TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Computer Science

Timo Loomets 211539IAPM

MOTION AND PATH PLANNING FOR A DIFFERENTIAL DRIVE ROBOT IN MAPPED SIDEWALK ENVIRONMENTS USING A CUSTOM TREE SEARCH TO FIND SMOOTH, SAFE AND ECONOMIC TRAJECTORIES

Master's thesis

Supervisors
Thomas Schildhauer
MSc
Asko Ristolainen
PhD

Tallinn 2023

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatika

Timo Loomets 211539IAPM

LIIKUMISE JA TEEKONNA PLANEERIMINE DIFFERENTSIAAL KONTROLLIGA ROBOTILE KAARDISTATUD KÕNNITEE KESKKONNAS KASUTADES PUUOTSINGUT, ET LEIDA SUJUVAT, OHUTUT JA SÄÄSTLIKKU TRAJEKTOORI

magistritöö

Juhendaja
Thomas Schildhauer
MSc
Asko Ristolainen
PhD

Tallinn 2023

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Timo Loomets

05.05.2023

Abstract

MOTION AND PATH PLANNING FOR A DIFFERENTIAL DRIVE ROBOT IN MAPPED SIDEWALK ENVIRONMENTS USING A CUSTOM TREE SEARCH TO FIND SMOOTH, SAFE AND ECONOMIC TRAJECTORIES

The purpose of this thesis is to analyze existing ideas in motion and path planning and develop a solution for online planning that can be run locally on a robot. Several requirements such as speed, smoothness and safety of the path by the proposed solution were validated from a pedestrian's and the robot's perspective. The thesis was written in collaboration with Starship Technologies.

The thesis contains a comparison of the current solution used by Starship Technologies and the proposed solution based on various simulated test cases. CPU load is also measured and compared on a real robot for realistic results. The complexity and capabilities of the proposed solution are also compared to A*, RRT and costmap solutions.

The thesis shows how the proposed solution has advantages in terms of time and space complexity. It also notes that the proposed solution has exponential complexity related to the explored distance which makes it acceptable for local path planning but suboptimal for global path planning.

This thesis is written in English and is 44 pages long, including 5 chapters, 10 figures, and 2 tables.

Annotatsioon

Selle lõputöö eesmärk on analüüsida olemasolevaid ideid teekonnaplaneerimisel ja töötada välja reaalajas planeerimise lahendus, mida saab robotil lokaalselt kasutada. See lahendus peab arvestama mitme kriteeriumiga, nagu kiirus, sujuvus ja teekonna ohutus. Need on vajalikud, sest robotid peavad keerulises kõnniteekeskkonnas liikudes inimestega läbi saama. Töö on kirjutatud koostöös firmaga Starship Technologies.

Lõputöö sisaldab võrdlust Starship Technologies poolt praegu kasutatava lahendusega ja arendatud lahenduse käitumise võrdlust. Need võrdlused sooritatakse simuleeritud testidel. Reaalsete tulemuste saamiseks mõõdetakse ja võrreldakse ka protsessori koormust päris robotil. Ka pakutud lahenduse keerukust ja võimalusi võrreldakse A*, RRT ja costmap lahendustega.

Lõputöö näitab, kuidas pakutud lahendusel on eelised aja ja ruumi keerukuses. Samuti kirjeldatakse välja pakutud lahenduse eksponentsiaalset keerukust teekonna pikkusega, mistõttu on see kohaliku tee planeerimise jaoks vastuvõetav, kuid globaalse tee planeerimise jaoks ebaoptimaalne.

Lõputöö on kirjutatud ingliskeeles ning sisaldab teksti 44 leheküljel, 5 peatükki, 10 joonist, 2 tabelit.

Abbreviations and terms

A*	*A-star*
CPU	*Central Processing Unit*
MPC	*Model Predictive Controller*
RRT	*Rapidly exploring Random Tree*

Contents

List of Figures

List of Tables

1. Introduction

Several companies in the world aim to provide last-mile transportation services using robots that drive on sidewalks. Some of the most well known among them are: Amazon[1], Kiwibot[2], Clevon[3] and Starship[4] (figure 1). Since the profit margins in the last mile delivery business are small, it is important to reduce costs where possible. Also, robots must get along with people while navigating the complex sidewalk environment. Therefore, robots must move quickly and economically on the sidewalk without disturbing or endangering fellow pedestrians.

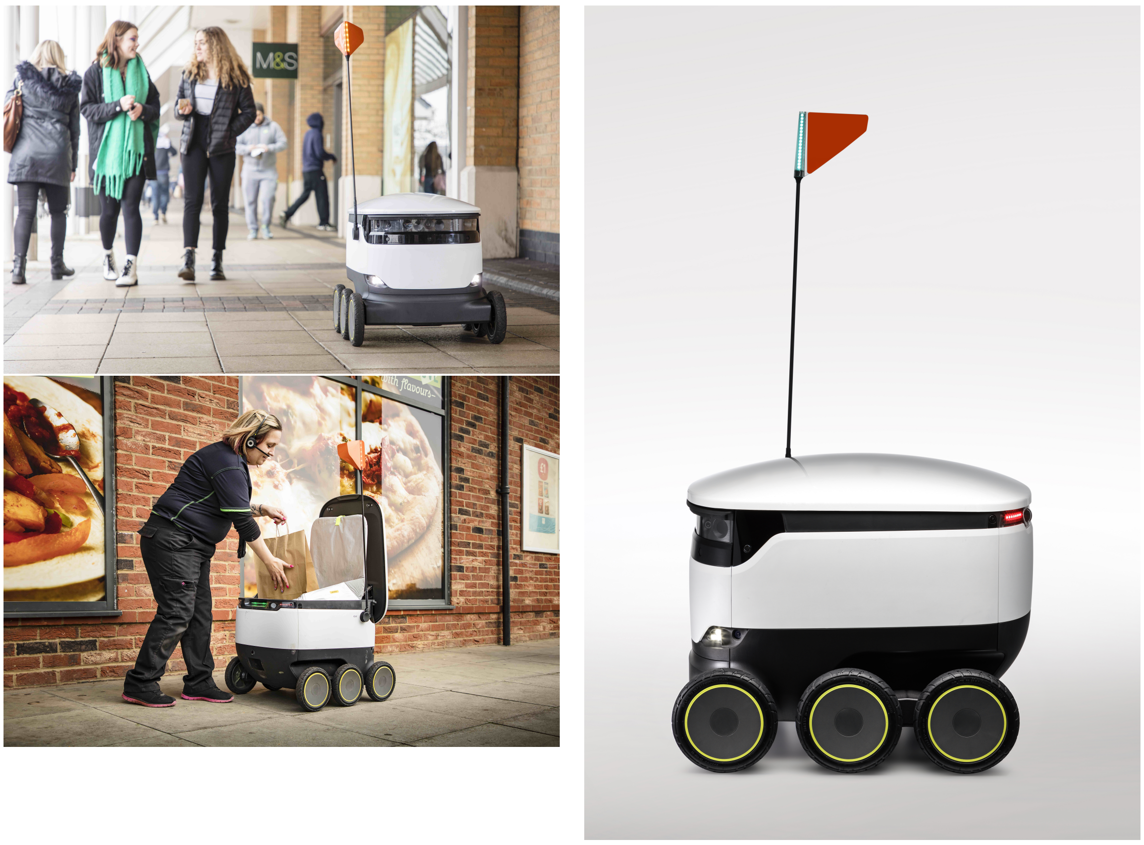


Figure 1. Self driving delivery robots from Starship[4] (from Starship Technologies internal resources)

To solve these problems a key component is path planning which takes into account all these different factors. This falls into the category of local path planning which creates a path starting from the robot's current pose and should be an actionable sequence not an idealized version. This thesis was created in collaboration with Starship Technologies and therefore designed based on their current platform (figure 1). This sets additional limits on the solution as it needs to work on a differential drive robot. These requirements set limitations and challenges that not all existing solutions can meet.

