technique.

In 1998, Steven M. LaValle proposed an efficient method for finding feasible paths more rapidly [20], i.e. Rapidly-exploring Random Tree (RRT). This method is based on randomly sampling a searching area by incrementally building up the searching tree. To obtain optimality, [16] introduced both PRM* and RRT* by implementing a distance-based cost function that will be minimized. In later stages, [21] further developed the RRT* in a off-road terrain with uncertain slip, yielding great results. [37] combined RRT* with LIDAR sensors, where samples were gathered from LIDAR point cloud data, before planning a path while consider roughness to increase safeness.

1.4 Thesis outline

The following sections of this thesis are structured as follows. In Chapter 2, important concepts and technologies in motion planning will be introduced. The methodology used to address the problem statement is covered in Chapter 3. The findings of the simulation are described in Chapter 4, as well as a section dedicated to analysis of the obtained data. Chapter 5 will discuss the validity of the results, challenges encountered during the thesis, along with proposals for future work. Finally, a conclusion is presented in Chapter 6.

Chapter 2

Background and literature

This chapter introduces important concepts and technologies in motion planning such as sampling-based method Rapidly-exploring Random Tree and search-based method Hybrid A*. This will form a foundation that will be used in later chapters.

2.1 Motion planning

Generally speaking, motion planning consists of finding a path or a trajectory for some moving object though an environment. The moving object could be a robot, an autonomous vehicle or perhaps some animated characters in a video game. The primary goal of motion planning is to find a *feasible* path from an initial state to a target state, with finding an *optimal* path as a secondary goal. All paths must be *feasible*, but they do not always have to be *optimal*.

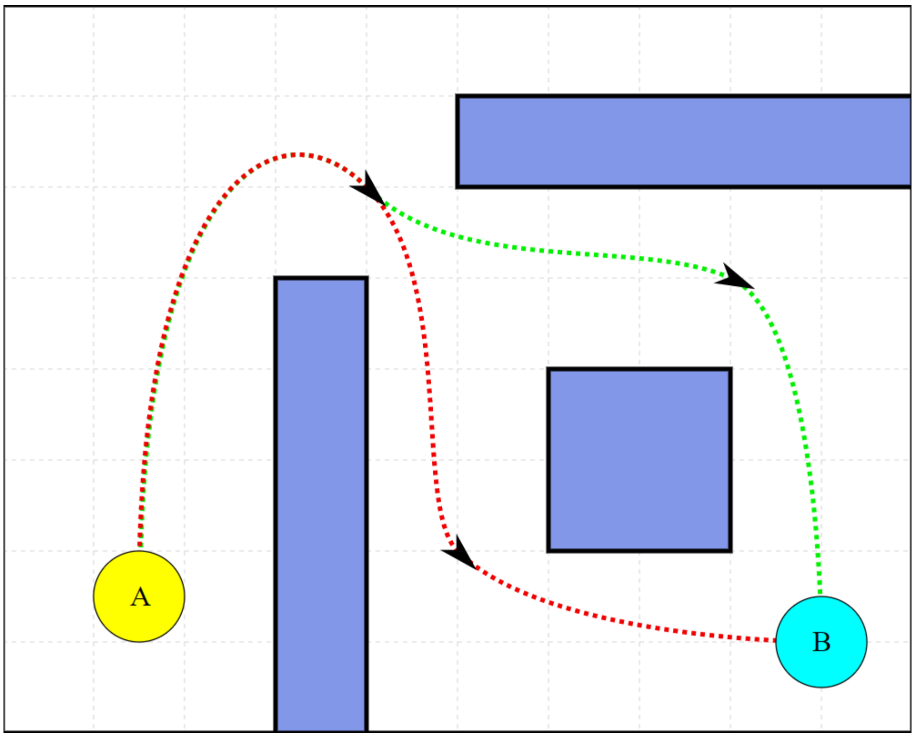


Figure 2.1: An example of a path planning problem showing two equivalent paths to end destination avoiding some obstacles. [30]

In the literature, the terms *feasible* and *optimal* are used to describe the quality of a solution path. A *feasible* path is one that satisfies all the constraints for the moving object. It is critical that the vehicle does not collide with any obstacles in the environment. In addition to this constraint, it is also important to implement constraints in term of physical limitations of the moving object, such as rates of turning and acceleration [28].

An *optimal* path is a path that is based on minimizing a specified cost function. For instance, the most common cost function is based on distance, where the objective is to minimize it to find the *optimal* path in term of distance. However, as mentioned in Chapter 1.1, the *optimal* path may not be the shortest one, but other factors such as incline, unevenness, vegetation or safeness that may be more important. To do so, one must design a customized cost function for the specific purpose of the path planning. An example of a path planning problem is illustrated in Figure 2.1.

2.1.1 Configuration Space

For the sake of planning, it is important to define the configuration space of the robot. A configuration is a set of variables that describes the robot's pose, such as the base position of the robot, its rotation and translation. For instance, if a robot is a single point translating in a two-dimensional plane, then each configuration **q** can be represented by two parameters (x, y) in a plane. In other words, a configuration space, or C-space, is a set of all possible configurations that a robot can assume [32].

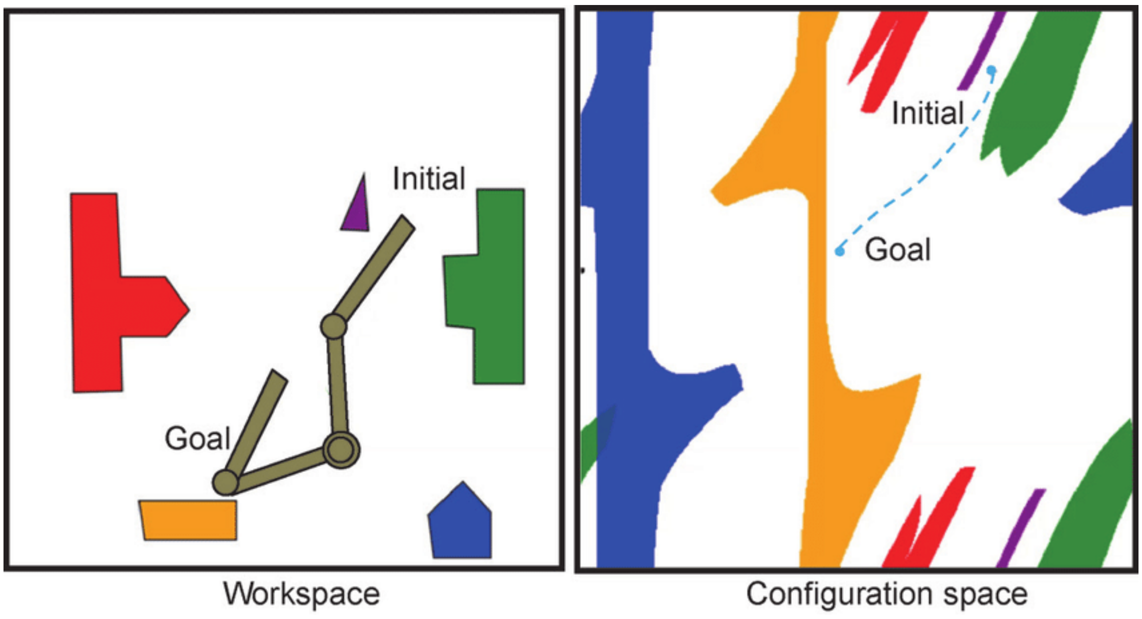


Figure 2.2: "Motion planning in workspaces and in configuration spaces." [27]

In Figure 2.2, the left figure shows a two-linked robot in a workspace with colored obstacles. Given both initial and goal settings in the workspace, it is possible to describe the correspondence between obstacles in the workspace and obstacles in the configuration space, as shown in the right figure. The blue curve is the result of a path planning algorithm, forming a path from start to goal destination [29].

Working in C-space is quite beneficial due to the fact that the moving object is a single point rather than a set of variables.

2.1.2 Environment

In motion planning, a map or a model of the environment is required. It is reasonable to divide the environment into obstacle and non-obstacle segments in order to solve the motion planning problem. In the literature, obstacle space is referred as a restricted area that the robot cannot move because the robot can encounter an obstacle. Non-obstacle space, also known as free space, is a set of configurations that prevent colliding with obstacles. Many algorithms includes a collision detection to test whether the robot's pose collides with the environment, for instance in [15] and [28].

However, everything is not straightforward when preparing to drive in terrain. When driving in a difficult setting, dividing the environment into obstacles space and free space is insufficient due to factors such as incline and terrain roughness. In order to address this problem, a LIDAR derived elevation map may be used in planning. In this case, the entire map of the environment, i.e. a global offline map, can be obtained in advance using LIDAR scans from a flight. A vehicle may also be equipped with sensing technologies such as LIDAR to perform real-time terrain mapping in order to sense the area, i.e. a local online map. LIDAR collects data on terrain features such as obstacle sizes and roughness on a local level and returns a collection of point data, as done in [37]. One may get a good overview of the area by combining terrain knowledge with the obstacle map. Furthermore, there are also many methods for establishing an environment model such as a grid decomposition map, quad split graph and visibility graph [23].

2.1.3 Motion planning with a vehicle model

A vehicle model must be specified for motion planning simulation purposes. Establishing a vehicle model that satisfies its non-holonomic constraints, i.e. constraints that prevent a vehicle from traveling directly to any point in its configuration space, is crucial. In order to go from one state to another, the vehicle must take a sequence of actions.

This section will cover some of the most popular vehicle models that meet these requirements, such as the unicycle and the bicycle model.

Unicycle model

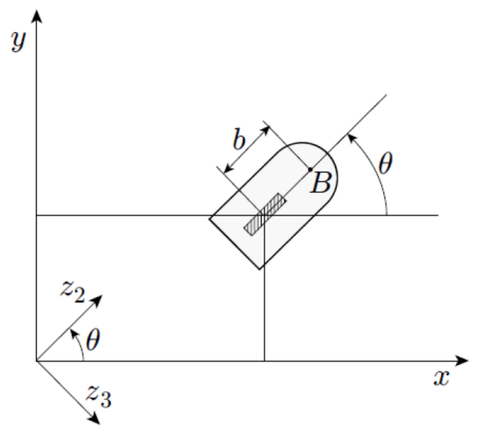
A single-wheeled vehicle is known as a unicycle. This model can be described by configuration $\mathbf{q} = \begin{bmatrix} x & y & \theta \end{bmatrix}^T$, where (x, y) are the Cartesian coordinates of the wheel base, and θ is the orientation of the wheel with respect to the x-axis [32]. This model has a rolling restriction, which implies that all motion in the direction of the wheel plane is determined by wheel spin. This constraint can be written as:

$$\dot{x} \cdot sin(\theta) - \dot{y} \cdot cos(\theta) = 0 \tag{2.1}$$

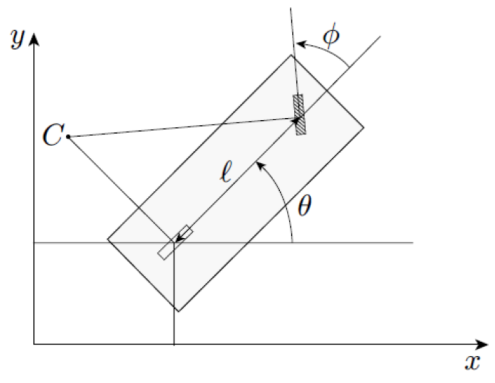
According to equation 2.1, the velocity is zero in the wheel's rolling direction. Figure 2.3a shows the unicycle model. However, the drawback of this model is that it has balance problem. The transition equation satisfies its non-holonomic constraints and is given as:

$$\begin{aligned} \dot{x} &= v \cdot cos(\theta) \\ \dot{y} &= v \cdot sin(\theta) \\ \dot{\theta} &= \omega \end{aligned} \tag{2.2}$$

where v is the driving velocity and ω is the steering velocity.



(a) Unicycle model [32]



(b) Bicycle model [32]

Figure 2.3: Motion planning with a vehicle model

Bicycle model

A bicycle is a two-wheeled vehicle that can be classified according to its configuration $\mathbf{q} = \begin{bmatrix} x & y & \theta & \phi \end{bmatrix}^T$, where (x, y) are the Cartesian coordinates of the wheel base, θ is the orientation of the wheel with respect to the x-axis and ϕ is the steering angle. The bicycle model is illustrated in 2.3b.

Similar to the unicycle model, this model has physical limitations e.g. the vehicle cannot drive sideways as well as the steering radius is bounded [3]. Consider a small time interval ΔT, then the vehicle must drive in the direction that the wheels are pointing. This condition can be written as:

$$\begin{aligned} \dot{x_f} \cdot sin(\theta + \phi) - \dot{y_f} \cdot cos(\theta + \phi) &= 0 \\ \dot{x} \cdot sin(\theta) - \dot{y} \cdot cos(\theta) &= 0 \end{aligned} \tag{2.3}$$

where (x_f, y_f) is the Cartesian coordinates of the center of the front wheel. In order to analyze the kinematics of the model, one must select a reference point (x, y). If the desired point is at the center of the front wheel, the transition equation is given as:

$$\begin{aligned} \dot{x} &= v \cdot cos(\theta + \phi) \\ \dot{y} &= v \cdot sin(\theta + \phi) \\ \dot{\theta} &= v \cdot sin(\phi / L) \\ \dot{\phi} &= \omega \end{aligned} \tag{2.4}$$

If the desired point is at the center of the rear wheel, the transition equation is given as:

$$\begin{aligned} \dot{x} &= v \cdot cos(\theta) \\ \dot{y} &= v \cdot sin(\theta) \\ \dot{\theta} &= v \cdot tan(\phi / L) \\ \dot{\phi} &= \omega \end{aligned} \tag{2.5}$$

where v is the driving velocity and ω is the steering velocity [32].

2.2 Sampling based algorithm

Sampling based algorithms have become a popular choice due to its effectiveness and robustness in high-dimensional continuous spaces. These algorithms explores the environment by randomly selecting way-points from the state space and examines whether the point is reachable for the vehicle using steering and collision checking routines [28]. If successful, the algorithm will return a valid path.

2.2.1 Rapidly-exploring random tree

As briefly mentioned in Chapter 1.3, LaValle proposed an efficient method for finding feasible paths more rapidly [20]. This method is based on randomly sampling a searching area by incrementally building up the searching tree.

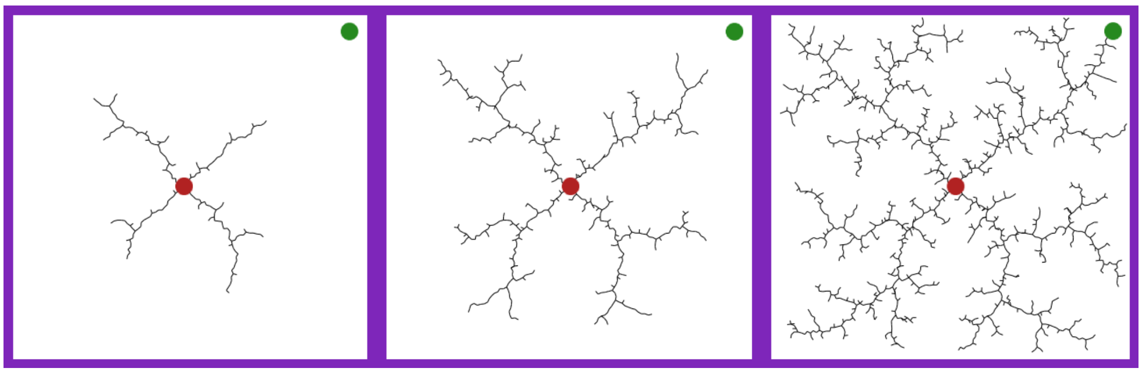


Figure 2.4: Example of RRT algorithm [36]

The RRT algorithm is very simple to understand and implement. Based on the motion planning problem mentioned in Chapter 1.1, i.e. to find a sequence of control inputs which drives the robot from its initial state to one of the goal states given a robot and its dynamics, a representation of the environment, an initial state, and a set of goal states. The vehicle must be able to smoothly plan and reach an end destination.

The general idea is to create a tree that starts at the beginning and expands until it reaches the desired region. A tree is a graph in which each node has only one parent. First, the only node in the tree is the start node. Afterwards, the algorithm generates a random sample in the map. If that sample is located in the free space and if the direct line to the closest node in the tree does not cross any obstacles, a step will be taken from the tree to the sample using a predefined step size ϵ. Lastly, the initial node will be connected to the new node, incrementally building up the searching tree to hopefully find the path.

If this operation fails, the algorithm will select the node in the tree that is nearest to it. This continues until the tree reaches the desired destination. The algorithm terminates when the target is connected to the tree.

Instead of choosing a random sample in the map, LaValle recommended a bias function. This bias improves exploration performance by selecting the goal node rather than a random sample. As a result, steering the search towards the final goal.