The goal of the thesis is to find a path generation and a path evaluation algorithm with good runtime complexity ($O(length)$ for path evaluation). Path generation should be versatile and adaptive enough to be able to solve situations with complicated sidewalk geometry. Path evaluation should be able to take into consideration speed, safety and smoothness of the path. The proposed solution being able to solve situations that previously required remote help is desirable as that is always more cost-effective.

To achieve this goal a heuristic search algorithm with tree structure and mathematical model-based movement predictions between nodes is proposed. The proposed solution's performance is implemented and compared to Starship's current solution. The CPU load of the proposed solution is benchmarked by running on a physical robot in the background while driving outdoors. The theoretical limitations and capabilities of the proposed solution are also compared to some of the better known path planning algorithms.

2. Background

2.1 Path planning classification

To better understand decisions that need to be made when choosing or creating a path planning algorithm it is useful to know some ways these types of algorithms can be classified. Different classification methods are usually centered around different challenges when describing path planning methods.

One way to classify algorithms is based on the path representations. Usually, a path describes a sequence of poses that can either be represented as nodes or as continuous functions. Most often continuous functions are in the form of splines and made of smaller continuous functions [5]. This allows for a more accurate description of a pose at any point along the path. The use of nodes allows for a simpler computation as a discrete sequence is easier to iterate upon. This approach often uses linear or circular arc shaped connections between the nodes. Some solutions such as Model Predictive Control (MPC) store the path as sequences of poses where every state has only one previous and one following state [6]. The alternative to that is a tree-like storage used in Rapidly exploring Random Tree (RRT) and A-star (A*). Tree-like storage takes advantage of the fact that the early parts of different paths may overlap to reduce the computation and storage cost [7] [8].

A different way of classification is by the planning step of the algorithm. While the path representations are similar to each other, the planning strategies can be very different based on their use cases. For example, genetic algorithms improve on a random set of paths by combining the best parts and slowly mutating the generated paths towards an optimal solution [9]. This requires a fixed-length path as the parts of the path are more interchangeable. Another common type of planning strategies are the heuristic approaches, similar to A*, that use generalizations about the searchable world to guide the search [10]. Different from both of these methods is the costmap approach which describes the cost of all the areas in the world and then finds a path as a connected sequence in that world.

A third distinguishing characteristic is the time at which the solution is available. For example, A* can be considered an offline solution as it requires to be run to completion to get the result, but provides an optimal path once it finishes [10]. Solutions such as genetic algorithms or heuristic searches can be considered any-time as they always have a solution but given more time will generally improve upon it.

In the context of this thesis the real-time solutions that can provide a path within a given short amount of time that could be used on an autonomous robot are most relevant. This means it either has to have a very short search time to completion or be an any-time solution.

2.2 Challenges and limitations

Before choosing or developing a solution it is important to first understand the challenges that need to be overcome. Challenges can be classified by their origin as internal and external challenges. The former is something that comes from the agent that is trying to achieve the goal. In the case of this thesis that would be the differential drive robot navigating the given space. The latter are the environment and other agents. The external category can be further divided into static and dynamic members. Something that is expected to remain constant throughout the execution of the task may be considered static. Things that change in a way that may require different behaviour from the agent would be considered dynamic. For example, road features can be expected to stay in the same location and form over the course of the task and are therefore static members. On the other hand, a pedestrian that may or may not move is a dynamic member of the environment.

2.2.1 Environmental challenges

The most important feature of the environment is the surface on which the robot navigates. It has the important property of texture and may have special features or shapes. The main features of the texture are friction and firmness. These affect the linear and angular acceleration as well as the maximum speed of the robot. It should also be noted that a more slippery surface will decrease the capabilities for braking and pose a safety risk. In the case of a sidewalk we can consider a couple of different surfaces: asphalt, sidewalk tiles, gravel and grass. The first three are common materials for roads while the last can often be found on the side of sidewalks. Both asphalt and sidewalk tiles have good friction while being smooth enough to move on with minimal unevenness. Gravel has more unevenness and can also have less predictable surface friction. Grass can be more slippery and softer making turning harder. This makes roads made of asphalt and tiles more desirable than gravel or grass while all are still viable surfaces.

There are also features on the navigable surface that affect performance and behaviour.

These can be purposeful such as slopes and curbs or coincidental such as holes or puddles. Based on the size of these features they may be considered: too small to affect behaviour and insignificant for navigation, small and affecting performance, medium and requiring special behaviour to be navigable, big and unnavigable even with specialized behaviour.

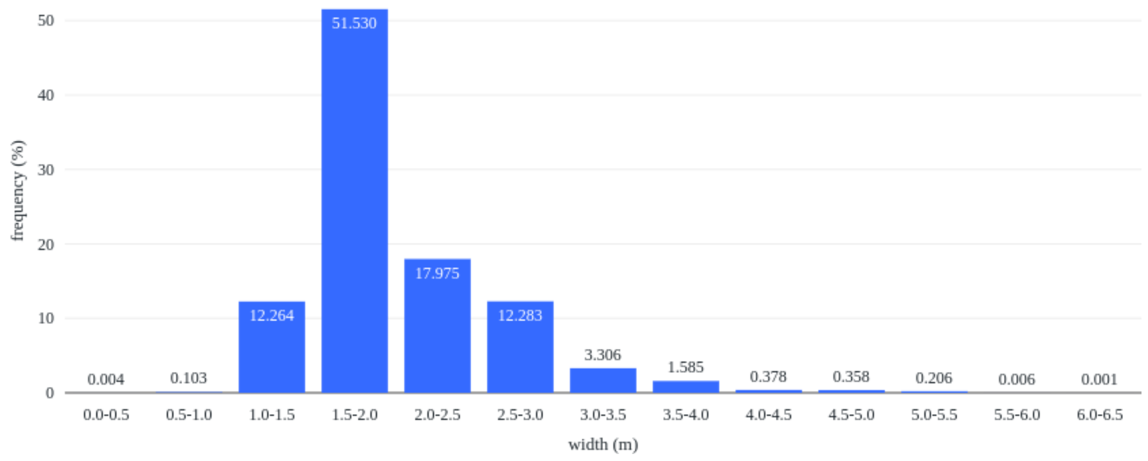


Figure 2. Sidewalk width frequency based on the map of Tallinn by Starship Technologies.

The surfaces describeda above form paths to be travelled to achieve a goal. Sidewalks can usually be represented as shapes of long stretches with more length than width. Most sidewalks in maps used by Starship Technologies are between 1.5 and 2 meters wide (Figure 2). With 94% of sidewalks being between 1 and 3 meters wide this should be considered the common case. There are also outliers with widths less than a meter wide or more than 3 meters wide that should be taken into account. On the side of a sidewalk robots may encounter a car road, greenery or obstacles. These may be same or different height. There are also different regulations about where a robot is allowed to drive.

These features affect the navigability of a surface. With these effects on navigability we can divide the areas into the following categories: well navigable, navigable with difficulties, and unnavigable.

There are also different objects in the environment. These objects may be considered obstacles when it comes to the robot's movement as they hinder or stop both movement and perception. Besides the movement of the obstacles there are also properties of permeability and consequences of making contact with the obstacle. An obstacle is considered permeable if it allows the robot to drive through. Permeable obstacles affect perception but not movement. One example of such obstacles is foliage. There can also be different consequences of making contact with obstacles. These consequences may be either negative or neutral.

2.2.2 Platform limitations

It is also important to consider the physical platform of the agent that is performing the actions in this environment. Platform dictates which actions can be taken and what kind of information is available. The former of those affects the action space while the latter affects the evaluation criteria.

Depending on the kinematics of the robot it can have different non-holonomic constraints. These are constraints that depend on the path taken and not just the current position. These limitations on the robot's movement constrain the space of future poses. Most common forms are often described as bicycle and unicycle models [11]. The main difference between the different kinematics models is that unicycle can turn in place while bicycle model requires linear movement to turn. Neither is able to move side to side without turning. The robot used in this thesis uses unicycle model.