Algorithm 1: RRT algorithm [20]

```
Function Build_RRT($x\_init$):
    $graph$.init($x\_init$)
    for $k = 1$ to $K$ do
        $x\_init \leftarrow random\_state()$
        EXTEND($graph$, $x\_init$)
    end
    return $graph$

Function EXTEND($graph$, $x$):
    $x\_near \leftarrow NEAREST\_NEIGHBOR$(x,graph)
    if $NEW\_STATE(x, x\_near, x\_new, u\_new)$ then
        $graph$.add_node($x\_new$)
        $graph$.add_edge($x\_near$, $x\_new$, $u\_new$)
        if $x\_new = x$ then
            return goal reached
        end
        return continue
    end
    return failed to find path
```

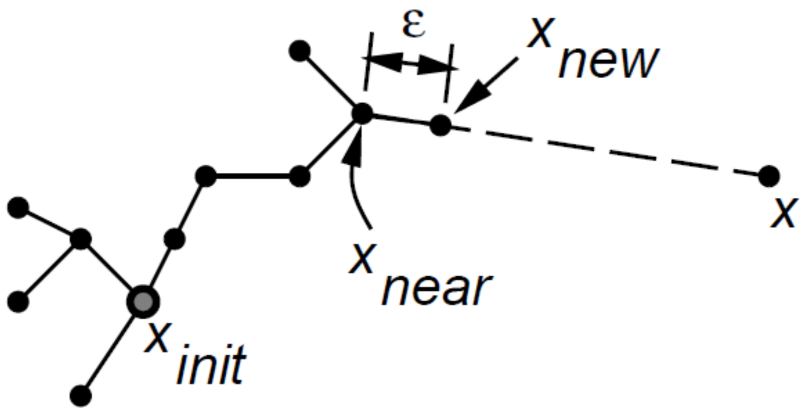


Figure 2.5: Extend operation in RRT [20]

An approach to extend RRT for a non-holomomic vehicle is to use motion primitives, i.e. a set of computed motions that the robot can take, to make the path admissible. The procedure is similar to Algorithm 1, where the main difference is that for each generated sample x, the motion primitives are applied starting from x_near. One could either select one of the configurations randomly or select the one that is closest to the random generated sample x. A collision test must be preformed before adding the path to the tree. Figure 2.6 shows an example of this approach.

The benefit of RRT is the efficiency with regard to computational effort. This algorithm finds a feasible path in high dimensional continuous spaces rapidly compared to searching algorithms like A*. However, one of the main drawbacks of RRT is that the path obtained is only feasible.

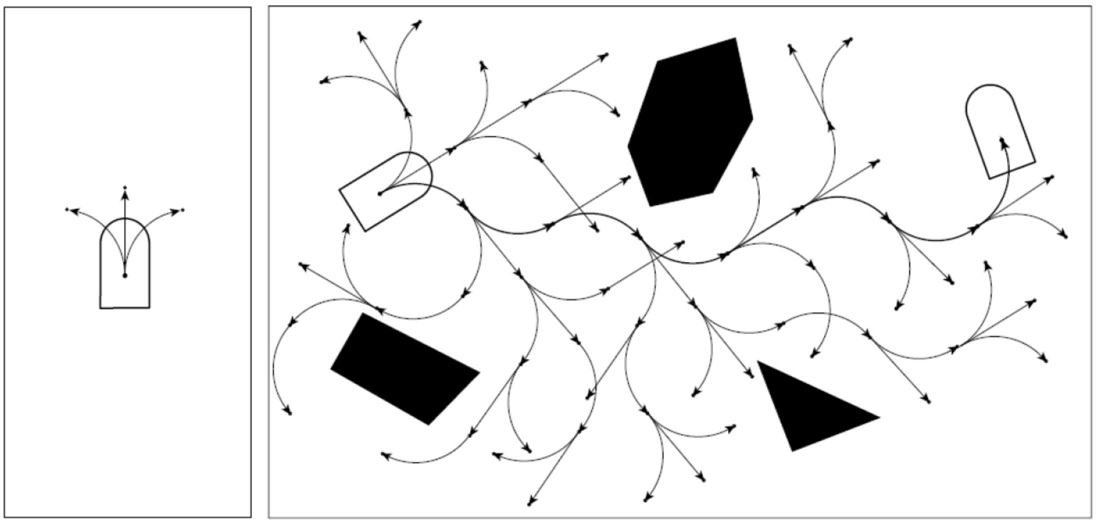


Figure 2.6: An example of motion planning for a unicycle, using a set of motion primitives [32]

2.2.2 Bidirectional Rapidly-exploring random tree

LaValle suggested the concept of having two growing trees instead of one to increase efficiency [20]. One tree begins at the initial state, while the other begins at the end state. A solution path is identified whenever the two RRTs intersect. To ensure that the trees meet before covering the entire searching area, the RRT construction must be biased. In this field, further research has been conducted, such as in [22] and [31].

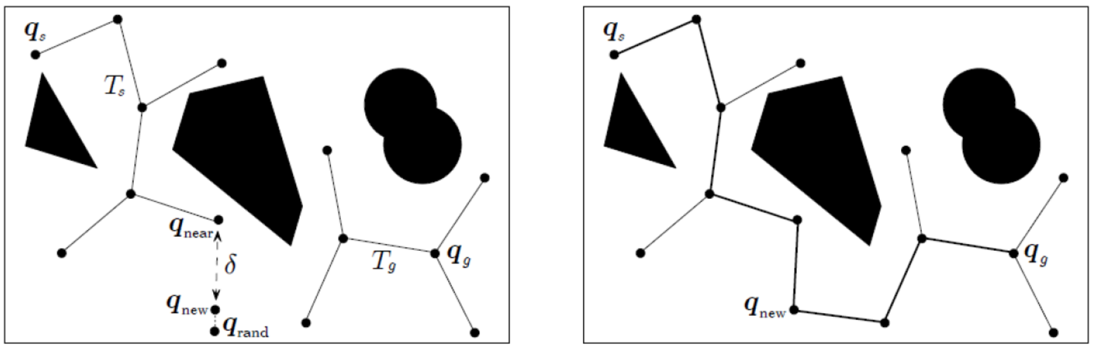


Figure 2.7: Bidirectional RRT [32]

2.2.3 Rapidly-exploring random tree*

One of RRT's drawbacks is that the solution is not optimal like the one given by the A* 2.3.1, but this can be dealt by using path optimization techniques that can reduce greatly the length of the RRT path. [16] tackled this problem by introducing RRT*, an algorithm that focused on path optimality. The purpose is to look at multiple nodes and choose the connection which preserve the structure, and which minimizes the path length. Furthermore, one wants to find the cost between the sampled node and the parent node such that it is minimized.

Once a path is found from an initial trial, the method performs sampling again and evaluates if the new path is more optimal than the previous path. The sampled paths will asymptotically converge to the optimal path.

2.3 Search-based algorithm

Search-based algorithms, also referred as grid-based algorithms in the literature, are based on overlaying a grid on the configuration space. When overlaying a grid on the configuration space, each configuration is identified with a grid point. For each grid point, the robot is allowed to move to nearby points given there are no obstacles in the path.

2.3.1 A* algorithm

One of the most known search-based algorithms is the A* search algorithm [11]. This informed search algorithm achieves great performance by utilizing heuristics to guide its search. With this approach, the goal is to find the shortest path from point A to B in a graph using the edge connections. At first, the starting node is added to the ***open set***. Essentially, the ***open set*** is a queue that keeps track of the nodes that one wants to consider. The algorithm is prioritizing nodes with respect of a total cost for distance, i.e.

$$F(n) = H(n) + G(n) \tag{2.6}$$

H score is a function $h(n)$ that gives us an estimate of the absolute distance from current node n to the end node, without considering any obstacles that blocks the path. G score, however, is the current shortest distance from the start node to the current node, while the F score is the total score with H score and G score combined. The F score, or the total score, represents how far the current node n is to the goal node n while taken in consideration of the shortest distance from the start node to current node n. In other words, the total score represents how the finalized path would look like.

As mentioned earlier, the algorithm considers nodes that has the lowest total score, because it attempts to find the shortest path. While considering nodes, the algorithm is computing the total score to its neighbor nodes n. When finished, the node that has been examined is moved to a set called the ***closed set***. By doing so, it prevents checking a node more than once. The neighbor nodes that are examined, are then moved to the open set which will be ordered in an ascending matter. The algorithm terminates once a path is found or there are no paths eligible to be extended, i.e. failing to find a valid path.

Algorithm 2: A* algorithm pseudocode [19]

```
OPEN //nodes to be evaluated
CLOSED //nodes that has been evaluated
add the start node to OPEN
while path_not_found do
    current = node in OPEN with the lowest f_cost
    remove current from OPEN
    add current to CLOSED
    if current_node = end_node then
        return path
    end
    for all neighbors of the current node do
        if neighbor IS NOT reachable OR neighbor IS IN CLOSED then
            skip to the next neighbor
        end
        if new path to neighbor IS shorter OR neighbor IS NOT IN OPEN then
            set f_cost of neighbor
            set parent of neighbor to current
            if neighbor IS NOT IN OPEN then
                add neighbor to OPEN
            end
        end
    end
end
```

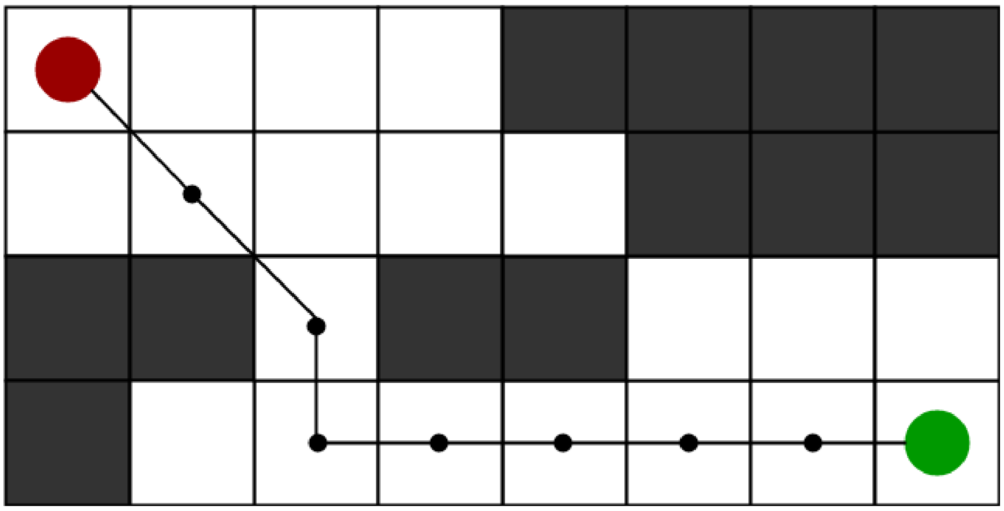


Figure 2.8: An example of A* algorithm in a grid search environment [8]

A* produces an optimal path when successful. However, this method is computational expansive and consumes a lot of memory space. This is because every cell in the grid needs to be stored in the memory, even if it is not contribute to finding the solution. The computations required are increasing exponentially as the dimensionality of the problem or the resolution of the grid increases [12].

When designing an A* algorithm, there are multiple ways to measure distance such as:

1. **Manhattan distance**, which allows 4 directions of movement.
2. **Diagonal distance**, which allows 8 directions of movement.
3. **Euclidean distance**, which allows any direction of movement.

Certainly, the choice of measuring distance has a large impact on what how many neighbor nodes that are going to be considered. As a result, not only could this impact the finalized path, but also the running time and performance.

2.3.2 Hybrid A* algorithm

In the last decade, there has been done further research in the field of Hybrid A* due to its outstanding performance in motion planning applications, especially in an unknown territory with uncertainties. Normally, in a Hybrid A* algorithm, the A* algorithm is expanded with a vehicle model bearing in mind of the constraints in term of physical limitations of the moving object, such as rates of turning and acceleration. A tree is growing out of the nodes to replicate the steering angles for a given vehicle. As a consequence of that, the path produced in Hybrid A* is smoother compared to A*. Figure 2.9 emphasizes these modifications.

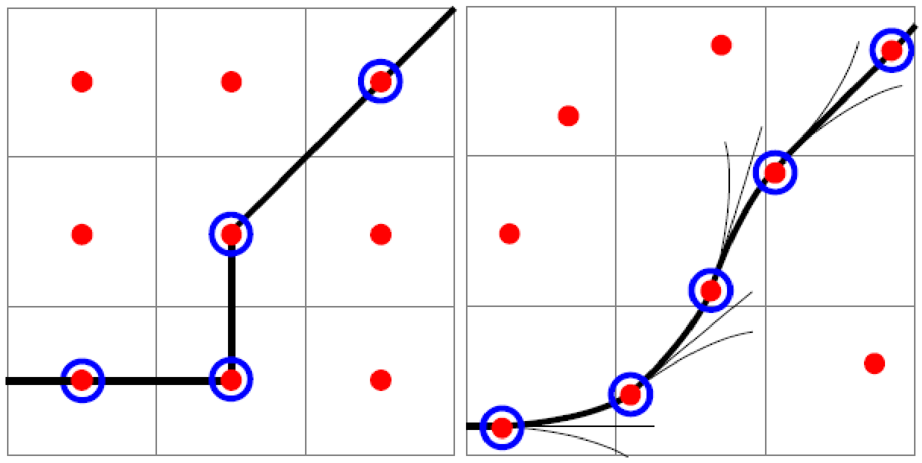


Figure 2.9: The main difference between A* and Hybrid A* [4]

Similar to A*, Hybrid A* algorithm intends to find the shortest path. Due to motion primitives and tree expansion, the path obtained in Hybrid A* is only an approximation of the shortest path, as illustrated in Figure 2.9. Furthermore, as mentioned in Chapter 1.3, researchers have also discovered ways to modify the cost function so that it takes into account steepness or curvature, as done in [6]. This can be archived by adding a cost term in equation 2.6 that utilizes terrain characteristics to calculate an additional cost, e.g.

$$F(n) = H(n) + G(n) + C(n) \tag{2.7}$$

Chapter 3

Methodology

This chapter describes the methodology adopted in order to answer the thesis' problem statement presented in Chapter 1.2:

1. Are there any motion planning methods that are particularly suitable for this specific task?

2. Investigate adaptation possibilities and utilize this for finding close to realistic and optimal paths.

3. Identify performance metrics for the two motion planning algorithms for evaluation.

3.1 Objectives

This thesis focuses on simulating a motion planning problem, i.e. planning a path in a challenging terrain with many uncertainties. In order to plan the optimal path, one may often try various methods with different environments which can be used for comparison in the later stages.

According to recent studies, there are two types of methods that are most commonly studied: sampling-based methods and searching-based methods. The methodology chosen in this thesis is to compare a sampling-based method with a searching-based one. RRT, a sampling-based method, has been chosen for investigation due to its ability to find a feasible path efficiency with regard to computational effort. Hybrid A*, on the other hand, has been chosen for further research as the searching-based method due to its extensive possibilities for developing a cost function that takes into account steepness, curvature and distance, as mentioned in Chapter 2.3.2. It would be particularly interesting to look at the differences between the two motion planning algorithms, as well as how well they would perform in a challenging terrain with numerous obstacles.

This thesis focuses on comparing RRT and Hybrid A* using a simple bicycle model, given the same conditions and environment. Furthermore, it is essential to utilize the terrain characteristics directly to investigate optimality conditions in terrain. Deciding upon what is the optimal path is something that must be examined in the thesis.

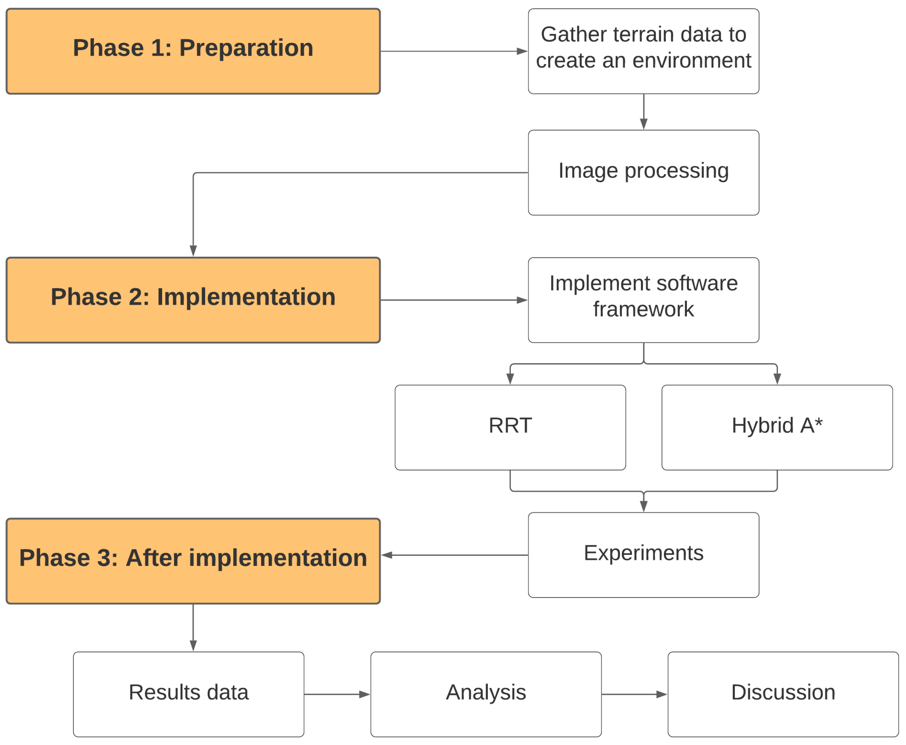


Figure 3.1: The thesis' methodology

The thesis' objectives can be illustrated as shown in Figure 3.1 and can be divided into three phases detailed as followed:

1. **Preparation**

   First and foremost, a model of the environment is required for motion planning applications. Since this thesis focuses on simulated driving, it is desirable to obtain and use real and high-resolution terrain data. As a result, a global offline map may be the best option. As stated in Chapter 2.1.2, one might combine terrain information with an obstacle map to get a good overview of the area. As a consequence, image processing is needed to manipulate the terrain data to meet the task's requirements.

2. **Implementation**

   The aim of this phase is to answer to the second thesis problem statement, as mentioned previously. The goal is to create an RRT and Hybrid A* software architecture based on a bicycle model. Following that, the implemented algorithms will be tested in the environments that were acquired during the preparation phase.