Another limiting factor is the sensing capabilities of the platform. In the case of Starship robots this comes from more sensors being directed in the forward direction. Therefore any movement made in reverse is more dangerous. There can also be limitations on speed from the rate and delay of the sensors. Driving at higher speeds is more dangerous due to sensors being unable to warn of dangers in time. Another factor to be considered is the precision of the sensors. The precision of the sensors detecting obstacles and the robot's location defines the closest distance to obstacles and sidewalk edge.

Any algorithm is also limited by the hardware that it is running on. This defines the computation speed, parallel processing capabilities as well as amount of available memory. There is also a related limitation from battery as doinf more processing requires more energy and therefore drains the battery faster. Parallelization of an algorithm should be considered as some processors have better single core performance while others excell at running multithreaded programs. Some algorithms also allow dynamic scaling of computation load to reduce battery drain when necessary.

2.3 Relevant algorithms

2.3.1 A*

A* is one of the most well known path planning algorithms that is often used as a benchmark or inspiration [8]. Its most notable feature is combining the already existing Dijkstra algorithm [12] with an additional heuristic function to speed up the search. This is a way of adding information about the search space's intrinsic properties. This allows searching in ways that incorporate human intuition and reduce unnecessary exploration.

The main requirement for using A* is for the searchable space to be in the form of a graph. The search is performed around a central point which is usually either the start or the goal. The heuristic function tries to estimate the summed cost to nodes not yet explored. With a perfect heuristic function the search can find the best path in $O(N)$ time. For A* to work correctly the heuristic function must never overestimate the cost. This guarantees that the search can be stopped when all predicted costs are higher than the already found path's cost.

2.3.2 RRT and RRT*

Rapidly exploring Random Tree (RRT) is a tree-based search algorithm [7]. It uses a node-based random tree generation to find paths from the origin point. This search is performed in continuous space and is capable of traversing around obstacles through a non-linear path. It searches the space by creating nodes in random directions connected to the already existing tree. The nodes are created randomly and connected to form a tree structure. It is also possible to use bias towards the goal to guide the direction of generated vertices. Once a generated node is sufficiently close to the goal the path to that node is considered the solution.

There exists also a well known improvement of RRT known as RRT* [13]. This is an advanced version of RRT that improves the found solution by reconnecting nodes when a new node is created. Reconnecting nodes optimizes the path, making the final path shorter. Having this improvement greatly enhances the usefulness of the found path at the cost of increased computational complexity.

2.3.3 Model Predictive Controller

Model Predictive Controller (MPC) is a mathematical model based control solution [14]. It is designed to make control decisions to optimize a limited time horizon. It stands out in its versatility for optimization targets as well as flexibility for being applied to any system that can be mathematically simulated. The weakness of the model predictive controller lies in the limitations on simulation. As the complexity of the system increases more advanced solutions are required for optimizing the time horizon.

Since MPC is a controller and not a path planner then for this thesis the path predicted by a model predictive controller is considered the solution path. This means MPC is not used as just a controller for selecting the next action but as a method of generating a proposed path that consists of multiple actions.

2.3.4 Layered Costmap

Costmaps are a way of describing space traversal cost in a way that focuses on describing the environment first and then finding solutions using that description [15]. Most often this is achieved by dividing the space into cells that then get evaluated. This process has been further optimized by dividing the costmap into layers that can be edited separately. Allowing for updating the costmap in areas that changed and not regenerating the entire map. The main limitation of costmaps is the granularity and complexity that comes from describing large areas of space and keeping it in memory.

2.3.5 Suitability of public algorithms

A*, RRT, RRT* and costmap are able to provide a path consisting of poses. These however lack the motion dimensions of velocity and acceleration. Since a system capable of considering velocity and acceleration limitations is required these solutions were rejected. MPC produces a result according to movement constraints, but it has a limited horizon with limited capabilities at further future.

The robot driving system requires path planning, simulating the driveability of the path and evaluating the path. A*, RRT, RRT* and costmap only do the path planning and evaluation. MPC does simulation and evaluation. Therefore these require additional systems that would perform the missing task. A custom solution that can perform both planning and simulation simultaniously is optimal as it doesn't require an additional system. There is also the requirement to integrate the solution with the existing system used by Starship Technologies. For this a solution capable of coming up with a path as well as producing movement speeds is needed. Therefore a custom solution is considered necessary.

3. Development

The development process for the proposed solution can be divided into three parts. The first part is the analysis of requirements that consists of logical reasoning defining the limitations. The second part is the core functionality development that involves the creation of the main process flow and its required components. This is done while keeping in mind the limitations set by the previous step. The third part is the improvement of core functionalities to enhance the performance and improve the results.

The proposed solution itself consists of cost functions, generation processes and simulation processes. Cost functions and simulation processes are used internally by the generation process.

3.1 Analysis of requirements

A key element in the proposed solution is the cost function which is composed of sub-cost functions. This can be compared to the distance function in RRT* or A* as it defines which solutions are preferred. Since it contains much more than just the distance it is called a cost function.

Cost function has two main properties: robustness and sensitivity. Robustness describes how well it describes the actual desired behaviour. Sensitivity describes how responsive it is to small changes. Both of these must be considered when designing a cost function to describe a desired property. For example, we can look at a cost function for the property of driving to reaching the goal in as short time as possible.

The maximum robustness would be just the value of time spent to reach the goal. This however means that any point before reaching the goal would be without a value since these have not yet reached a goal and the final time is unknown. To increase the sensitivity we can look at the current speed instead of total time. This assumes that a faster speed reduces the total time to the goal on average. However such approach reduces the robustness as in situations which require sharp turns high speed might prove counterproductive.

There are also important functional and mathematical properties for the cost function.

Firstly the cost function should not depend on the number of segments the path is divided into. That is necessary so that reducing the time step size would add extra precision without having undesirable effects on exploration choices. Thherefore any cost should be calculated along the path before the node. To achieve this the cost function for a single point in path needs to be integrated over the path between nodes. The cost functions also need to be summable over path in a way that the sizes of these segments do not change the final sum. This means that applying any non-linear function to the cost has to happen before the integration.

For path generation the driving constraint is limitation on computation. This limits the tree-based path planning solutions on the number of nodes that can be created. With the computational and memory complexity of creating a tree proportional to the number of nodes in the tree. The number of nodes in a tree can be expressed in the form of $O(b^d)$ where b is the average branching factor and d is the depth. The branching factor is the average number of child nodes that a node has. For a desired prediction future around 1 to 5 seconds and time step size between 0.01 and 0.25 seconds, the needed depth is 4 to 500 layers. Depending on the rate of running and time cost of a single node generation the number of maximum nodes may vary. A reasonable frequency for a driving robot is usually between 10 and 50 Hz. This gives 20 to 100 ms for creating the nodes. The exact time for creating a single node depends a lot on the implementation as well as the platform the code is running on. Assuming desire of creating 1000 to 5000 nodes then the target average node creation time is 20 μs.

Given the range for depth value as well as maximum node count we can estimate the branching factor range using the formula

$$N = b^d \tag{3.1}$$

where N is the number of nodes, b is the branching factor and d is the depth of the tree. From this we can algebraically derive the function

$$b = e^{ln(N)/d} \tag{3.2}$$

where e is Euler's number and ln is natural logarithm. Using this equation 3.2 we can get the approximate range of branching factor

$$e^{ln(1000)/500} \approx 1.014 \tag{3.3}$$

$$e^{ln(5000)/4} \approx 8.409 \tag{3.4}$$

from 1.014 to 8.409. A branching factor of 1 would mean that the tree is a sequence of nodes without any alternative choices. Therefore a branching factor higher than that is preferred to guarantee exploration of different options. Using a branching factor below the range will result in more depth at the cost of exploring lower depth options less. Branching factor above the range will result in better exploration of earlier steps, but will lack the desired depth.