3. **After implementation**

   The final phase is dedicated to addressing the third thesis problem statement. Multiple simulations in various environments will be conducted with the aim of investigate optimality conditions based on different terrain characteristics. Later on, the gathered data will be analyzed and discussed.

3.2 Preparation phase

Initially, a map or a model of the environment is required in preparation for software implementation. This is important due to the fact that the methods will directly utilize the terrain characteristics to plan a path given some start and end coordinates. Since this research study is focusing on simulating motion planning for autonomous off-road vehicles, it is satisfactory to download terrain data, i.e. a global offline map, rather than using outdoor map tools such as LIDAR for real experiments. As a result, terrain data has been downloaded from hoydedata.no [13], a website that provides high-resolution terrain map data over Norway. Figure 3.2 illustrates the website which will be immensely used during the preparation phase.

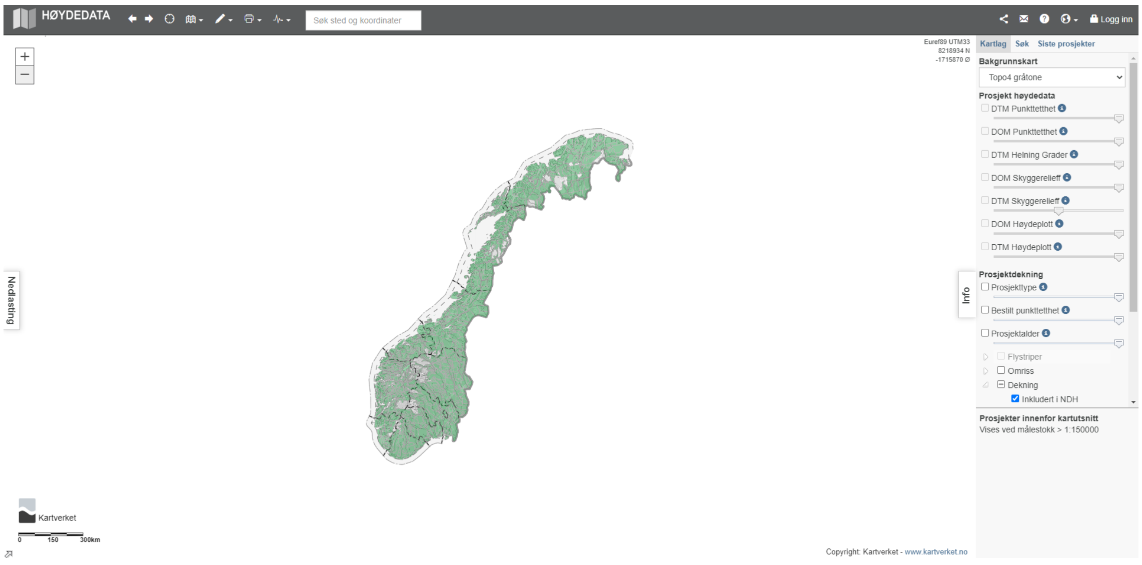


Figure 3.2: High-resolution terrain data from hoydedata.no

3.2.1 Data collection

When collecting data from hoydedata, the first step is to find a desirable field of their choice. The received data is in GeoTIFF, an image format file for high-quality graphics which includes georeferencing information. This format is inconvenient when using computer vision libraries like OpenCV, because tif-files are not supported. Fortunately, there are various methods to visualize satellite images, for instance RasterIO, Georaster and GDAL [17]. GDAL is the most popular library amongst them, making it a natural choice for converting datatypes [10] [41]. By using GDAL, one may successfully convert tif-files to matrices, which makes image processing achievable.

3.2.2 Models

When retrieving the data from hoydedata.no, one will get a model containing information about the surface and the terrain of a certain area. After converting tif-files to matrices, one can exploit the models to create an environment. Scaling has been performed due to the size of the obtained data. The original data was in 0.25 point density, but it was scaled to 1.05 for storage and display purposes.

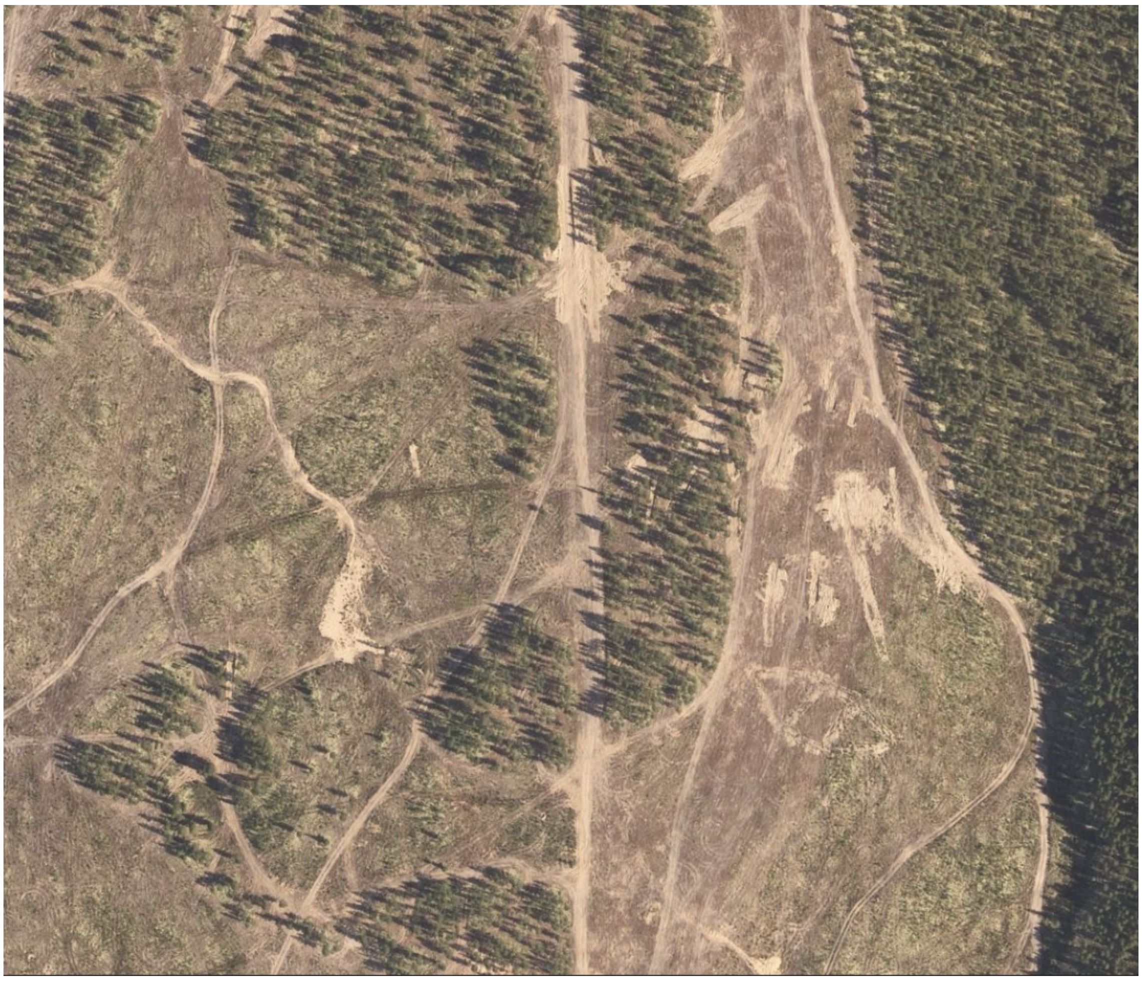


Figure 3.3: High-resolution terrain data of Rena, Norway obtained from hoydedata.no

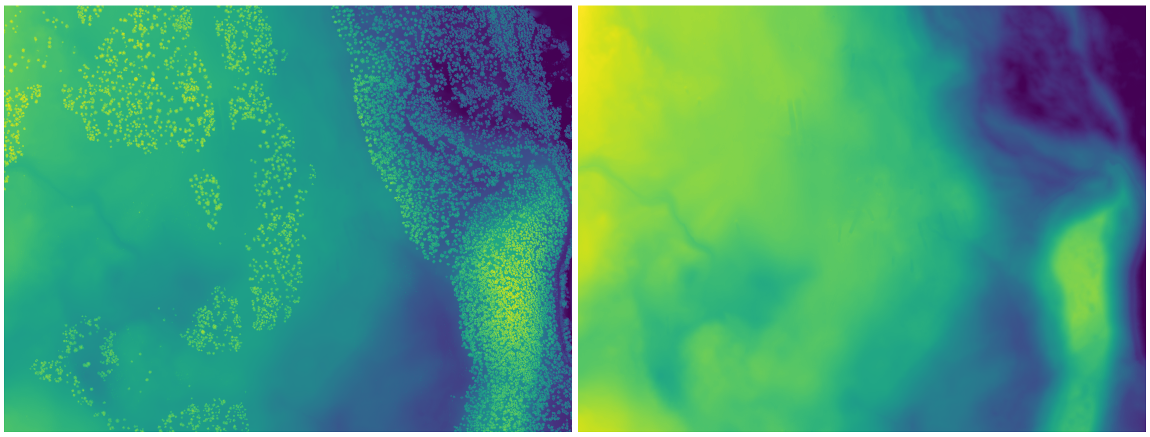


Figure 3.4: The left image illustrates the Digital Surface Model and the right image illustrates the Digital Terrain Model of Rena, Norway.

Digital Surface Model

A Digital Surface Model (DSM) is a three dimensional representation of a surface that captures the elevation of all objects, such as trees, buildings and vegetation seen from above [33]. This information is crucial in the process of creating an obstacle map. Figure 3.4 illustrates a DSM obtained from hoydedata.no.

Digital Terrain Model

Similar to DSM, a Digital Terrain Model (DTM) is also a three dimensional representation of a surface. The main difference in DTM compared to DSM is that it does not include underlying objects such as trees. It includes height differences in a terrain. An example of a DTM can be seen in Figure 3.4.

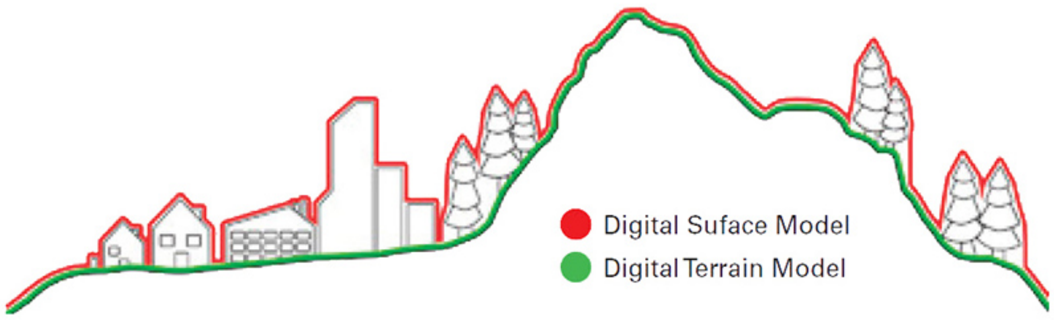


Figure 3.5: An illustration showing the main differences between DSM vs DTM [9]

3.2.3 Image processing

Now that both DSM and DTM are acquired, it is time to do some image processing to create a obstacle map. An easy approach is to compute the difference between DSM and DTM to get a map consisting of obstacles. With this relatively easy approach, one may see some adequate results like shown in Figure 3.6. The code can be inspected in Listing B.1.

To sort the image during the development of the obstacle map, a threshold is set. The noise is removed using a 3 meters threshold. Everything over 3 meters in Rena, such as trees, would be considered as obstacles.

Sobel operator

Map analysis is essential when creating a motion planning application. A notable tool is the Sobel operator [34]. One can consider using a Sobel operator to obtain even more information from the gathered terrain data. Sobel operation is a edge detector which is used to find the gradient of an image. If a Sobel operator is applied to the Digital Terrain Model, i.e. the plain terrain model without objects, the result will be a map containing valuable information about the gradient, hence a pure steepness map. Figure 3.7 illustrates the steepness of the same area as in Figure 3.3.

Because the objective of this thesis is to simulate in many environments, a similar procedure is used to create a new model. This segment will take place in Hardangervidda, Norway, a mountainous region with numerous gradient variations. Instead of utilizing a threshold for the $DTM - DSM$ picture, a threshold is utilized in the Sobel picture for creating an obstacle map of Hardangervidda. When the gradient exceeds 3 meters, it is classified as an obstacle. The code can be inspected in Listing B.2.

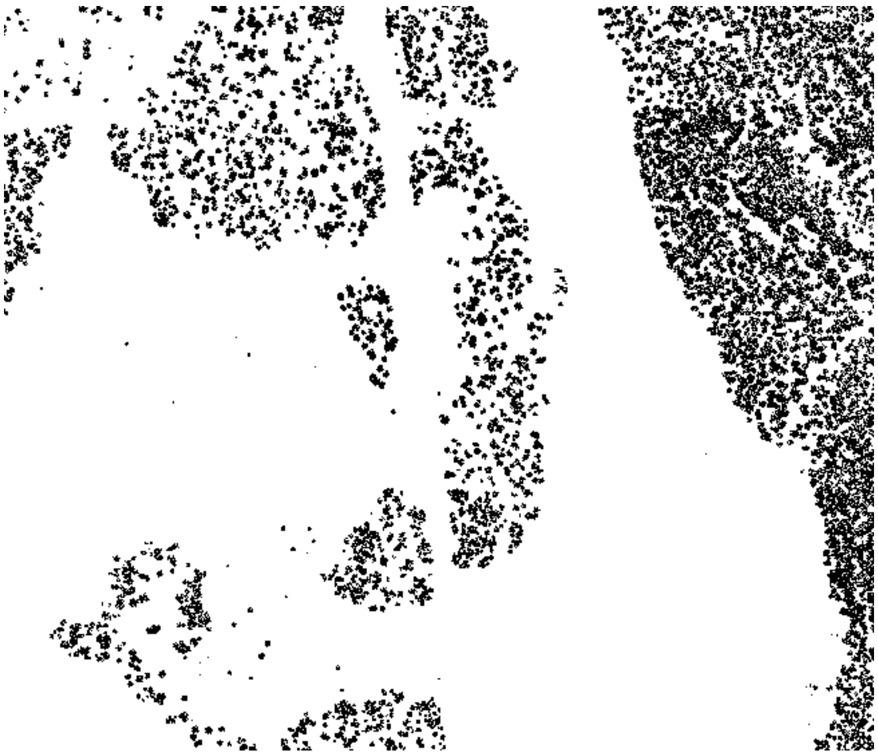


Figure 3.6: The difference between DSM and DTM from the same image as in Figure 3.4. In addition, a threshold of 3 meters is used to remove noise.

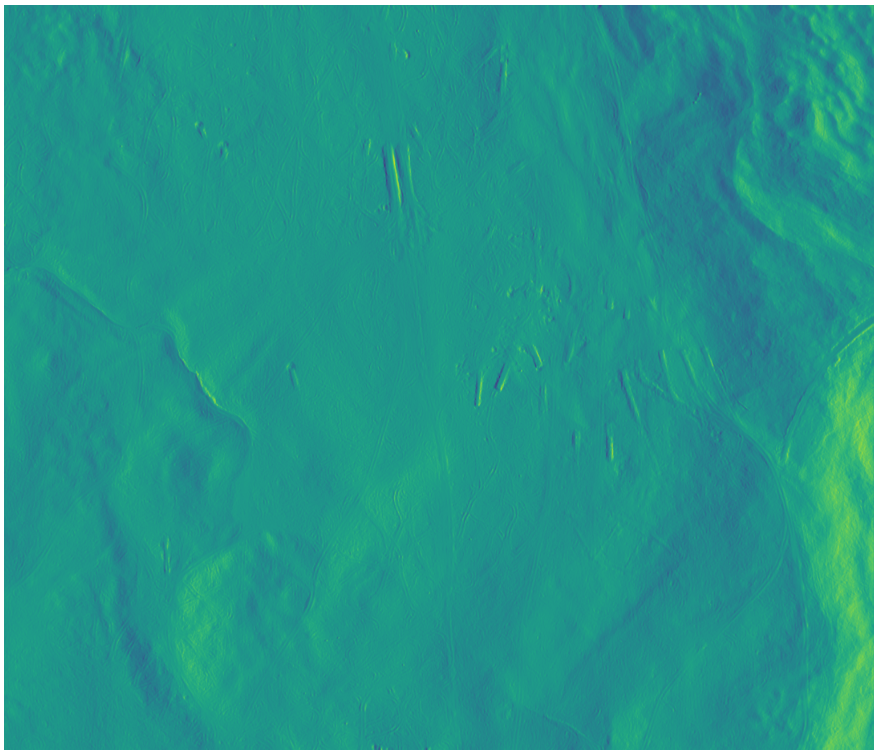


Figure 3.7: Pure steepness map obtained using Sobel operator. The Sobel picture is from Rena.

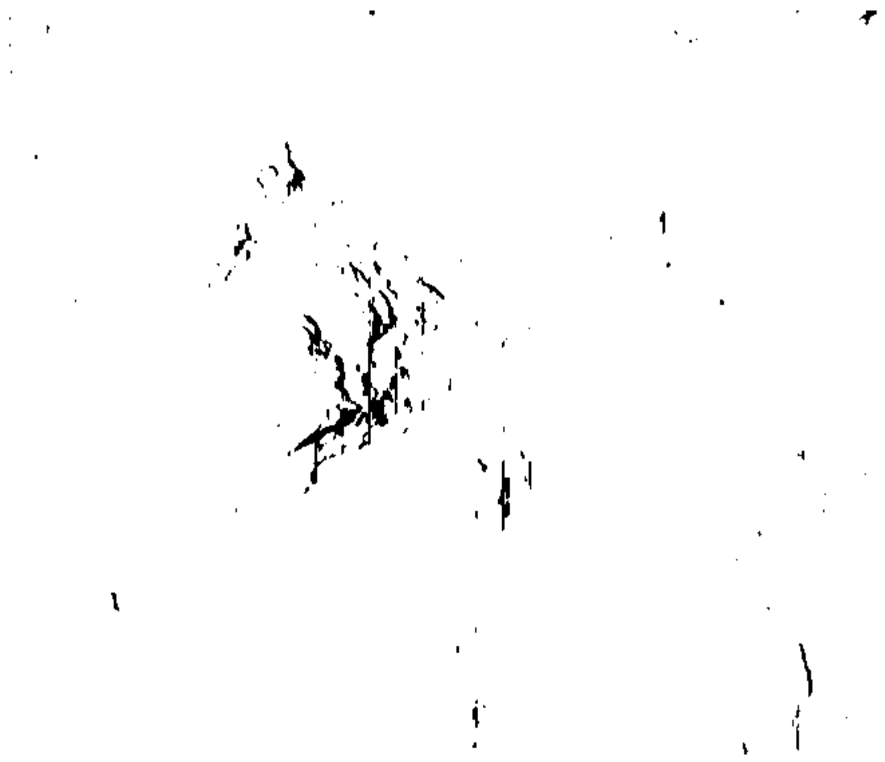


Figure 3.8: Obstacle map of Hardangervidda. When the gradient is greater than 3 meters, it is considered an obstacle.

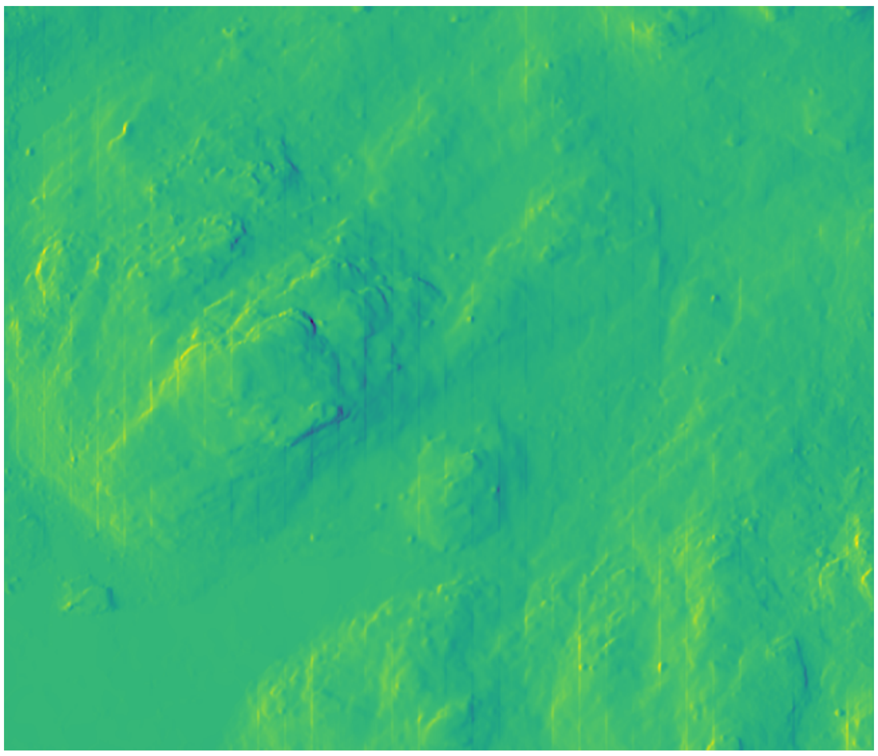


Figure 3.9: The Sobel image of Hardangervidda

3.3 Implementation phase

The implementation phase aims directly at the problem statement presented in Chapter 1.2 and Chapter 3. Since the preparation phase is finished, and the maps are prepared, it is time to start with the implementation. In this phase, the software framework of RRT (presented in 2.2.1) and Hybrid A* (presented in 2.3.2) will be implemented.

The software will be implemented in Python 3 due to its popularity amongst programmers and its huge library versatility in image processing, automation and visualization. Many modules in Python are being used, including pygame for visualization, OpenCV for image processing and others that will be explained in further sections.

Obviously, there are other programming languages that are suitable for this task. C++ and ROS has been considered in the earlier stages of this thesis, but due to personal preference and coding capability, it was natural to choose Python for this task.

3.3.1 Designing RRT

In this section, RRT will be implemented based on Chapter 2.2.1 and explained in detail. The code can be viewed in GitHub repository [35] and is based on [1]. The following flowchart presented in Figure 3.10 shows the algorithm in a step-by-step manner.

First, based on the motion planning problem mentioned in Chapter 1.1, it is required to define an initial state, goal state and get a representation of the environment in order for the vehicle to plan a path from A to B. These parameters will be set in the initialization section of Figure 3.10, along with the obstacle map obtained from Chapter 3.2.3. This map is essential in order to do collision checks for the given vehicle. Second, the tree that is built by nodes, has to be stored in arrays to keep track of its parent.

The general idea is to create a tree that starts at the beginning and expands until it reaches the desired region. First, the only node in the tree is the start node. The algorithm then moves on to sampling a new point on the map at random. By comparing distances between all nodes in the tree, the nearest node in the tree will be identified. If the sample is in free space and the direct line to the nearest node in the tree does not cross any obstacles, a predefined step size will be taken from the tree to the sample. Finally, the initial node will be connected to the new node, gradually increasing the size of the searching tree in the hopes of discovering the path.

The collision check routine is done mainly by using a look-up table of the obstacle map, with the assumption that the vehicle is a dot in the environment. This routine is divided into two parts. Firstly, whenever a new random point is sampled, a check is done whether or not that point is in a obstacle. The new node (x_n,y_n) is sent to the obstacle map, checking whether the pixel value is 255. If this is the case, the pixel must have a white background. If the pixel value is 0, i.e. black filling, the new node will be removed, and a new point will be sampled. Secondly, the function ***crossObstacle***(x_1,x_2,y_1,y_2) is checking whether there are obstacles in between those two coordinates, making it untraversable. This is done by using a linear interpolation approach with 100 samples in between two coordinates. If those conditions are satisfied, the edge will connect the two nodes, further expanding the tree. The code can be inspected in Listing C.2.

If the tree doesn't reach the goal within 10 seconds, the algorithm will terminate. However, if the tree finds the goal position, it will go through the stored nodes and append its parents starting from the goal node. This way, the algorithm can track where the nodes came from and backtracks to the initial position. The code can be inspected in Listing C.3.

For every 20 iterations, a bias function is called. This function is designed to drive the search towards the goal by selecting the goal position as the reference direction rather than a

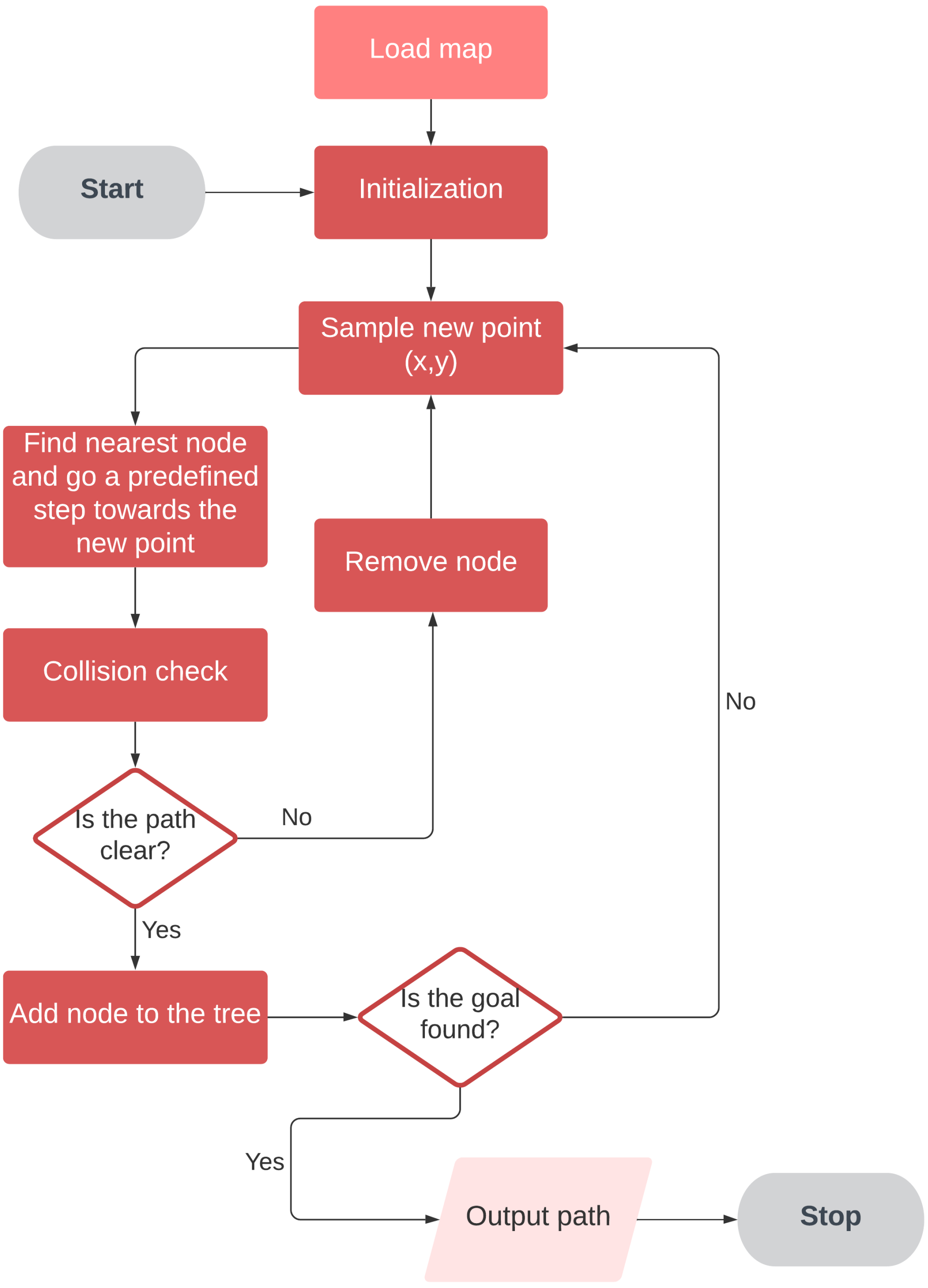


Figure 3.10: RRT flowchart

random sample, as mentioned in Chapter 2.2.1. The code for this function can be inspected in Listing C.4. Every other iterations will use the expand function until a path is found.

3.3.2 Designing Hybrid A*

In this section, Hybrid A* will be implemented based on the literature provided in Chapter 2.3.2. The code can be viewed in GitHub repository [35] and the reference code for Hybrid A* is based on [39]. The algorithm is described in the flowchart below in Figure 3.11.

Since the objective is to compare RRT and Hybrid A*, the same environment is loaded to form an identical base as in RRT. In this way, it will be easier for compare the algorithms in the analysis part of the thesis. The provided Hybrid A* algorithm is an extension of A* and where a simple bicycle model is implemented for the vehicle to find its neighbor nodes for any given node that the vehicle is in. With this approach, kinematic constrains will be preserved. Motion primitives, i.e. computed motions that the robot can take, is used to calculate costs.

As previously stated in Chapter 2.3.1, the *open set* that keeps track of the nodes in a queue, prioritized by the lowest total score. To keep track of the priority queue *open set*, the python module **heapq** is used. This module is advantageous because it automatically sorts the incoming path and the corresponding costs in an ascending order. At first, the starting node is added to the *open set*, and it is the first node that will be explored to kick off the searching process. Then, the bicycle model described in Chapter 2.1.3 will be used for finding its neighbor nodes from the starting node. A similar collision check as in RRT will be preformed, in order to check whether the path is suitable for driving. Afterwards, each motion will be assigned a total cost, and the lowest score among them will be selected for further exploration until a solution is found.

The cost function that will be used for the proposed method is,

$$F(n) = H(n) + ((1-\delta) \cdot G(n) + \delta \cdot C(n\_prev, n)) + C_s + C_v \tag{3.1}$$

where

1. $F(n)$ is the total cost.
2. $H(n)$ is the heuristics function that gives us an estimate of the absolute Euclidean distance from current node to the end node.
3. $G(n)$ is the distance travelled from start to current node.
4. $C(n\_prev, n)$ is the cost function. This function directly uses the Sobel image, summing the absolute gradients, i.e. change in elevations for 20 samples between the motion primitives. Interpolation of the curved path is used to determine the waypoints of 20 samples between nodes.
5. C_s is the cost for steering. [1]
6. C_v is the cost for velocity. [2]
7. δ is the weighting scalar $\in [0, 1]$.

By assigning a low δ value, for instance 0, the algorithm will behave like a pure A* algorithm where the objective is to find the shortest path. Furthermore, as δ increases towards 1, the algorithm will focus more on the terrain slope when planning a path. This way, the algorithm is expected to find a less rough path focusing more on the terrain cost rather than pure distance. The flowchart describing the algorithm can be seen in Figure 3.11.

[1]Must be tuned prior to simulation.
[2]Must be tuned prior to simulation.

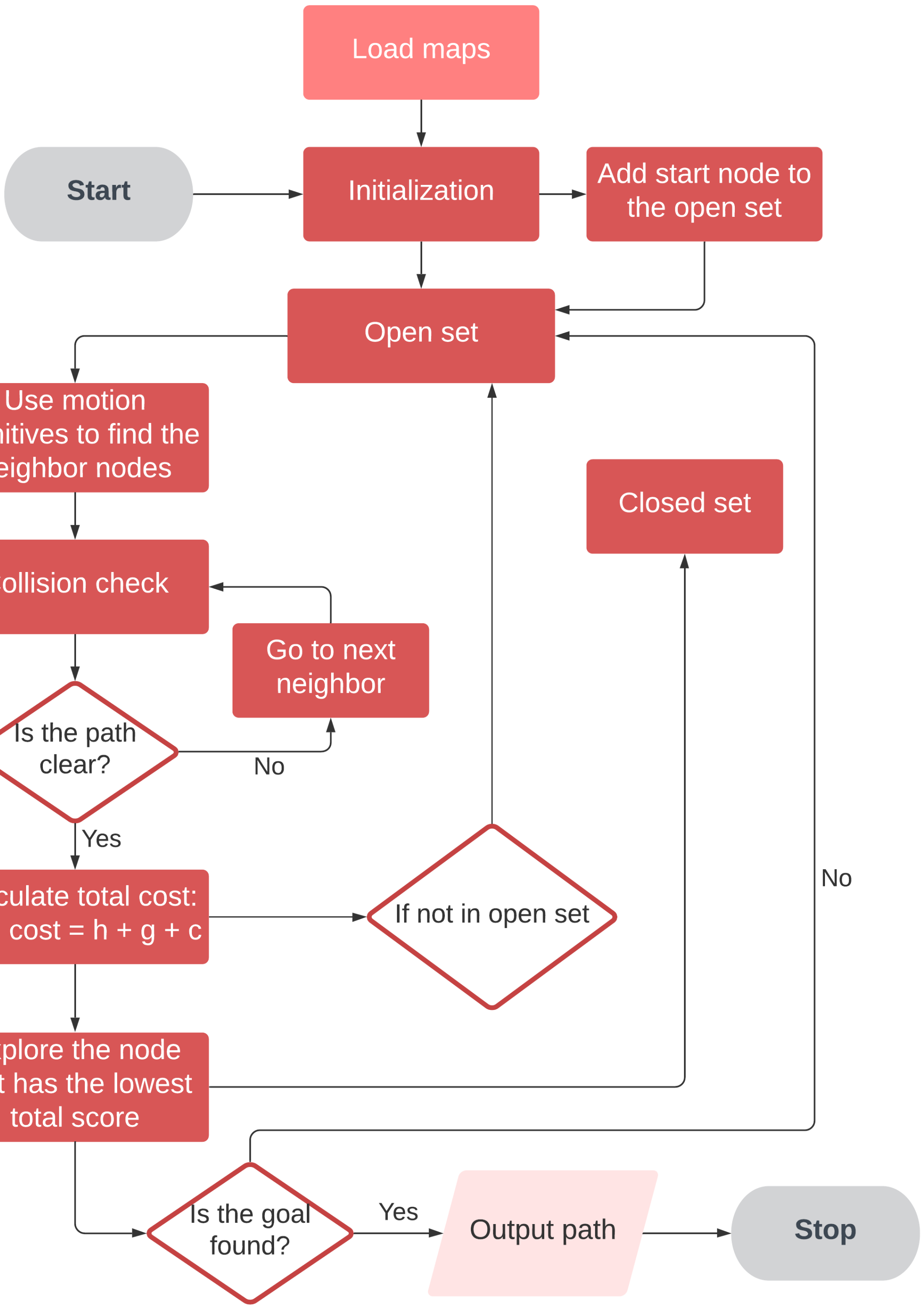

Load maps
Start
Initialization
Add start node to the open set
Open set
Use motion primitives to find the neighbor nodes
Closed set
Collision check
Go to next neighbor
Is the path clear?
No
Yes
Calculate total cost: total cost = h + g + c
If not in open set
No
Explore the node that has the lowest total score
Is the goal found?
Yes
Output path
Stop

Figure 3.11: Hybrid A* flowchart

3.4 Expected results

As mentioned in Chapter 2.2.1, the main benefit of RRT is its efficiency in terms of computational time. As opposed to searching algorithms like Hybrid A*, RRT finds a feasible path in high-dimensional continuous spaces faster. Hybrid A* with $\delta = 0$, on the other hand, produces an optimal path in term of minimum distance when successful. However, this method is computational expansive and consumes a lot of memory space because all cell information needs to be stored in the grid. Moreover, it will be interesting to look at the results obtained in Hybrid A* algorithm with different δ values. It is expected that the algorithm will find a alternative path by partially or only taking in consideration of incline, depending on the δ value. This will make the vehicle drive longer compared to minimum distance path planner in Hybrid A* with $\delta = 0$.