3.2 Development process

The creation process can be divided into several periods. The first of these was focused on developing the cost functions. This was necessary to get the understanding of what are the necessary criteria and resources for choosing a good path. Using information from this the initial structure was created in the second development period. This consisted of creating initial structures for data storage as well as the core generation loop. Implementation of generation with the initial evaluation using the previously created cost functions allowed to start creating paths. The third period consisted of iterative improvements of these processes as well as adding new features beyond the minimum viable product.

3.2.1 Cost function development

Based on the idea of dividing cost into economic, predictability and safety we can come up with ideas what to measure for the cost. Starting with economy it is fairly logical to assume that shorter distances are better. In addition to this we can assume that the longer it takes for the robot to reach it's goal the higher the opportunity cost from having the robot occupied. These two metrics are in different units and therefore need to be converted to a common unit to be comparable. For the proposed solution the unit of currency in the form of USD was chosen. This was chosen for ease of use for economy calculations, but there is no other clear benefit from this.

Coming back to the question of creating the cost functions for distance we can start by estimating the wear of the robot from driving. With a constant describing robot cost and distance it takes for parts to break down, it is possible to describe the distance cost as follows:

$$C_d = s * c_r / s_r \tag{3.5}$$

where C_d is the cost of distance, s is the distance travelled in meters, c_r is the cost of a repair in USD and s_r is the distance between repairs in meters.

To describe the cost of time in dollars over a given path it is necessary to first get the time. It is useful to calculate this through distance and speed since that gives a function for evaluating cost from speed which is directly controlled by the chosen commands. To do this time can be derived from the formula for speed $v = s/t$ as such $t = s/v$ where v is speed, s is distance and t is time. Using inverse of speed which is also known as pace we can write the function as a multiplication:

$$C_t = s * p * c_o \tag{3.6}$$

where C_t is cost of time, s is distance in meters, p is pace in seconds per meter and c_o is the cost of robot being occupied in USD per second.

It is important to note that since pace is $p = 1/v$ then in cases where robot is standing still pace starts approaching infinity. To avoid infinite cost in those cases it is good to limit the maximum value of pace. One way to apply a maximum limit to the value while keeping the function mostly linear, continuous and differentiable is to use a reversed softplus (a function derived algebraically from softplus [16]) function:

$$y = l - ln(1 + e^{a-bx}) \tag{3.7}$$

$$a = ln(e^l - 1) \tag{3.8}$$

$$b = \frac{2}{l}(a - ln(e^{\frac{l}{2}} - 1)) \tag{3.9}$$

where l is the desired limit, x is the input value and y is the limited value.

For the predictablity the cost needs to capture actions that make the robot's actions hard to predict for pedestrians. First thing that can be captured in this is the following of a road rule which shows how well the robot stays on the correct side of the sidewalk. To measure the undesirability of a path we can measure the distance to the road rule. For integration of this there are two main candidates: distance and time. The better option here is integrating over time, since that gives advantage to paths with faster velocity. For simpler integration between nodes we can use the assumption that the distance to the road rule changes linearly. It is also important to use unsigned integral to avoid the danger of a negative integral resulting in smaller than correct cost. Therefore the cost function can be written as follows:

$$C_{rr} = c_{rr} * \int_0^{\Delta t} |D(t)|\, dt = c_{rr} * \frac{D(\Delta t) * |D(\Delta t)| - D(0) * |D(0)|}{2 * (D(\Delta t) - D(0))} * \Delta t \tag{3.10}$$

where C_{rr} is the cost of being away from road rule, c_{rr} is the constant multiplier for cost to road rule in USD per meters times seconds, Δt is the time step in seconds and D is the function for distance to road rule in meters at the given time.

Another predictability metric that was investigated was acceleration and jerk of the robot. For a differential drive robot this can be measured in two dimension, one in longitude and the other in angular. Due to the difference in range and volatility of these values it is useful to pass the values through a function that cuts off low level noise while limiting the values above a threshold. A useful function for this is the sigmoid function [17]:

$$y = \frac{1}{1 + e^{a-b|x|}} \tag{3.11}$$

where a and b are the tuning values for the slope and place of the threshold. It is also useful to pass the input through an absolute so that both negative and positive values give the same cost.

For safety cost we want to capture actions that highly undesirable due to danger. For this purpose the event of driving outside of mapped sidewalks was chosen. Since the event itself does not occur, it is necessary to find a way to estimate the danger of driving off sidewalks. This estimate can be based on the predicted time until driving off sidewalk. The time itself can be predicted from the speed and distance to the sidewalk edge. To calculate the distance to the sidewalk edge it is useful to use a circular arc movement assumption as that takes into account possibility of turning before reaching the edge. The radius of the arc can be calculated from the current linear and angular velocity of the robot using the following formula:

$$r = v/\omega \tag{3.12}$$

where r is the radius of the arc in meters, v is the linear velocity of the robot in meters per second and ω is the angular velocity of the robot in radians per second. Using the arc distance to intersection with sidewalk edge as the distance the time can be predicted as follows:

$$t = s/v \tag{3.13}$$

where t is the predicted time in seconds, s is the calculated arc distance in meters and v is the linear velocity of the robot in meters per second. The predicted time needs to be then converted to estimated probability of driving outside of the sidewalk. A greater time means that the robot has more time to react therefore the time needs to be passed through a decreasing function. Since the probability of an event at a given moment is always between 0 and 1 the outputted value needs to be normalized to that range. It is also useful to have the rate of output be non-linear as the chance of being able to react becomes insignificantly small at small time values while big time values mean future without any certainty of prediction and therefore insignificant danger. For the proposed solution a sigmoid function with the input multiplied by -1 was chosen as it fulfills all the previously mentioned criteria. Therefore the probability of driving outside of sidewalk can be represented as such:

$$p = \frac{1}{1 + e^{a(t)-b}} \tag{3.14}$$

$$a = ln((\frac{0.99}{0.01})^2)/t_{p1} \tag{3.15}$$

$$b = -ln(\frac{0.01}{0.99}) \tag{3.16}$$

where p is the probability of driving outside of sidewalk, e is the Euler's number, $ln()$ is the natural logarithm, t_{p1} is the time at which the probability is estimated to be 1 percent and t is predicted time until driving outside of sidewalks in seconds. For the ease of integration assumption of close to linear change between probabilities at small time steps was used. That allowed calculating the probability at discrete points in time and estimating the integrated probability by linear function integration between these values. This resulted with the following cost function:

$$C_o = c_o * \frac{p_0 + p_1}{2} * t \tag{3.17}$$

where C_o is the cost of danger of driving outside of sidewalk in USD, c_o is the cost of a single event of driving outside of sidewalk in USD per seconds, p_0 is the probability of driving outside of sidewalk at a previous moment, p_1 is the probability of driving outside of sidewalk at the current moment and t is the time between the moments in seconds.

To create and tune the cost functions it is useful to have driven paths to evaluate. The easiest way to get those is through simulation and recording the driving. From the previously mentioned cost functions (3.5), (3.6), (3.10), (3.17) we can see that most of the costs can be calculated using the pose and velocity time series. For the simulation a number of test cases are necessary. These should be of situations that have suboptimal behaviour with the current system. For this thesis the following situations were chosen:

1. turn of 90 degrees (Appendix 2),
2. turn over 90 degrees (Appendix 3),
3. short and high angled connection between straight segments (Appendix 4),
4. wide sidewalk to narrow sidewalk transition (Appendix 5),
5. smooth large radius curve (Appendix 6).

The main reason for these cases was the necessity of combining different priorities in these cases. The first case of turning 90 degrees while the road rule is inside the curve poses a challenge due to high angle of the turn. This means the speed of completing such a path has to be balanced with the predictability to avoid going too close to the opposite edge of sidewalk and blocking the corner for all other pedestrians.

The second case of over 90 degrees turn is more speed oriented as the robot needs to change movement direction to the reverse of what it was at start. There is also a significant safety concern with such a case as cutting the corner can lead to ending up outside of mapped sidewalk on the inside of acute-agnled corner.

The third case of a high angled connection between straight sidewalk segments is common in case of sidewalk misalignment and proposes a challenge in predictability and economy. Overreaction to sidewalk heading and road rule change can lead to unintuitive movements for humans as well as loss of speed.

Fourth case of transitioning from wide sidewalk to a narrower one proposes a challenge in all three categories. The predictability criteria needs to account for the change in road rule smoothly. The safety needs to avoid driving outside of sidewalks by accounting for the sudden end of sidewalk along some of the wide segment. The economy cost of the situation requires the robot to be able to drive this segment at reasonable speed and preferably without stops.

In the fifth case a curve with a large radius may seem simple at first, but it challenges the safety and economy of the robot. Throughout this sidewalk feature the robot's heading is consistently close to the sidewalk edge while the economic completion of this is still feasible thanks to continuous turning. Predictability also plays a role in this situation as people would expect the robot to follow path similar in shape to the sidewalk and avoid inconsistent turns. There is also possible variations of this case where the road rule can either be on the inside or the outside of the curve.

3.2.2 Core functionality development

The core functionality of the proposed solution is consists of tree generation and storage and choosing the best path. For basic tree generation using only economic costs is sufficient as it incentivizes the algorithm to move efficiently towards the goal.

The tree has some properties that need to be considered when choosing the storage format. Firstly the number of subsequent nodes may vary significantly and a well generated tree should not be balanced. For creating and pruning the tree it needs connections from root node towards the nodes further in predicted future so that nodes could be easily added or pruned. For constructing a single path from the tree it also needs reverse connections from the leaves towards the root. This implies that the most efficient way to store the tree is in a linked manner where nodes themselves contain references to nodes after them as well as their parent.

The main path generation loop consists of three parts. Firstly the setup that takes the robot's current state and either constructs a new root node from it or finds a node best suited to take the place of a root node from the already existing tree. The construction of a new root just requires copying of input values. To move the root to an already existing node requires a comparison function that finds the node most similar in robot state that is also below some constant threshold. It is useful to apply an optimization through assumption that search beginning from the old root is close to the global optimum due to the change being relatively small compared to the entire tree. From that a distance function for comparing similarities between bot states can be applied. The search can be concluded once the distances start increasing again as the assumption requires us to only find the local minimum. In case of root node movement the branches lying outside of the new root can be discarded as they have become unreachable.

Secondly part adds the given number of nodes to the existing tree. For the core functionality here it is necessary to have a comparison of nodes that allows choosing the one with most potential for low cost. Using a good heuristic function is the main requirement for that. The proposed solution estimates the average recent cost as well as the distance covered by it. From there an estimate of the distance to the goal along sidewalk with a penalty for being angled to the side or backwards on the sidewalk is used to get an estimated distance until goal. Multiplication of average recent cost per distance with the remaining distance to goal is used for heuristic cost estimate. Since the cost until the goal can be disproportionately large compared to the prediction distance it is useful to apply a clamp on the distance.

The third part of the core functionality is choosing the optimal path. This starts with comparing all of the generated nodes to find the best final node. Due to the goal of choosing a good path and not just a good next action this should be comparison of the leaf nodes of the tree. This avoids choosing nodes that have all suboptimal child nodes as it would indicate that in the long run this sequence of actions is not viable. In addition to this it is useful to only account for the calculated cost and not the heuristic cost in this stage since the heuristic is less accurate due to estimation of both cost and distance. After choosing the optimal final state to reach the optimal path can be created by selecting the sequence between the root of the tree and the chosen final node. From this path the actions that were used to predict the robot states can then be used for robot control.

This development created all of the necessary components for the core functionalities of the proposed solution that could then be expanded upon according to observed shortcomings. It was able to complete the main goals of a path planning algorithm by making predictions and evaluations and select a path according to economic criteria.

3.2.3 Refinement and additional features

After the creation of the core functionalities the iterative improvements started. The first observed problem was regarding the exploration and exploitation tradeoff. In the context of this proposed solution exploitation is used to describe the process of choosing to expand the tree from a node with the lowest cost. A complementary feature to exploitation is exploration that describes the process of discovering new ideas that may be outside the optimum in the selection. In other terms exploitation can be described as approaching the local minimum while exploration is about crossing ridges to find areas with different minima.

The combination of real cost and heuristic cost is susceptible to being overly optimistic or pessimistic about predictions. This can lead to search creating shapes of fans which start expanding either from the root or the leaves depending on which category the function currently lands in. Neither of these is optimal, since they are likely to fall into a local minimum and create unnecessary amount of nodes in that minimum instead of exploring more distinct paths. To solve this problem two new exploitation strategies were added. Both of these work by only exploiting a subset of existing nodes.

Firstly the depth based exploitation was added to allow for balanced exploitation. In this strategy each of the existing depths of the tree was exploited separate to each other (figure 3). This guarantees creation of new nodes at each depth which allows for exploitation of nodes that were close to the most optimal node, but might have better options further in the future. The second strategy involved depth first exploitation into a completely new path. This strategy is used from as early node as possible and creates longer separate branches that often lead to discovery of new minima instead of further exploiting already existing ones (figure 3). This adds significant variety to the option pool and can discover paths that require multiple nodes deep exploration before becoming comparable in global optimality. By adding these two strategies the necessary number of nodes could be reduced by an order of magnitude since the solution no longer depended on saturating a minimum.

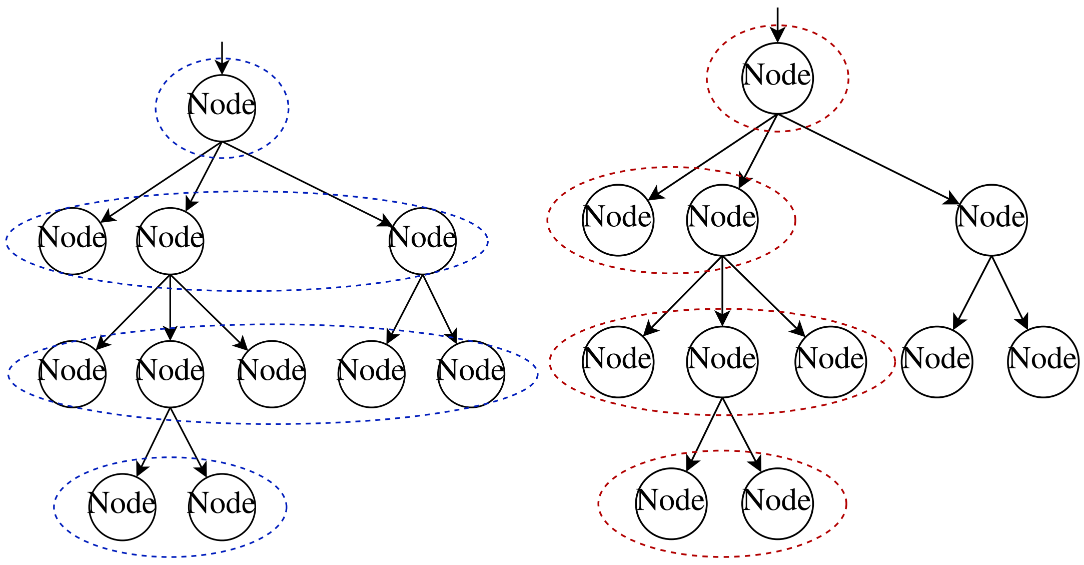


Figure 3. Strategy exploitation selection groups marked with a dashed line. Layer-based exploitation on the left and new path exploitation on the right.