Chapter 4

Results

This chapter describes the result of the methodology outlined in Chapter 3. To evaluate the efficiency of the proposed methods, the algorithms will be evaluated in two completely different settings. Prior to simulation, the proposed cost function's parameters will be tuned.

4.1 Parameter tuning

The first experiments will be in conducted Rena, Norway, with the focus on tuning the cost function prior to simulation. This place is suitable for simulation because this place is open and the surface is quite flat. In the past few years, FFI has also conducted real experiments here.

In the first experiment, the focus is on tuning the cost function of Hybrid A* to investigate optimality conditions based on terrain characteristics. Hybrid A* with $\delta = 0.25$ will be checked for steering and velocity without incurring any extra costs. This means that the planner can plan a path without taking into account driving in reverse and can steer as it pleases.

Experiment 1 - Parameter tuning	
Algorithm	Hybrid A* with $\delta = 0.25$
Start coordinates	(458, 164)
End coordinates	(272, 331)
Starting pose	110° deg
Steering	$[-30°, 0°, 30°]$
Cost for steering	[0, 0, 0]
Velocities	[-5, 20]
Cost for velocities	[0, 0]

Table 4.1: Setup for experiment 1

As shown in Figure 4.1, the starting coordinates (458, 164) are indicating with a green circle in the upper right corner of the image, while the goal coordinates (272, 331) are the larger circle in the left side of the image. The dense section of the found path, marked by a large number of red circles, takes up the majority of the path and indicates that the vehicle is planning a path by driving backwards. The planner refuses to increase the overall cost, in order to set up the

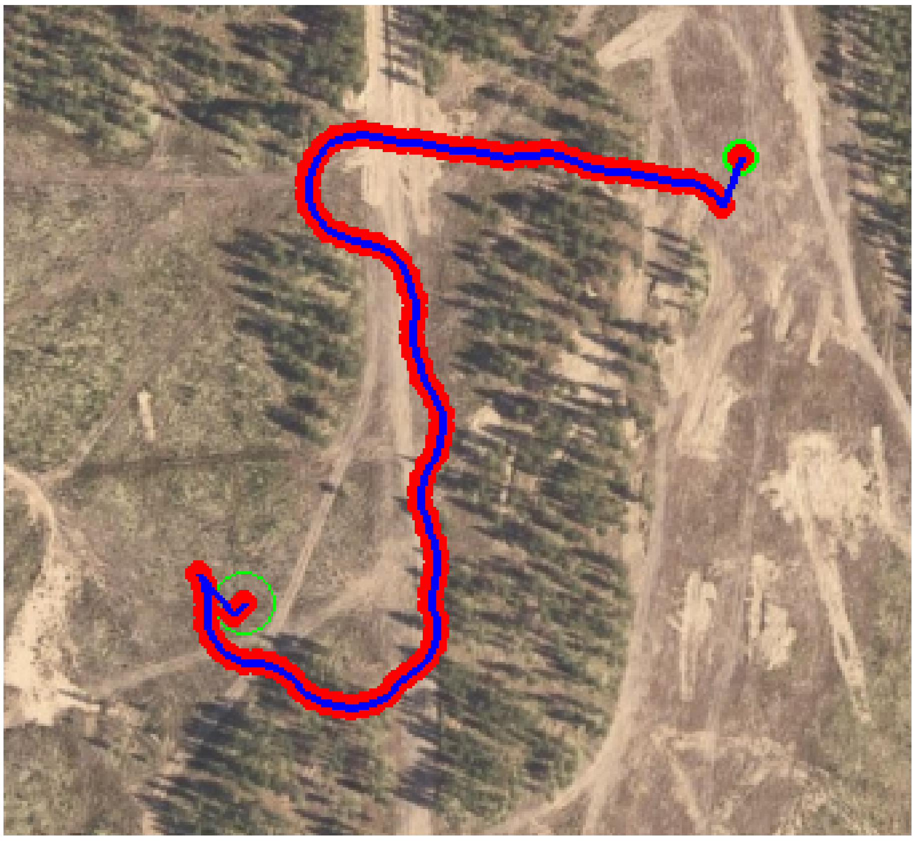


Figure 4.1: Hybrid A* with $\delta = 0.25$, without costs for steering and velocity

vehicle to drive forward. This is certainly not ideal, and the reason of this behavior is that the cost of driving backwards is identical as driving forward. To prevent such kind of unwanted behavior, it is necessary to put a penalty on the vehicle of driving in reverse.

The next experiment, additional costs will be added into the cost function. Here, a penalty of driving reverse is set to 25. In addition, in order to limit the vehicle to steering left and right, a minor cost of steering is set to 2.5. The complete experiment setup can be seen in Table 4.2.

Experiment 2 - Parameter tuning	
Algorithm	Hybrid A* with $\delta = 0.25$
Start coordinates	(458, 164)
End coordinates	(272, 331)
Starting pose	110° deg
Steering	$[-30°, 0°, 30°]$
Cost for steering	[2.5, 0, 2.5]
Velocities	[-5, 20]
Cost for velocities	[25, 0]

Table 4.2: Setup for experiment 2

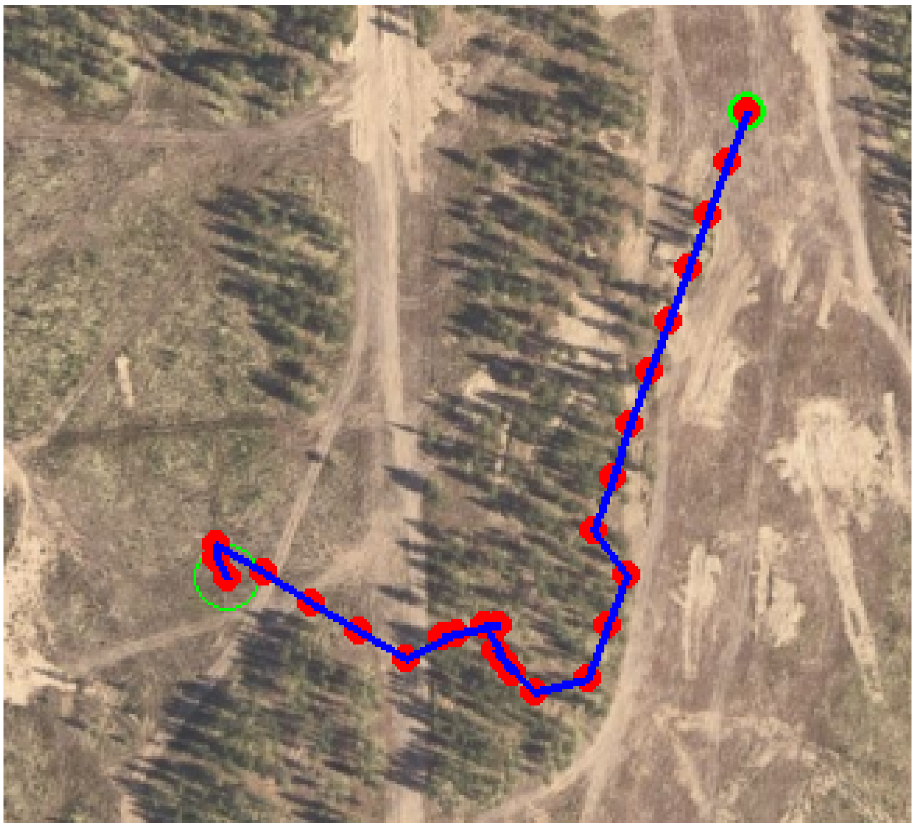


Figure 4.2: Hybrid A* with $\delta = 0.25$, with additional costs for steering and velocity

Figure 4.2 highlights the importance of costs that penalize undesirable behavior, as seen in Figure 4.1. This modification allowed the vehicle to move forward to a larger extent, but it is far from what a human would prefer of driving, due to its planning in a dense and obstacle-filled area.

After more experimenting, the cost of velocities is set to [25, 0], i.e. a penalty of 25 for driving in reverse. As well as assigning a cost of driving in reverse, the proposed method of Hybrid A* algorithm also punishes the vehicle of steering frequently to maintain a more smooth path. The proposed cost for steering is set to [5, 0, 5], which means to assign 5 in cost when steering left or right.

The impact of the vehicle's starting pose θ on the finalized path is worth noting. Consider the same experiment setup as previously. The starting pose has been slightly modified, and will be set to 80° deg. Table 4.3 shows the experiment setup.

Experiment 3 - Parameter tuning	
Algorithm	Hybrid A* with $\delta = 0.25$
Start coordinates	(458, 164)
End coordinates	(272, 331)
Starting pose	80° deg
Steering	$[-30°, 0°, 30°]$
Cost for steering	[2.5, 0, 2.5]
Velocities	[-5, 20]
Cost for velocities	[25, 0]

Table 4.3: Setup for experiment 3

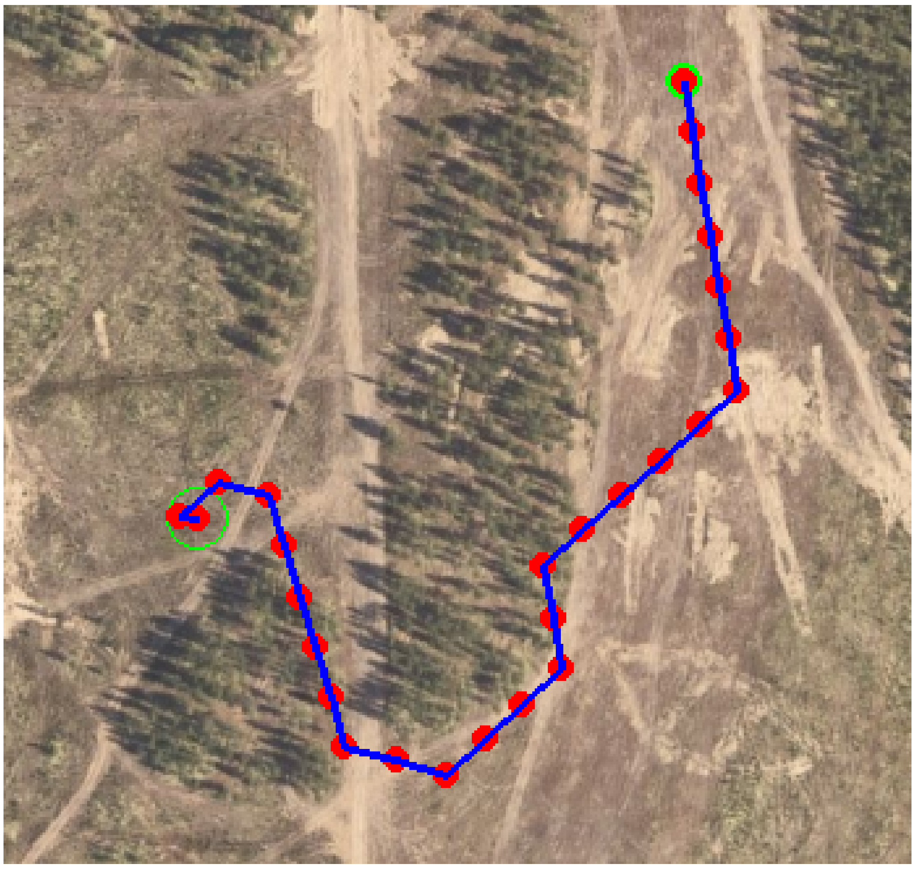


Figure 4.3: Hybrid A* with $\delta = 0.25$, with additional costs for steering and velocity as well as a slightly changed starting pose θ

In terms of behavior, Figure 4.3 indicates an improvement over the previous experiments. By avoiding all wooded areas, the finalized path seems to be much like what a person would drive. When performing tests in later stages, the choice of starting pose θ would be kept in mind.

4.2 Rena

Figure 4.4 shows one of the test runs conducted in Rena. As anticipated, RRT finds a feasible path fastest among the algorithms with a elapsed time of 0.851 seconds. However, the obtained path is more than twice as long, in terms of distance, then the shortest path obtained in Hybrid A* with $\delta = 0$.

Rena run 4	
Start coordinates	(30, 231)
End coordinates	(458, 130)
Starting pose [1]	0° deg
Steering	$[-30°, 0°, 30°]$
Cost for steering	[5, 0, 5]
Velocities	[-5, 20]
Cost for velocities	[25, 0]

Table 4.4: Experiment setup in Rena - Run 4

As can be seen in Table 4.5, the traveled distance increases as δ approaches 1. When δ is zero, the algorithm behaves more or less like an A* algorithm, with the aim of finding the shortest path from point A to point B. When δ rises, it is noticeable that the vehicle intends to drive further and places greater focus on the terrain's steepness, prioritizing low cost, i.e. driving in flat terrain rather than finding the shortest path. In comparison to the Hybrid A* algorithm, the path obtained in RRT is very random, as the tree may changing between runs.

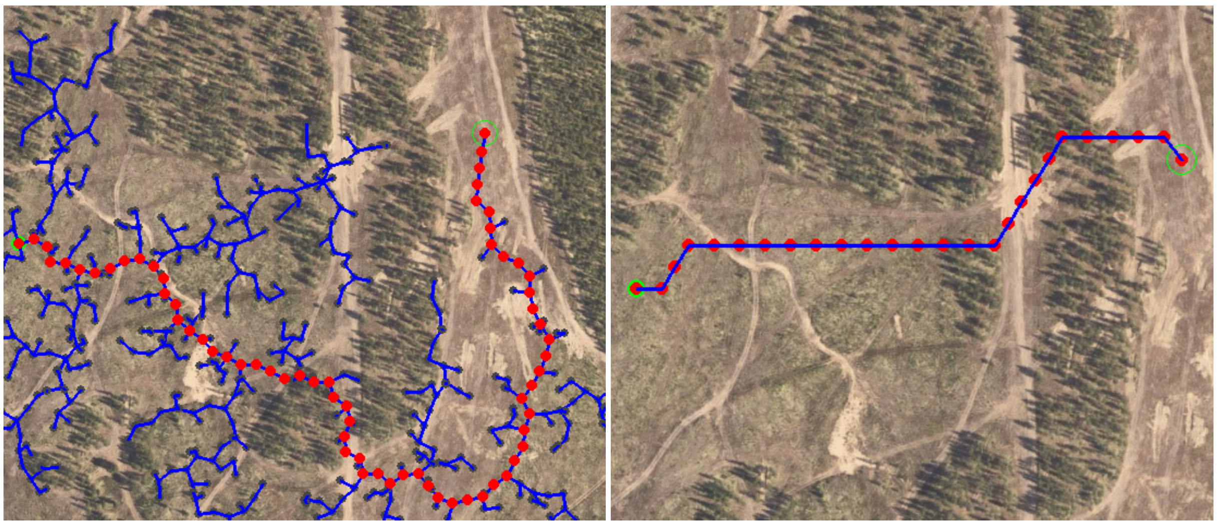
Except when delta is equal to 0.25 and 0.5, every algorithm were able to find a feasible path. This will be addressed in greater detail in the following sections.

Simulated data for run 4 in Rena, Norway				
Algorithm	**Elapsed time**	**Distance**	**Incline** [2]	**Successfully finding a path?**
RRT	0.851 s	970.34 m	16.12 deg	Yes
Hybrid A*, $\delta = 0$	356.2 s	480.0 m	11.45 deg	Yes
Hybrid A*, $\delta = 0.25$	-	-	-	No
Hybrid A*, $\delta = 0.5$	-	-	-	No
Hybrid A*, $\delta = 0.75$	0.827 s	760.0 m	12.73 deg	Yes
Hybrid A*, $\delta = 1$	0.839 s	720.0 m	10.73 deg	Yes

Table 4.5: Simulation data for Figure 4.4

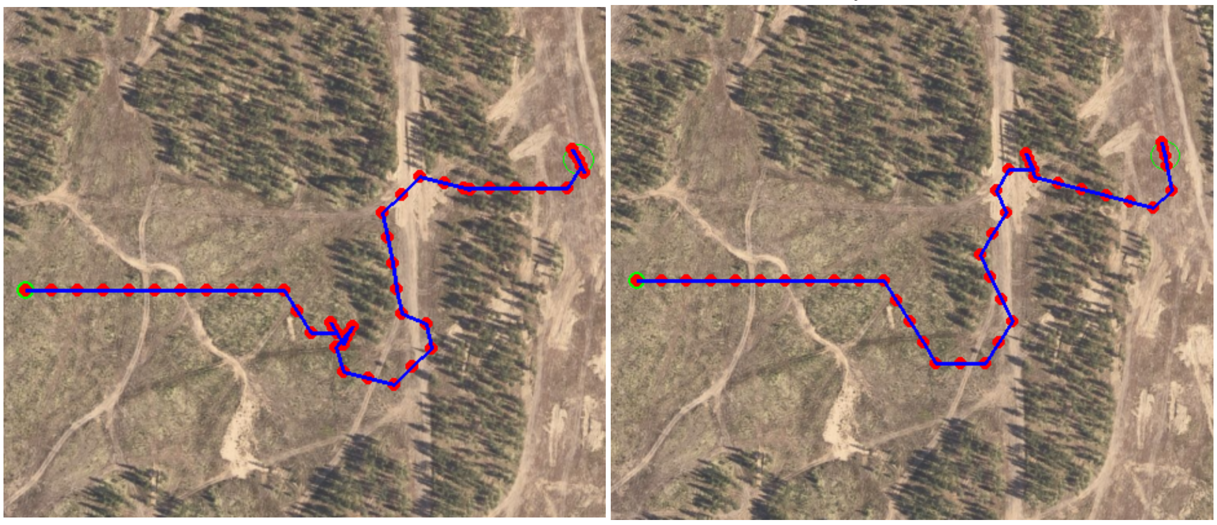
[1]Not applicable for RRT, since this algorithm does not include vehicle pose.

[2]Total absolute incline. The incline was measured using a Sobel image and linear interpolation of 20 samples between two nodes. See Listing B.3 to view the code.



(a) RRT

(b) Hybrid A*, $\delta = 0$



(c) Hybrid A*, $\delta = 0.75$

(d) Hybrid A*, $\delta = 1$

Figure 4.4: One of the iterations in Rena. $\delta = 0.25$ and $\delta = 0.5$ failed to find a feasible path.

Figure 4.4 shows the different paths obtained in the experiment. By observing the results, one may argue that the path obtained in RRT is the most optimal one in term of safeness and most likely what a person would like to drive. However, as seen in Table 4.5, the incline is lower by choosing a path through the forest. This will be further addressed in later sections.

4.3 Hardangervidda

Similar experiments are carried out in a more challenging setting. The setting for this segment will be in Hardangervidda, Norway, a mountainous region with numerous gradient variations. Examining the performance when steepness is taken into account would be particularly interesting. There may be further differences in the found solution due to the incredibly difficult environment with many steepness variations.

Hardangervidda run 2	
Start coordinates	(64, 91)
End coordinates	(143, 281)
Starting pose [3]	0° deg
Steering	$[-30°, 0°, 30°]$
Cost for steering	[5, 0, 5]
Velocities	[-5, 20]
Cost for velocities	[25, 0]

Table 4.6: Experiment setup in Hardangervidda - Run 2

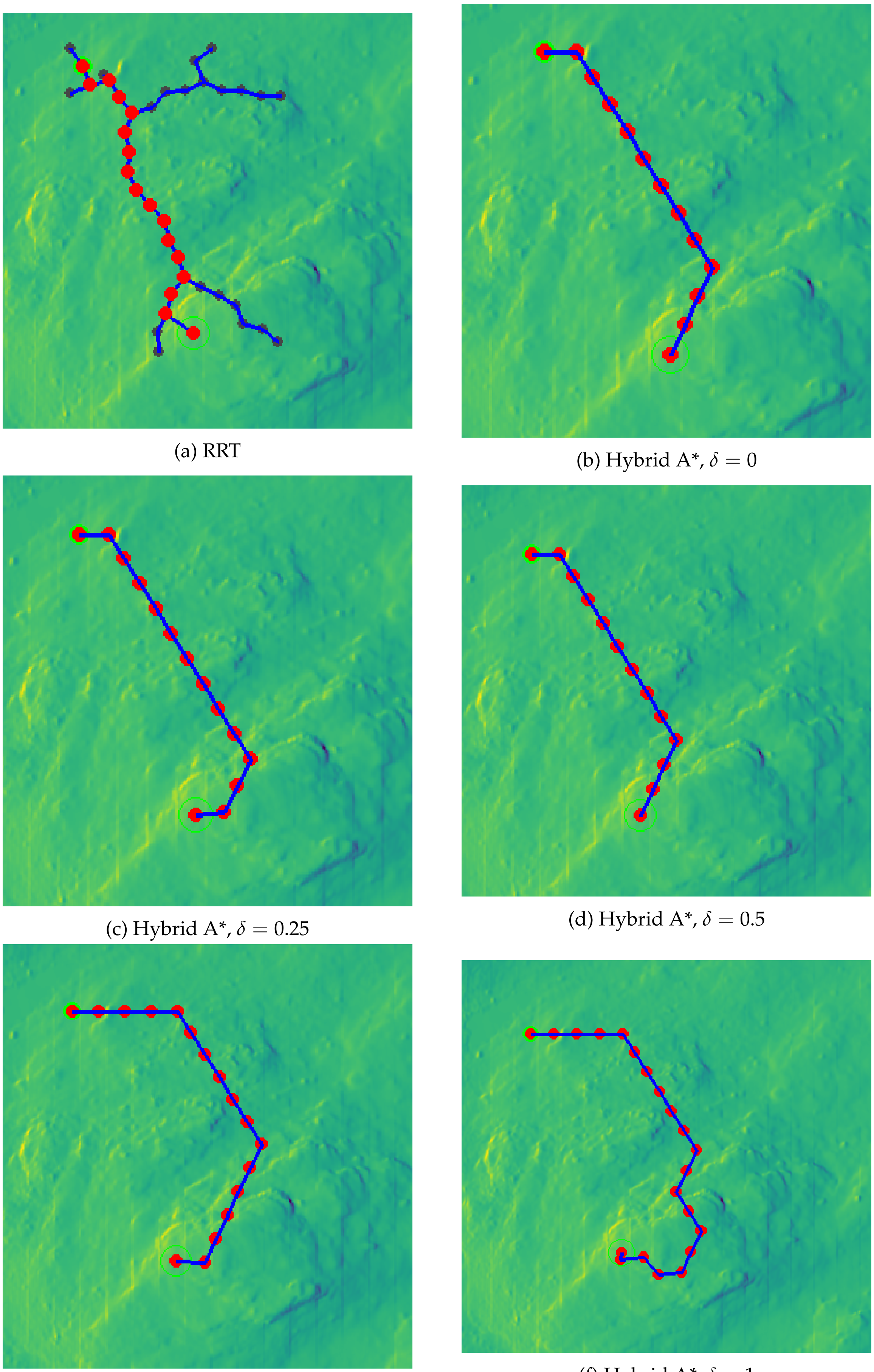
With this shift in environment, all algorithms were able to plan a solution path, with a success rate of 100% after one iteration.

Simulated data for one run in Hardangervidda, Norway				
Algorithm	**Elapsed time**	**Distance**	**Incline** [4]	**Successfully finding a path?**
RRT	0.442 s	240,86 m	37,91 deg	Yes
Hybrid A*, $\delta = 0$	1.191 s	220.0 m	17.49 deg	Yes
Hybrid A*, $\delta = 0.25$	0.566 s	240.0 m	21.96 deg	Yes
Hybrid A*, $\delta = 0.5$	0.475 s	220.0 m	17.49 deg	Yes
Hybrid A*, $\delta = 0.75$	0.548 s	300.0 m	11.37 deg	Yes
Hybrid A*, $\delta = 1$	0.794 s	380.0 m	3.19 deg	Yes

Table 4.7: Simulation data for Figure 4.5

[3]Not applicable for RRT, since this algorithm does not include vehicle pose.

[4]Total absolute incline. The incline was measured using a Sobel image and linear interpolation of 20 samples between two nodes. See Listing B.3 to view the code.


(a) RRT

(b) Hybrid A*, $\delta = 0$

(c) Hybrid A*, $\delta = 0.25$

(d) Hybrid A*, $\delta = 0.5$

(e) Hybrid A*, $\delta = 0.75$

(f) Hybrid A*, $\delta = 1$

Figure 4.5: One of the iterations in Hardangervidda

In this experiment, the proposed methods are demonstrated in an environment with a wide variety of gradient variations. Due to the elevation variations in the terrain, the impact of using an incline-based cost function is much more emphasized in Hardangervidda. As δ increases, one can see better results in terms of finding less steep paths. The results are illustrated in the Sobel map in Figure 4.5 and the corresponding earth map can be found in Appendix A.

The discrepancies between these algorithms can be seen after performing experiments in two different environments. In the following pages, the simulated results will be further clarified.

4.4 Analysis

The aim of this section is to dig deeper into the data collected and presented in the previous chapter. The behavior of the presented algorithms will be compared in order to answer the research questions presented in Chapter 1.2, especially research question 3, i.e. identify performance metrics for the two motion planning algorithms for evaluation.

The key figures that are used for describing the quality of the path are:

1. **Mean elapsed time** - How long does it take the algorithm to come up with a solution?
2. **Mean distance** - How far does the vehicle travel to arrive at the destination?
3. **Mean incline** - What is the slope of the path?
4. **Success rate** - What is the probability that the algorithm can find a solution?

After conducting experiments in two different settings, the differences between these algorithms are evident after 10 simulated drives for each environment. Table 4.8 shows the statistics of the conducted experiments in Rena. It is worth noting that the success rate has an effect on the mean values. If an algorithm fails to find a correct path, the mean value will include fewer samples, which can be misleading when evaluating the results.

Simulated driving in Rena, Norway				
Algorithm	**Mean elapsed time**	**Mean distance**	**Mean incline**	**Success rate**
RRT	0.51 s	441.94 m	14.90 deg	100.0%
Hybrid A*, $\delta = 0$	52.34 s	248.0 m	14.02 deg	100.0%
Hybrid A*, $\delta = 0.25$	0.50 s	277.86 m	12.19 deg	70.0%
Hybrid A*, $\delta = 0.5$	0.55 s	325.63 m	15.93 deg	80.0%
Hybrid A*, $\delta = 0.75$	0.59 s	341.11 m	13.06 deg	90.0%
Hybrid A*, $\delta = 1$	0.63 s	330.63 m	13.96 deg	80.0%

Table 4.8: Monte Carlo simulation in Rena: 10 iterations

The first thing to note is how quickly RRT comes up with a solution. In Table 4.8, RRT solves the problem 102.62 times faster than the classical A*, i.e. Hybrid A* with $\delta = 0$. However, while this approach may not be ideal in terms of effort or fuel consumption, it is feasible. In

Rena run 2	
Start coordinates	(148, 173)
End coordinates	(200, 340)
Starting pose [5]	0° deg
Steering	$[-30°, 0°, 30°]$
Cost for steering	[5, 0, 5]
Velocities	[-5, 20]
Cost for velocities	[25, 0]

Table 4.9: Experiment setup in Rena - Run 2

comparison, the mean distance driven in RRT is 1.78 times longer than $\delta = 0$, 1.36 times longer than $\delta = 0.5$ and 1.34 times longer than $\delta = 1$.

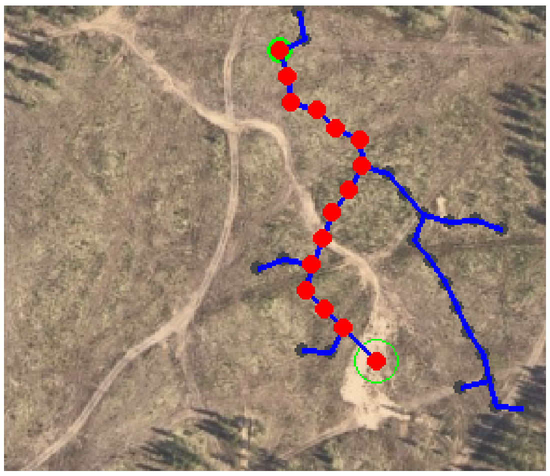
Whenever $\delta = 0.25$ or $\delta = 0.5$, the found paths has the tendency to drive in circles when close to the goal, as illustrated in Figure 4.6. This has to do with the cost function, as the algorithm can be confused if both distance and incline are taken into account at the same time. Furthermore, the motion primitives may be too high in term of velocity combined with steering angle, requiring the vehicle to drive in reverse to reach the desired destination. This unfortunate behavior has a negative impact on the average distance and incline. Due to how incline is calculated, whenever the vehicle is driving in reverse, the incline will be wrongly measured because of the fixed step of 20 samples in between nodes.

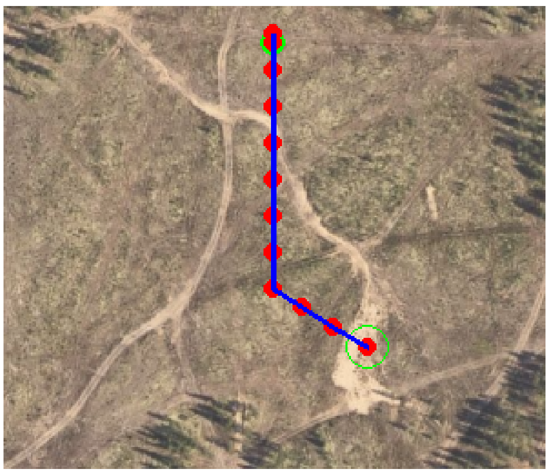
Simulated driving in Hardangervidda, Norway				
Algorithm	**Mean elapsed time**	**Mean distance**	**Mean incline**	**Success rate**
RRT	0.52 s	439.93 m	13.04 deg	100.0%
Hybrid A*, $\delta = 0$	59.03 s	290.50 m	14.92 deg	100.0%
Hybrid A*, $\delta = 0.25$	0.60 s	425.0 m	12.61 deg	100.0%
Hybrid A*, $\delta = 0.5$	0.59 s	419.0 m	12.98 deg	100.0%
Hybrid A*, $\delta = 0.75$	0.67 s	427.5 m	12.69 deg	100.0%
Hybrid A*, $\delta = 1$	0.91 s	516.0 m	9.35 deg	100.0%

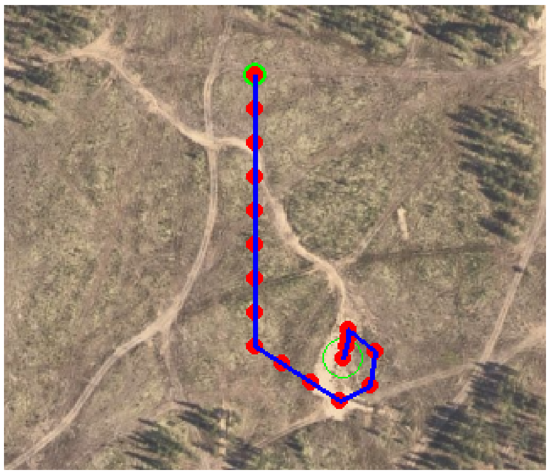
Table 4.10: Monte Carlo simulation in Hardangervidda: 10 iterations

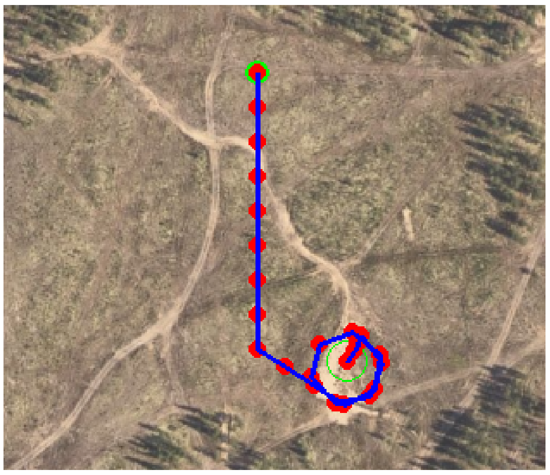
In contrast, all algorithms were able to plan a solution path in Hardangervidda, with a outstanding success rate of 100 % after 10 iterations. Because of the significant elevation differences in the landscape, the influence of choosing an incline-based cost is much more emphasized, as indicated before in chapter 4.3. The final results after 10 simulations can be seen in Table 4.10.

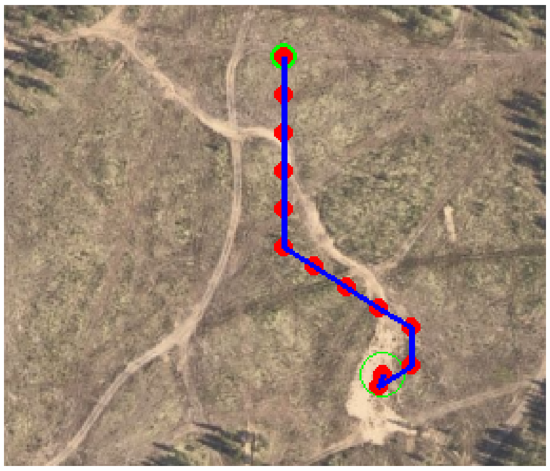
[5]Not applicable for RRT, since this algorithm does not include vehicle pose.


(a) RRT


(b) Hybrid A*, $\delta = 0$


(c) Hybrid A*, $\delta = 0.25$


(d) Hybrid A*, $\delta = 0.5$


(e) Hybrid A*, $\delta = 0.75$

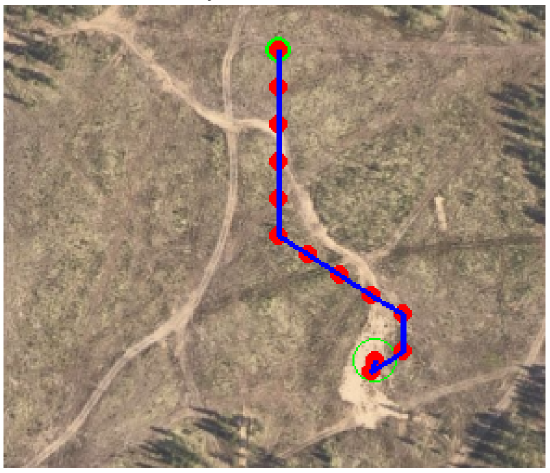

(f) Hybrid A*, $\delta = 1$

Figure 4.6: Run 2. Unexpected behavior with $\delta = 0.25$ and $\delta = 0.5$

Chapter 5

Discussion

This chapter summarizes the analysis that has been presented. Various difficulties faced during the thesis will also be addressed. A summary of possible work that has been proposed will also be included in this section.