The second challenge to be tackled is the lack of certain cost considerations. First of these is the safety as it has significant effect on avoiding options with high risk. The chosen safety metric is probability of driving outside of sidewalk and it is measured by the predicted time until driving outside. If the time to predicted driving outside of sidewalk is smaller it is less probable that the robot will be able to take the necessary actions to avoid it. Using the integrated time the risk of driven segments is estimated and added to the existing cost. This had the positive effect of requiring the path to either turn or slow down when getting closer to edge. This also affected positively the smoothness of driving in curves as in smooth continuous change of sidewalk edge the cost is also gently guided towards the correct direction.

Second cost to be added is related to predictability and is related to road rule. This is necessary to make the results more comparable to the existing solutions as a road rule can make a path significantly more challenging. This is due to the easiest path to follow often being close to the center of sidewalk while this makes it harder to navigate for other pedestrians. Without road rule cost the optimal path is to stay on the outside after turn while with respect to road rule the robot returns to the side of sidewalk (figure 4). As described previously this is implemented using an integral of time and distance away from the road rule.

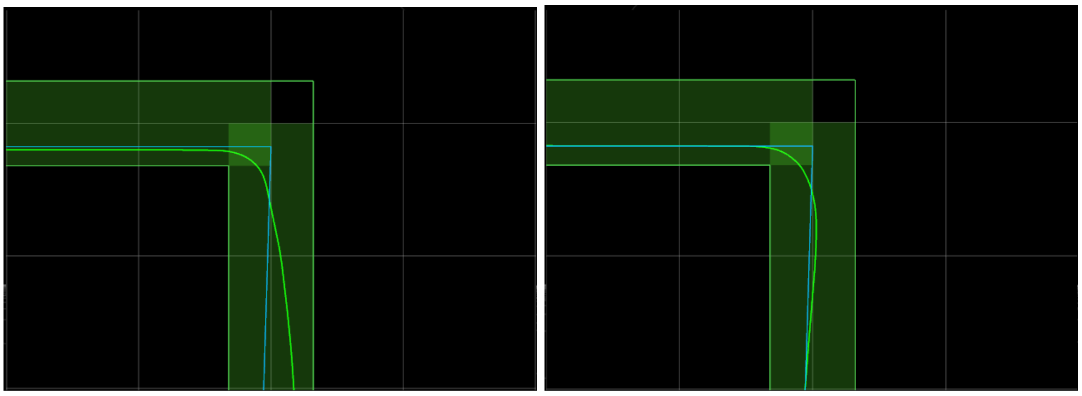


Figure 4. Path (green) and road rule (cyan) without road rule cost (left) and with road rule cost (right).

3.3 Proposed solution

For finding the possible paths a tree search is used (figure 5). This allows reusing steps closer to the root and explore more in further depths. To select the best path a cost function is used where better paths have lower cost.

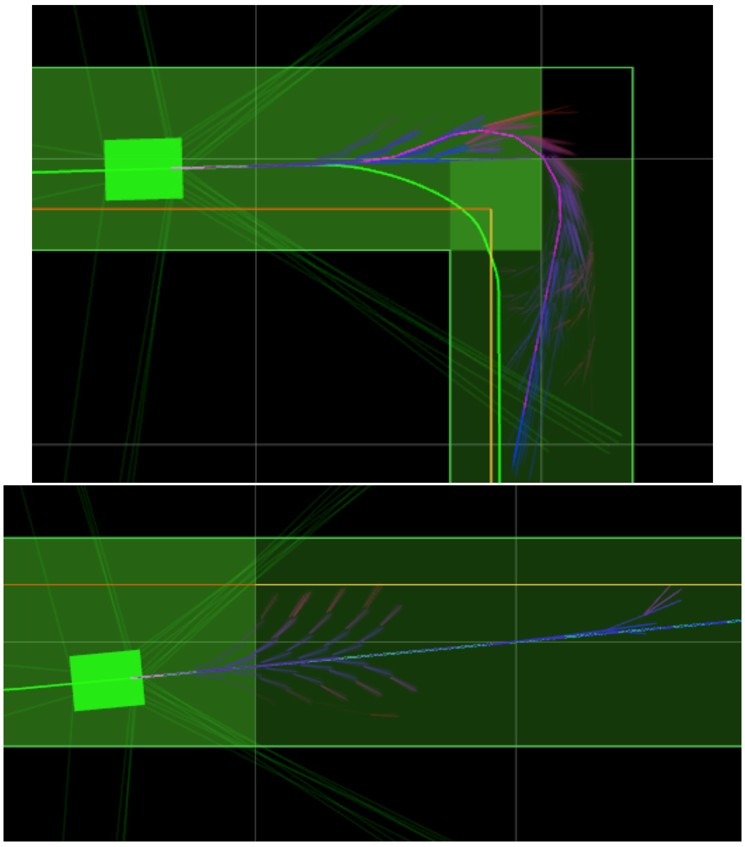


Figure 5. Proposed solution predicted motion trees for 90 degree turn (top) and approaching road rule (bottom). The predicted tree is colored blue to red based on the predicted cost with blue being the minimum cost and red maximum. The best found path is colored magenta. The final taken path is colored green. Road rules are colored yellow.

3.3.1 Path generation

The path generation starts with a root node located in robot's current pose (6). Each node consists of robot's pose and current velocity. From there further nodes are created at discrete time steps based on selected commands. A command consists of desired linear and angular velocity. Using the selected command and the previous state a $O(1)$ complexity computation is possible to find the new state achieved after the given time step.