5.1 Validation

An evaluation of the study is important in order to examine and bring out any issues influencing its validity.

5.1.1 Unexpected behavior from path planning

During the simulation phase, the paths generated by the various motion planning methods seemed to have a few unexpected behaviors. First, behavior such as in Figure 4.6d where the vehicle is driving in circle prior to reaching the goal destination should be avoided in motion planning. One possible solution may be to reduce the vehicle's velocity as well as the steering angle so that it is possible to drive slower when reaching the desired destination, e.g.

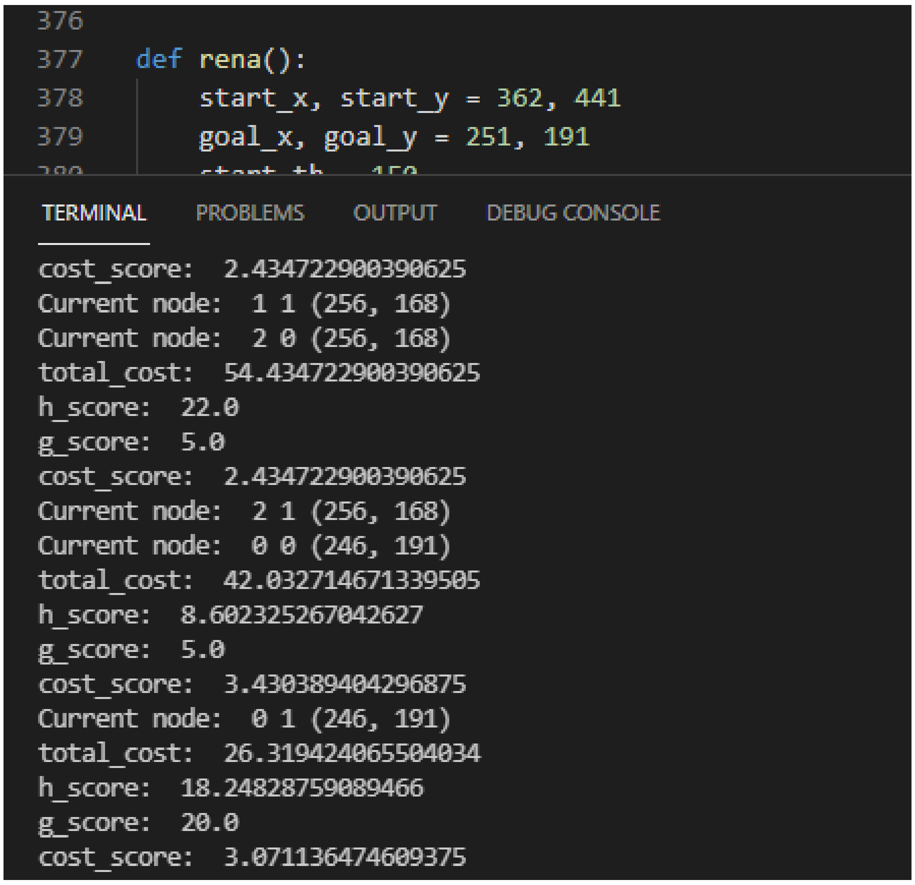
Example of motion primitives	
Steering	$[-30°, -15°, 0°, 15°, 30°]$
Velocities	[-5, 5, 10, 20]

Table 5.1: Motion primitives

Notably, finding a feasible path is preferable to being stopped by the termination criteria. In the last experiment in Rena, Hybrid A* algorithm with $\delta = 1$, struggled to find a goal, and alternated between motion primitives near the goal. Figure 5.1 illustrates the search before it got terminated.

Second, the vehicle has an tendency of driving in reverse in order to get to the desired destination, illustrated in Figures 4.4c and 4.6c. This behavior might also be fixed by adapting new motion primitives as shown in Table 5.1.

Lastly, in Figure 4.4, all algorithms except when δ is equal to 0.25 and 0.5, were able to find a feasible path. The path planning algorithms were stuck inside the forest in the upper right of the image, since the total cost were lower in there because of the heuristics function $H(n)$. As a result, the search was terminated after 15 minutes according to the termination criteria.



```python

def rena():
    start_x, start_y = 362, 441
    goal_x, goal_y = 251, 191
```

```
cost_score:  2.434722900390625
Current node:  1 1 (256, 168)
Current node:  2 0 (256, 168)
total_cost:  54.434722900390625
h_score:  22.0
g_score:  5.0
cost_score:  2.434722900390625
Current node:  2 1 (256, 168)
Current node:  0 0 (246, 191)
total_cost:  42.032714671339505
h_score:  8.602325267042627
g_score:  5.0
cost_score:  3.430389404296875
Current node:  0 1 (246, 191)
total_cost:  26.319424065504034
h_score:  18.24828759089466
g_score:  20.0
cost_score:  3.071136474609375
```

Figure 5.1: Hybrid A* with $\delta = 1$, struggling to find a path

5.1.2 Obstacle map of Hardangervidda

The obstacle map of Hardangervidda was created using the Sobel operator, as mentioned in Chapter 3.2.3. However, it seems unreasonable to set the threshold of the gradient to be 3 meters, as the vehicle may not be capable of driving in such high elevation differences. This mistake must be considered in future research, with the threshold possibly being tuned down to 1 meter.

5.1.3 Visualization

The nodes are saved in a list when a path is found using the Hybrid A* method. However, the nodes are connected by edges in a linear fashion for visualization, which is incorrect due to the tree's extension in terms of steering angles. This is a minor flaw for the visualization part in Hybrid A*.

5.1.4 Calculating incline

Related to the previous section, the incline is calculated using the list of nodes after a path is found. By using linear interpolation, the captured incline is not aligned with the actual incline (the curved line as shown in Figure 2.9), resulting in somewhat misleading results that are used for analysis.

Furthermore, the total absolute incline is affected by one significant issue that was covered before in Chapter 4.4 regarding computing the inclination when driving backwards. Since the motion primitives when driving backwards are 1/4 the length of driving forward, i.e. [-5, 20], the interpolation of 20 samples will therefore be an issue. Ideally, whenever the vehicle is driving backwards, the number of samples could for example decreased to 5 samples instead of being constant at 20.

5.1.5 Simplified vehicle model

Due to the challenges around retrieving data from hoydedata.no, which will be discuss later in Chapter 5.2.1, the bicycle model utilized in this thesis is a little underdeveloped. This is related to the map's difficulty in obtaining accurate distance readings without taking in consideration of the provided metadata. In addition, since the original data were scaled from 0.25 point density to 1.05 due to display purposes, the distances may be measured inaccurately. As a result, the length of the vehicle model is set to 0.2 m, making it seem as a dot in the environment.

5.2 Encountered challenges

Throughout the thesis, there were a number of significant difficulties faced during the execution. This section will go through the challenges to be aware of.

5.2.1 Retrieving data from hoydedata

When retrieving data from hoydedata.no as explained in Chapter 3.2.1, the data might sometimes be corrupted. The reason for this may be that the current map segment crosses the GeoTIFF data, which is divided into minor areas. Therefore, the field of selection is something to be aware of. A lot of trial and error has been done in order to find a suitable field that has data that can be used for further image processing.

Moreover, when downloading data from hoydedata.no, a desired set of data, in this case DSM and DTM, can be received. In addition, metadata is provided, which can be used to obtain information about the terrain and the field's coordinates. Due to time limitation, metadata is not used in this thesis. The procedure for matching the obstacle map to the current map, on the other hand, is achieved by trial and error using human eyes. This might not be ideal for further research.

5.2.2 Making a cost function

The cost function is essential for motion planning. All searching based approaches are based on a cost function in order to plan a path. As explained in Chapter 3.3.2, the searching is based on minimizing a specific cost function for some variables, for instance distance. Unquestionably, it is difficult to establish a perfect cost function, which has been analyzed and modified during the thesis. Some parameters are only dependent on an integer, such as punishing the vehicle for steering left or right or restricting the vehicle's ability to drive backwards. As a result, this is something that should be investigated further for future work.

5.2.3 Execution time for Hybrid A*

During the experiments, there were several concerns with the implementation's execution time. After experimenting with various initial states, it was found out that the start position direction, i.e. starting pose θ, has a significant effect on the execution time. The Hybrid A* algorithm must look for the best possible path to turn to the target whenever the starting pose is not headed towards the goal states. This process could take up to 15 minutes, especially when δ is equal to zero. As a result, whenever the vehicle is heading towards the target, the proposed method has an easier time finding a path.

5.3 Future work

Other research questions for future work have arisen as a result of the findings presented in this study.

First, as mentioned previously, the cost function is essential for motion planning since all searching based methods are based on minimizing a cost function in order to plan a path. In attempt to develop paths that are more human-like, in contrast to what is discussed in Chapter 5.1.1, various parameters must be fine-tuned and researched further, as pointed out in Chapter 5.2.2. The idea of implementing cost dynamically using LIDAR, is something worth investigating when planning a real experiment [6].

Second, investigating the usage of metadata in the environment design increases the simulation's validity. It is possible to conduct more realistic experiments/simulations by combining georeferencing data with a more comprehensive vehicle model. The bicycle concept has shown to be an effective and easy approach for car-like vehicles. However, by incorporating a thorough kinematic model of the vehicle, such as the engine's forces, it will be easier to monitor the vehicle's intended path. This enables for further experiments and perhaps even real-world experiments using UGV provided by FFI.

Third, to increase safety of the planned trajectories, one may consider implementing the obstacle grid's Voronoi graph as done in research papers, such as [4] [6]. By planning a path that is farthest away from barriers, the car will safely avoid obstacles, which is especially crucial in motion planning algorithms in unfamiliar landscape.

Finally, besides the two environments, it could be interesting to test the proposed methods in other settings. This way, one may obtain a better understanding of the cost function and possibly fine-tune it to improve the results.

Chapter 6

Conclusion

The main focus of this research is regarding motion planning for autonomous off-road driving. Based on previous research about motion planning algorithms, this thesis' problem statement can be formulated as follows:

1. Are there any motion planning methods that are particularly suitable for this specific task?
2. Investigate adaptation possibilities and utilize this for finding close to realistic and optimal paths.
3. Identify performance metrics for the two motion planning algorithms for evaluation.

When it comes to preparing to drive in rough terrain, nothing is simple. Due to characteristics such as inclination and terrain roughness, separating the environment into obstacles space and free space is insufficient when driving in a challenging environment. The environment in this thesis is constructed using a combination of terrain knowledge and an obstacle map. This enables the proposed methods to utilize the terrain characteristics directly to investigate optimality conditions in terrain.

The sampling based method RRT is chosen due to its ability to find a feasible path efficiency with regard to computational effort. On the other hand, Hybrid A* has been chosen due to its extensive possibilities for developing a cost function that takes into account steepness, curvature and distance.

The experiments were carried out in two different ecosystems, Rena and Hardangervidda. Especially in a open and flat place like Rena, the Hybrid A* algorithm with higher δ values struggled to find a path, due to the minor elevation difference in the landscape. Here, RRT and Hybrid A* algorithm with $\delta = 0$, both yielded great results with a success rate of 100% after 10 iterations. In terms of examining optimality conditions, one may argue that RRT has a tendency to find paths that are more safe and likely to be what a person would prefer to drive. Hybrid A* with $\delta = 0$, on the other hand, takes significantly longer to approximate the shortest path. Although the paths obtained in RRT may differ significantly between iterations, RRT and Hybrid A* with $\delta = 0$ might be the best choice in a flat surface such as in Rena.

In Hardangervidda, on the other hand, all algorithms were able to develop a solution path with a success rate of 100% after 10 iterations. The impact of choosing an incline-based cost is significantly more emphasized due to the large elevation difference in the landscape. In this case, choosing the path with the least amount of incline is more important than finding the shortest path. For this reason, it is reasonable to argue that Hybrid A* with $\delta = 1$ is the optimal planner in this environment among the presented algorithms in the thesis.

The goal of this study is to compare two motion planning algorithms and determine an appropriate path that may be utilized as a rough global planner before moving on to local search, as done in [15]. Nevertheless, there is potential for improvement as discussed in Chapter 5.3 and it may be beneficial to include more motion primitives in order to prevent unexpected behavior as discussed in Chapter 5.1.1.

Bibliography

[1] Mouad Boumediene. *GitHub - mouad-boumediene/python-visualization-of-the-RRT-algorithm-with-pygame*. Mar. 2021. URL: https://github.com/mouad-boumediene/python-visualization-of-the-RRT-algorithm-with-pygame (visited on 07/05/2021).

[2] *DARPA Urban Challenge*. 2007. URL: https://archive.darpa.mil/grandchallenge/ (visited on 02/05/2021).

[3] Yan Ding. *Simple Understanding of Kinematic Bicycle Model | by Yan Ding | Medium*. Feb. 2020. URL: https://dingyan89.medium.com/simple-understanding-of-kinematic-bicycle-model-81cac6420357 (visited on 22/05/2021).

[4] Dmitri Dolgov, Michael Montemerlo and James Diebel. 'Practical Search Techniques in Path Planning for Autonomous Driving'. In: *AAAI Workshop - Technical Report* (Jan. 2008).

[5] E W Drrksrra. *A Note on Two Problems in Connexion with Graphs*. Tech. rep., pp. 269–296.

[6] Dennis Fassbender, André Mueller and Hans Joachim Wuensche. 'Trajectory planning for car-like robots in unknown, unstructured environments'. In: *IEEE International Conference on Intelligent Robots and Systems* Iros (2014), pp. 3630–3635. ISSN: 21530866. DOI: 10.1109/IROS.2014.6943071.

[7] Marta Palau Franco. *euRathlon 2013 land robotics competition – Day Four recap | Robohub*. Sept. 2013. URL: https://robohub.org/eurathlon-2013-land-robotics-competition-day-four-recap/ (visited on 02/05/2021).

[8] GeeksforGeeks. *A* Search Algorithm - GeeksforGeeks*. 2021. URL: https://www.geeksforgeeks.org/a-search-algorithm/ (visited on 04/05/2021).

[9] Geoimage. *Mining Satellite Digital Elevation Models*. URL: https://www.geoimage.com.au/_blog/News/post/high-demand-for-digital-elevation-models-from-satellite/ (visited on 25/04/2021).

[10] *GISInternals Support Site*. 2021. URL: https://www.gisinternals.com/query.html?content=filelist&file=release-1928-x64-gdal-3-2-1-mapserver-7-6-2.zip (visited on 28/04/2021).

[11] Peter E. Hart, Nils J. Nilsson and Bertram Raphael. *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*. 1968. DOI: 10.1109/TSSC.1968.300136.

[12] Kris Hauser. *Robotic Systems (draft) - Chapter 10*. 2018. URL: http://motion.pratt.duke.edu/RoboticSystems/.

[13] *Høydedata*. URL: https://hoydedata.no/LaserInnsyn/ (visited on 26/04/2021).

[14] Yonghoon Ji et al. 'Adaptive Motion Planning Based on Vehicle Characteristics and Regulations for Off-Road UGVs'. In: *IEEE Transactions on Industrial Informatics* 15.1 (2019), pp. 599–611. DOI: 10.1109/TII.2018.2870662.

[15] Julian Jordan and Andreas Zell. 'Real-time model based path planning for wheeled vehicles'. In: *Proceedings - IEEE International Conference on Robotics and Automation* 2019-May (2019), pp. 5787–5792. ISSN: 10504729. DOI: 10.1109/ICRA.2019.8794133.

[16] Sertac Karaman and Emilio Frazzoli. 'Incremental sampling-based algorithms for optimal motion planning'. In: *Robotics: Science and Systems* 6 (2011), pp. 267–274. ISSN: 2330765X. DOI: 10.15607/rss.2010.vi.034. arXiv: 1005.0416.

[17] Mohit Kaushik. *Reading and Visualizing GeoTiff | Satellite Images with Python | by Mohit Kaushik | Towards Data Science*. 2020. URL: https://towardsdatascience.com/reading-and-visualizing-geotiff-images-with-python-8dcca7a74510 (visited on 27/04/2021).

[18] Lydia Kavraki et al. 'Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces'. In: *Robotics and Automation, IEEE Transactions on* 12 (Sept. 1996), pp. 566–580. DOI: 10.1109/70.508439.

[19] Sebastian Lague. *Pathfinding/Pseudocode at master · SebLague/Pathfinding · GitHub*. 2016. URL: https://github.com/SebLague/Pathfinding/blob/master/Episode%2001%20-%20pseudocode/Pseudocode (visited on 04/05/2021).

[20] Steven M. LaValle and James J. Kuffner. 'Rapidly-Exploring Random Trees: Progress and Prospects'. In: (1998). URL: http://msl.cs.uiuc.edu/~lavalle/papers/LavKuf01.pdf.

[21] Sang Uk Lee, Ramon Gonzalez and Karl Iagnemma. 'Robust sampling-based motion planning for autonomous tracked vehicles in deformable high slip terrain'. In: *Proceedings - IEEE International Conference on Robotics and Automation* 2016-June (2016), pp. 2569–2574. ISSN: 10504729. DOI: 10.1109/ICRA.2016.7487413.

[22] Haoyue Liu et al. 'Goal-biased Bidirectional RRT based on Curve-smoothing'. In: *IFAC-PapersOnLine* 52.24 (2019). 5th IFAC Symposium on Telematics Applications TA 2019, pp. 255–260. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2019.12.417. URL: https://www.sciencedirect.com/science/article/pii/S2405896319323250.

[23] Hui Liu. 'Chapter 1 - Introduction'. In: *Robot Systems for Rail Transit Applications*. Ed. by Hui Liu. Elsevier, 2020, pp. 1–36. ISBN: 978-0-12-822968-2. DOI: https://doi.org/10.1016/B978-0-12-822968-2.00001-2. URL: https://www.sciencedirect.com/science/article/pii/B9780128229682000012.

[24] *Mars 2020 Perseverance Rover - NASA Mars*. URL: https://mars.nasa.gov/mars2020/ (visited on 01/05/2021).

[25] Kim Mathiassen et al. 'Development of an Autonomous Off-Road Vehicle for Surveillance Missions'. In: *Proceedings of IST-127/RSM-003 Specialists' Meeting on Intelligence & Autonomy in Robotics*. DOI: 10.14339/STO-MP-IST-127, NATO UNCLASSIFIED. NATO Science and Technology Organization. Bonn, Germany, Oct. 2016. DOI: 10.14339/STO-MP-IST-127.

[26] Kim Mathiassen et al. 'Making the Milrem Themis UGV ready for autonomous operations'. In: *Unmanned Systems Technology XXIII*. Ed. by Paul L. Muench, Hoa G. Nguyen and Brian K. Skibba. SPIE, Apr. 2021. DOI: 10.1117/12.2585879. URL: https://doi.org/10.1117/12.2585879.

[27] *Motion planning in workspaces and in configuration spaces. The left... | Download Scientific Diagram*. URL: https://www.researchgate.net/figure/Motion-planning-in-workspaces-and-in-configuration-spaces-The-left-figure-shows_fig2_281389394 (visited on 01/05/2021).

[28] Brian Paden et al. 'A survey of motion planning and control techniques for self-driving urban vehicles'. In: *IEEE Transactions on Intelligent Vehicles* 1.1 (2016), pp. 33–55. ISSN: 23798858. DOI: 10.1109/TIV.2016.2578706. arXiv: 1604.07446.

[29] Jia Pan and Dinesh Manocha. 'Efficient Configuration Space Construction and Optimization for Motion Planning'. In: *Engineering* 1.1 (2015), pp. 046–057. ISSN: 20958099. DOI: 10.15302/J-ENG-2015009. URL: http://dx.doi.org/10.15302/J-ENG-2015009.

[30] *Pathfinding 2D Illustration - Pathfinding - Wikipedia*. URL: https://en.wikipedia.org/wiki/Pathfinding#/media/File:Pathfinding_2D_Illustration.svg (visited on 01/05/2021).

[31] Ahmed Hussain Qureshi and Yasar Ayaz. 'Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments'. In: *arXiv* (2017). ISSN: 23318422. arXiv: arXiv:1703.08944v1.

[32] Bruno Siciliano et al. *Robotics: Modelling, Planning and Control*. 1st. Springer Publishing Company, Incorporated, 2008. ISBN: 1846286417.

[33] Saurabh Singh. *Difference between DEM/DTM and DSM*. URL: https://www.gisresources.com/confused-dem-dtm-dsm (visited on 30/04/2021).

[34] *Sobel Edge Detection - an overview | ScienceDirect Topics*. URL: https://www.sciencedirect.com/topics/engineering/sobel-edge-detection (visited on 28/04/2021).

[35] *songl/Masterthesis: Comparing the motion planning methods Hybrid A* and RRT for autonomous off-road driving of bicycle vehicles*. URL: https://github.uio.no/songl/Masterthesis (visited on 22/05/2021).

[36] Justin Svegliato. *How does a robot plan a path using RRT? | by Justin Svegliato | Towards Data Science*. 2020. URL: https://towardsdatascience.com/how-does-a-robot-plan-a-path-in-its-environment-b8e9519c738b (visited on 03/05/2021).

[37] Reiya Takemura and Genya Ishigami. 'Traversability-based RRT for planetary rover path planning in rough terrain with LIDAR point cloud data'. In: *Journal of Robotics and Mechatronics* 29.5 (2017), pp. 838–846. ISSN: 18838049. DOI: 10.20965/jrm.2017.p0838.

[38] Marius Thoresen et al. 'Path Planning for UGVs based on Traversability Hybrid A*'. In: *IEEE Robotics and Automation Letters* (2021). DOI: 10.1109/lra.2021.3056028.

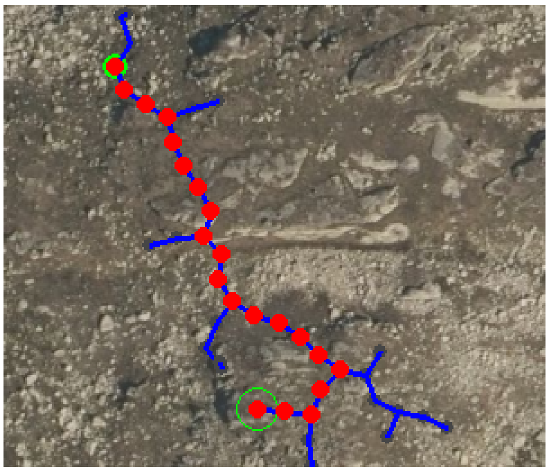
[39] Jasvir Virdi. *GitHub - jvirdi2/$A_star_andH_ybridA_star$*. Mar. 2020. URL: https://github.com/jvirdi2/A_star_and_Hybrid_A_star (visited on 07/05/2021).

[40] Saloni Walimbe. *The Role of Autonomous Unmanned Ground Vehicle Technologies in Defense Applications - Aerospace & Defense Technology*. Oct. 2020. URL: https://www.aerodefensetech.com/component/content/article/adt/features/articles/37888 (visited on 01/05/2021).

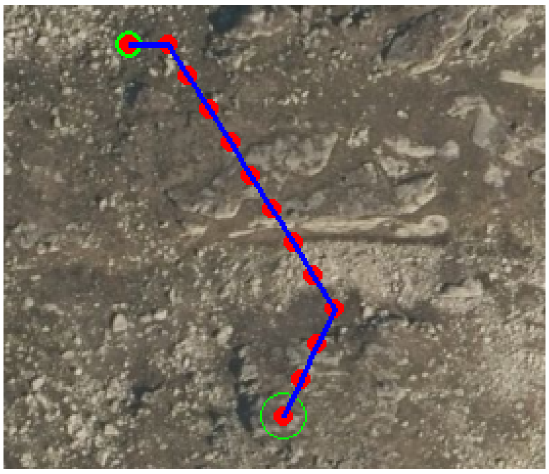
[41] Frank Warmerdam and Even Rouault. *GDAL — GDAL documentation*. 2021. URL: https://gdal.org/ (visited on 26/04/2021).

Appendix A

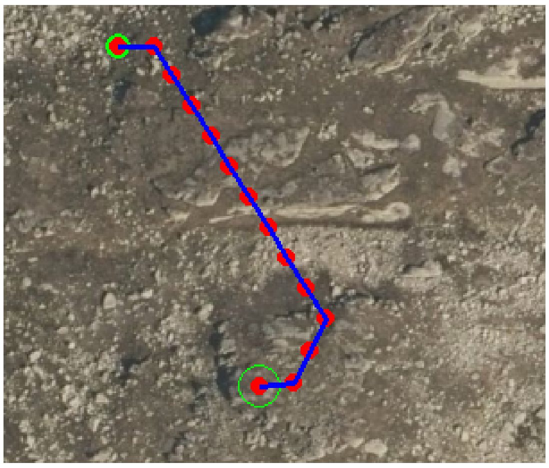
Simulation - Hardangervidda



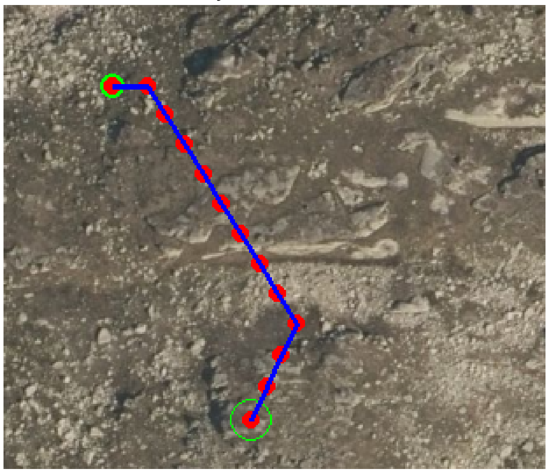
(a) RRT



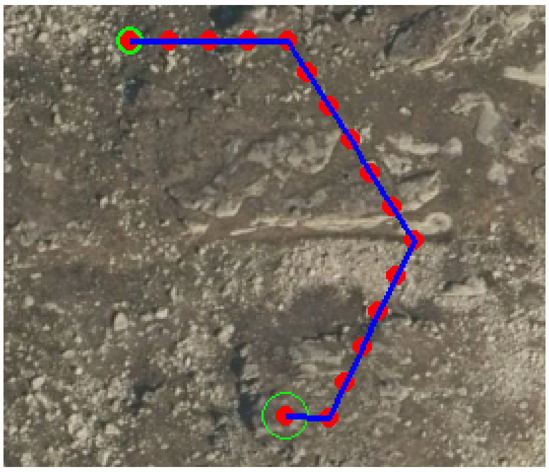
(b) Hybrid A*, $\delta = 0$



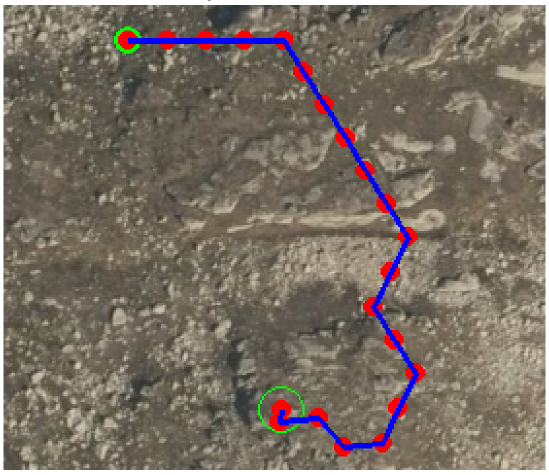
(c) Hybrid A*, $\delta = 0.25$



(d) Hybrid A*, $\delta = 0.5$



(e) Hybrid A*, $\delta = 0.75$



(f) Hybrid A*, $\delta = 1$

Figure A.1: Run 2 - Hardangervidda

Appendix B

Map analysis

B.1 Rena

```python
from osgeo import gdal
import numpy as np
from scipy import ndimage, misc
import scipy
import cv2


## Changing resolution - original res is 0.25
gdal.Warp('rena/rena1/dom/DOM_32-1-520-164-61_new.tif', 'rena/rena1/dom/DOM_32
    -1-520-164-61.tif', xRes=1.05, yRes=1.05)
gdal.Warp('rena/rena1/dtm/DTM_32-1-520-164-61_new.tif', 'rena/rena1/dtm/DTM_32
    -1-520-164-61.tif', xRes=1.05, yRes=1.05)


dataset_DOM = gdal.Open('rena/rena1/dom/DOM_32-1-520-164-61_new.tif', gdal.
    GA_ReadOnly)
band = dataset_DOM.GetRasterBand(1)
arr_DOM = band.ReadAsArray()
arr_DOM = arr_DOM[18:574, 0:650]

dataset_DTM = gdal.Open('rena/rena1/dtm/DTM_32-1-520-164-61_new.tif', gdal.
    GA_ReadOnly)
band = dataset_DTM.GetRasterBand(1)
arr_DTM = band.ReadAsArray()
arr_DTM = arr_DTM[18:574, 0:650]

diff = arr_DTM-arr_DOM
test_sobel = scipy.ndimage.sobel(arr_DTM)


#Segmentation
col = len(diff[:,0])
row = len(diff[0,:])
env = np.ones((col,row), dtype=np.uint8)
env[diff<-3] = 0
plt.imshow(env, cmap = "gray", vmin = 0, vmax = 1)
pyplot.imsave("rena/segmented.png", env)
```

Listing B.1: Creating environment of Rena

```python
import cv2

#Threshhold
im_gray = cv2.imread("rena/segmented.png", cv2.IMREAD_GRAYSCALE)

thresh = 127
im_bw = cv2.threshold(im_gray, thresh, 255, cv2.THRESH_BINARY)[1]
cv2.imwrite('rena/blackwhite.png', im_bw)
```

Listing B.2: Creating environment of Rena part 2

B.2 Hardangervidda

```python
from osgeo import gdal
import numpy as np
from scipy import ndimage, misc
import scipy


## Changing resolution - original res is 0.25
gdal.Warp('vidda/1/vidda/data/dom/DOM_32-1-487-134-31_new.tif', 'vidda/1/vidda/
    data/dom/DOM_32-1-487-134-31.tif', xRes=1.05, yRes=1.05)
gdal.Warp('vidda/1/vidda/data/dtm/DTM_32-1-487-134-31_new.tif', 'vidda/1/vidda/
    data/dtm/DTM_32-1-487-134-31.tif', xRes=1.05, yRes=1.05)


dataset_DOM = gdal.Open('vidda/1/vidda/data/dom/DOM_32-1-487-134-31_new.tif',
    gdal.GA_ReadOnly)
band = dataset_DOM.GetRasterBand(1)
arr_DOM = band.ReadAsArray()
arr_DOM = arr_DOM[18:574, 0:650]


dataset_DTM = gdal.Open('vidda/1/vidda/data/dtm/DTM_32-1-487-134-31_new.tif',
    gdal.GA_ReadOnly)
band = dataset_DTM.GetRasterBand(1)
arr_DTM = band.ReadAsArray()
arr_DTM = arr_DTM[18:574, 0:650]


diff = arr_DTM-arr_DOM
plt.imshow(diff)

test_sobel = scipy.ndimage.sobel(arr_DTM)
plt.imshow(test_sobel)


#Segmentation
col = len(diff[:,0])
row = len(diff[0,:])
env = np.ones((col,row), dtype=np.uint8)
env[test_sobel<-3] = 0
plt.imshow(env, cmap = "gray", vmin = 0, vmax = 1)
pyplot.imsave("vidda/seg.png", env)
```

Listing B.3: Map creation of Hardangervidda

B.3 Calculating incline for analysis

```python
def inc(self, corr, corr2):
    '''
    Gather terrain information of DTM.
    Input: A set of corrdinates (x,y)
    Output: Elevated terrain information
    '''
    x1,y1 = corr
    x2,y2 = corr2
    dist = distance(x1, x2, y1, y2)
    step = dist/20
    incline = []
    for i in range(0,21):
        u = i/20
        x = x2*u + x1*(1-u)
        y = y2*u + y1*(1-u)

        a = np.arctan2(self.img_sobel[math.floor(y)][(math.floor(x))], step)
        incline.append(np.rad2deg(a))
    return incline
```

Listing B.4: Function for incline

Appendix C

RRT code

The complete code can be viewed in GitHub repository [35].

C.1 Initialization of RRT

```python
class RRTGraph:
    def __init__(self, start, goal, MapDimensions, map_filename):
        (x,y) = start
        self.start = start
        self.goal = goal
        self.goalFlag = False
        self.maph, self.mapw = MapDimensions

        #Storing the tree
        self.x = []           #X and Y coordinates of every nodes
        self.y = []
        self.parent = []      #Storing the parents of those nodes

        #Initialize the tree
        self.x.append(x)      #Append start node x coordinate
        self.y.append(y)      #Append end node y coordinate
        self.parent.append(0)

        #Load map
        img = cv2.imread(map_filename)
        self.env_list = img.tolist()

        #Path
        self.goalstate = None
        self.path = []
```

Listing C.1: Initialization of RRT

C.2 Collision routine - RRT

```python
def isFree(self):
    '''
    Collision-test; is the new sample located in free space
    '''
    n = self.number_of_nodes()-1    #get the last node (new sample)
    (x,y) = (self.x[n], self.y[n])
    env = self.env_list              #obstacle map

    if (env[y][x])[0] != 255: #if collision, remove node, return false
        self.remove_node(n)
        return False

    if (env[y][x])[0] == 255: #if no collision
        return True

def crossObstacle(self,x1,x2,y1,y2):
    '''
    Does this edge cross an obstacle?
    Testing if two coordinates are crossing any obstacle using interpolation
    '''
    env = self.env_list

    for i in range(0,101):
        u = i/100
        x = x1*u + x2*(1-u)
        y = y1*u + y2*(1-u)

        if (env[math.floor(y)][(math.floor(x))])[0] != 255: #if collision
            return True
    return False
```

Listing C.2: Collision routine in RRT

C.3 Finding path to goal - RRT

```
def path_to_goal(self):
    if self.goalFlag:                          #If a path is found
        self.path = []                         #Empty list to hold path nodes
        self.path.append(self.goalstate)       #Append goal as the first element
    on the list
        newpos = self.parent[self.goalstate] #Append its parents to the list
        while (newpos !=0):                    #Append parents until it hits the
    start node
            self.path.append(newpos)
            newpos = self.parent[newpos]
        self.path.append(0)                    #Append start node
    return self.goalFlag
```

Listing C.3: Finding a solution path in RRT

C.4 RRT algorithm

```
def bias(self, ngoal):
    n = self.number_of_nodes()
    self.add_node(n, ngoal[0], ngoal[1])
    nnear = self.nearest(n)
    self.step(nnear, n)
    self.connect(nnear, n)
    return self.x, self.y, self.parent


def expand(self):
    n = self.number_of_nodes()
    x,y = self.sample_env()           #sample the environment
    self.add_node(n,x,y)              #includes the function crossObstacle(x1,x2
,y1,y2)
    if self.isFree():
        xnearest=self.nearest(n)
        self.step(xnearest, n)
        self.connect(xnearest,n)
    return self.x, self.y, self.parent
```
Listing C.4: RRT algorithm