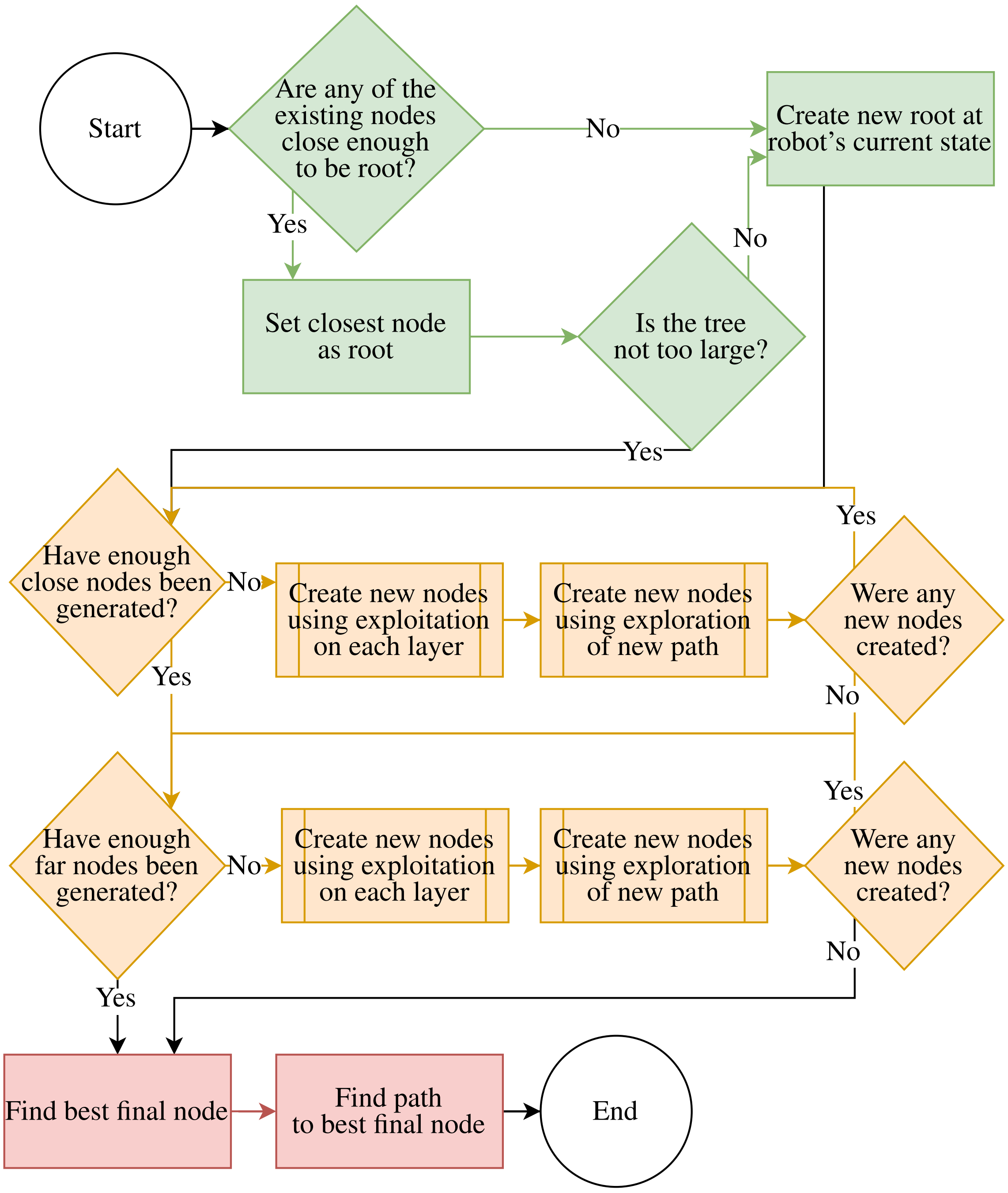

Start
Are any of the existing nodes close enough to be root?
No
Create new root at robot's current state
Yes
No
Set closest node as root
Is the tree not too large?
Yes
Yes
Have enough close nodes been generated?
No
Create new nodes using exploitation on each layer
Create new nodes using exploration of new path
Were any new nodes created?
Yes
No
Yes
Have enough far nodes been generated?
No
Create new nodes using exploitation on each layer
Create new nodes using exploration of new path
Were any new nodes created?
Yes
No
Find best final node
Find path to best final node
End

Figure 6. Tree generation flow with the setup as green, generation as orange and best path selection as red.

There is the challenge of exponential growth in number of nodes as the depth increases. This can be described as $O(b^D)$ where b is the branching factor and D is the depth. If branching factor is left unchecked it can lead to inability to look further away due to exploration close to the root will take up all of the time. With a reduced branching factor the depth can be greater before a limiting amount of node count is reached.

Another factor is that the depth D is present in the exploration distance function of $t = D * \Delta t$. The exploration distance in time dimension is important to make choices that are beneficial in the longer run. This can be affected by increasing either of the two factors of depth D or time step Δt. From this we can also deduce that a larger time step is a relatively easy way to increase deep exploration capabilities. With larger time steps the algorithm starts losing granularity and decisions can become suboptimal if the optimal motion requires fidelity.

To limit the branching factor an future predicting exploration methodology is used. This is a further development of ideas from A* of using a combination of calculated and predicted cost. The summed cost of actual and heuristic cost is used to choose nodes with the best potential for good paths. A* has a strict requirement on the heuristic function to never overshoot the real cost. Since the proposed solution's stopping criteria does not expect to find the global optimum this constraint can be violated. An optimistic heuristic function will result in more of a breadth first search while a more pessimistic heuristic will lead to a more depth heavy search. This effect can be used to tune the search pattern.

The other key part of search algorithm are the different search sub-strategies making up the overall search strategy. These are methods of limiting from which nodes the optimal nodes are chosen. The purpose of these is to solve the problem of exploration versus exploitation. As an example the basic A* method of choosing the node with globally best potential is fully exploitative as it improves the best path found so far towards the local minimum. The problem from a fully exploitative approach is the danger of getting stuck in a local minimum. For A* this is usually acceptable since the graph is discrete and number of nodes is sufficiently limited to allow for saturation of local minimum. With the proposed solution and exponential increase in number of possible nodes saturation within any feasible time limit is not possible. Therefore two additional strategies are proposed to limit the exploitation to a local group of nodes and therefore increase exploration of possible new minima.

Firstly there is the strategy of layer-based exploitation. This goes through the tree from root to a given depth exploiting each layer once. This forces the search to look into depth and stops it from having to saturate earlier layers before getting any information about the further layers. Through this method the minima of early layers can create new minima for later layers that would then be explored further if they are better than the currently existing minima of a later layer. The second strategy is a constrained depth first search. This exploits a node as early in the tree as possible and then continues exploiting the child nodes created layer by layer while ignoring the other nodes at same layer that were created previously. The benefit of this approach is that it allows the search to ignore the existing minima and explore paths that require a certain depth before becoming competitive with other previously created nodes. These two strategies are then used alternately to get the benefit of each of these and create a varied pool of samples to choose from for the final solution.

3.3.2 Cost functions

As mentioned previously the proposed solution depends on a cost function for determining which path is better and which is worse. The proposed solution divides the cost into three components based on type: economy, predictability and safety. The purpose of this categorization is to enable prioritization based on the situation. For example, in dangerous locations or situations the weight of safety cost can be increased to achieve safer paths.

In the proposed solution the category of economy contains two components: pace and distance. This is chosen because it increases as the robot drives more slowly and since the desired behaviour is faster speed then pace and therefore cost increase with slower speed. The faster speed is economically beneficial as it allows the robot to reach more locations in the same time period. The other component of distance is chosen to work in combination with pace to incentivize shorter paths. Longer paths would mean more ware on robot's moving components as they would be doing more work. This however means that these components would have to be replaced and would therefore have a cost. At higher speeds the robot will have to travel greater distances during cornering. Therefore the distance and pace costs will work in opposite directions and choose a compromise between the two. It is also important to note that since the pace includes division by distance then in case of turning in place the result would be infinite pace. Making the cost infinite in a behaviour that might be desirable in certain situations is clearly suboptimal. To solve this we can use a normalizing function that limits the maximum value while preserving the linear nature of the function. In case we want to use any differential dependent approach the function should also be fully differentiable. In this solution an inverted softplus function is used.

The second category of predictability contains costs related to behaviour that would look predictable for other pedestrians. The main reason for separation from economy is the abstract nature of this metric. Predictability can be estimated with varied priority with very different size relative to economy which can be measured and estimated based on statistics. Therefore it allows for better relative tuning of different cost types without requiring other multipliers of economy costs to distance from the real world data based values. Currently this category only contains distance from the road rule. For example, in most of Europe and USA it is common to navigate on the right side of the sidewalk to allow pedestrians from the opposite direction to pass. At the same time in United Kingdom it is customary to navigate on the left side. Therefore it is desired to have cost that would increase with the distance from the desired road rule. As stated in the requirements for the cost functions this should be evaluated along the entire path taken and not just the node. In the proposed solution the distance is integrated along the time. This means the algorithm will choose the solution that approaches the road rule the quickest. This is better than the alternative of integrating along the distance as it will benefit the faster solutions.

The third and final category of safety represents cost from dangers of performing actions that may result in highly undesirable behaviour. For the proposed solution driving outside of the sidewalk is highly dangerous and therefore considered a safety problem. To add a related cost to this the danger needs to be measured and evaluated. For the measurement unit predicted time to driving outside of sidewalk is used. This adds the necessary robustness as the path can be amended by changing either the speed or trajectory. To measure this the given pose and velocity were compared to the mapped edges of sidewalks. From there simple geometric prediction with assumption of constant speed is used to predict the time to driving out of sidewalk at a given node. This time is passed through a reversed sigmoid function to get a cost that decreases as the time increases. Then the cost at the last node and the current node are used with the assumption of linear change between them to integrate it over the time between the nodes. This results in cost to outside of sidewalk integrated over time to estimate the danger.

The cost evaluation is performed for each node with $O(1)$ complexity as all the equations are in form on algebraic expressions without loops or recursion. The cost at a given node is equal to the cost of previous node plus the cost of the segment between the two nodes. Therefore to evaluate any node only the nodes before it have to be evaluated. Since the number of nodes is proportional to time taken which in turn is on average proportional to distance the solution can be considered $O(length)$ in relation to the path length.

4. Result analysis

4.1 Evaluation criteria

For comparison of solutions the metrics of economy and safety were chosen. This choice was made to cover variety of different possible scenarios. These categories also represent different interest groups. Economy represents the business interests regarding the robot which involves optimisation of driving that has significant effects at larger scale or over longer distances. Predictability represents the interests of other pedestrians who's main concern is being able to move alongside the robot. Safety represents the interests of the community and involves the well-being of the environment and people not navigating on the sidewalk but also the owner of the robot as it may result in damages.

For economy evaluation the main affecting factors are pace and distance driven. These are constrained to a predefined range to avoid extreme values according to previously mentioned methods. Since it is an economic consideration it can be calculated in currency, time or some other relevant resource. It is also good to use real world constants to get an accurate prioritization between pace and distance.

For predictability comparison the acceleration and jerk of the robot are used. Linear acceleration peaks describe rapid starts or brakes and are a sign of insufficient planning ahead. Therefore acceleration and braking values closer to zero are preferred. It should be noted that below certain threshold the effects for pedestrians become trivial and therefore the cost is non-linear. The main dangers here arise when pedestrians try to predict robot's future speed and position while navigating close to the robot. The jerk is a derivative of acceleration and describes the indecisiveness of the robot or inconsistent path. Most commonly this value peaks in cases of quick corrections in opposite directions of either speed or turning. These indicate instability of the system and make the robot's next actions harder to predict for a bystander.

For safety comparison the metric of closeness to the sidewalk edge was chosen. This describes the danger of the robot ending up outside of designated driving area and potentially disturbing vehicles on a nearby road. As it is not something that happens in a planned path the danger is estimated based on robot's current velocity and pose relative to the closest sidewalk edge.

4.2 Comparison to Starship's current solution

The developed solution is compared to the existing solution by Starship Technologies in a black box style. To do this both were executed in the same simulated environment and then compared according to evaluation criteria. In addition to this both were run on a Starship robot to measure and compare the CPU load.

4.2.1 Test case performance comparison

In the first case the existing solution stops at the corner of the 90 degree turn. This increases the time it takes to drive and can also surprise pedestrians as it requires stopping and sharp turning as seen in figure 7. The proposed solution's path takes a smoother turn located on the following sidewalk. The deviation from the sidewalk edge can be considered minimal as the difference in position is less than half a meter. Comparing the costs of the two solutions it is visible that the current solution has a peak at the corner and higher peak cost than the proposed solution (figure 8).

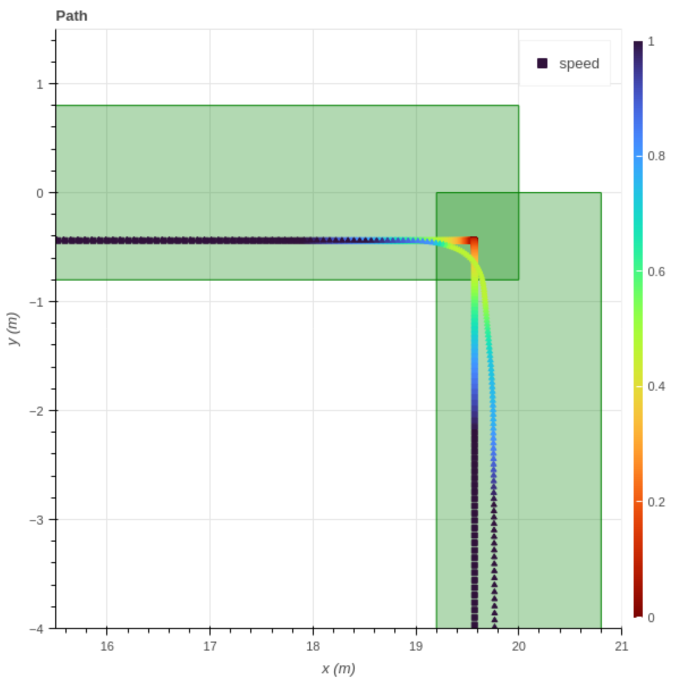


Figure 7. Case 1 fraction of max speed along the path with the existing solution with squares along the path and the proposed solution with triangles.

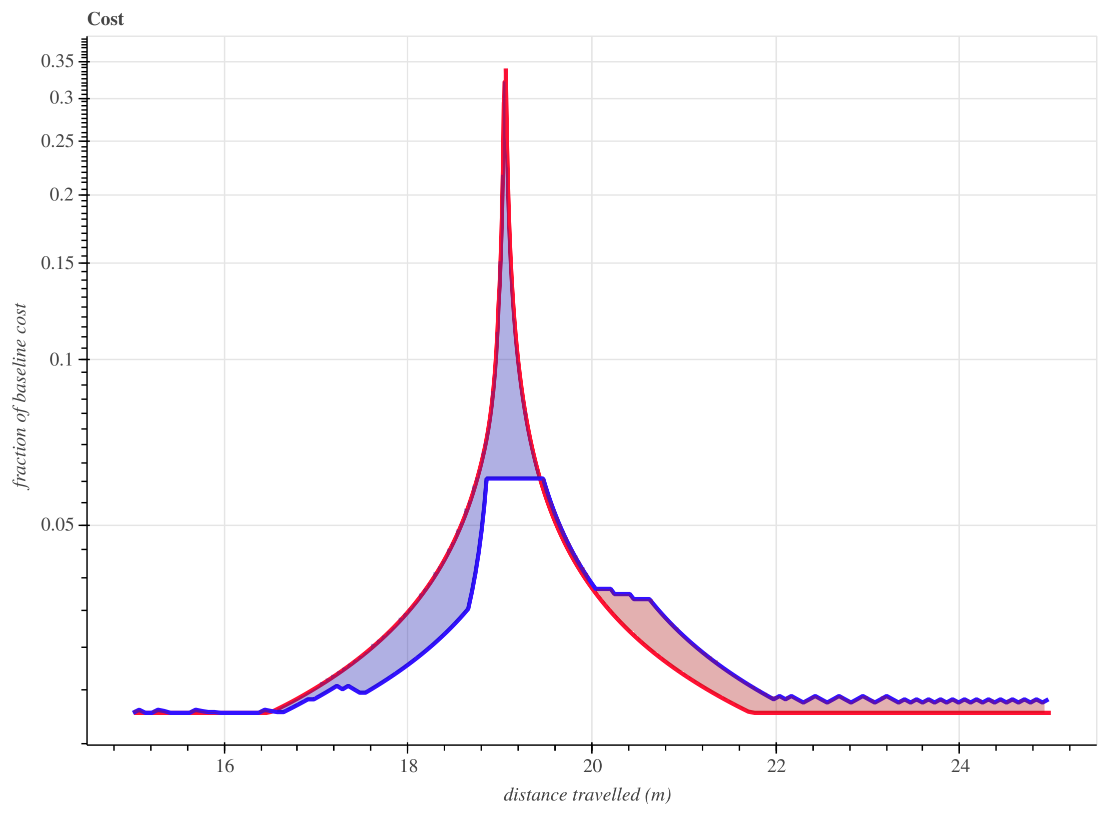


Figure 8. Case 1 cost along the path with the existing solution in red and proposed solution in blue

In the second case the existing solution's behaviour has similar patterns as in the first one where the robot stops at the corner (figure 9) which results in the similar pattern of cost visualized in appendix 8. The cost is also delayed with the proposed solution as it performs the motion over a longer distance. One clear difference from the first case is that due to the steeper angle the proposed solution moves closer to the opposing sidewalk edge before returning to the designated side. By reaching the middle of the sidewalk this may propose a difficulty to other pedestrians trying to pass the robot at the same time (either from the same or opposing direction). In the third case with the existing solution robot turns sharply near the end of the connecting segment (appendix 9) creating problems regarding both smoothness and economy (appendix 9). The proposed solution can utilize the edge of the sidewalk for a transition with a smoother pace. The existing solutions cost peak with the slopes before and after from the acceleration and decceleration create a bigger area of cost per meters compared to the proposed solution.

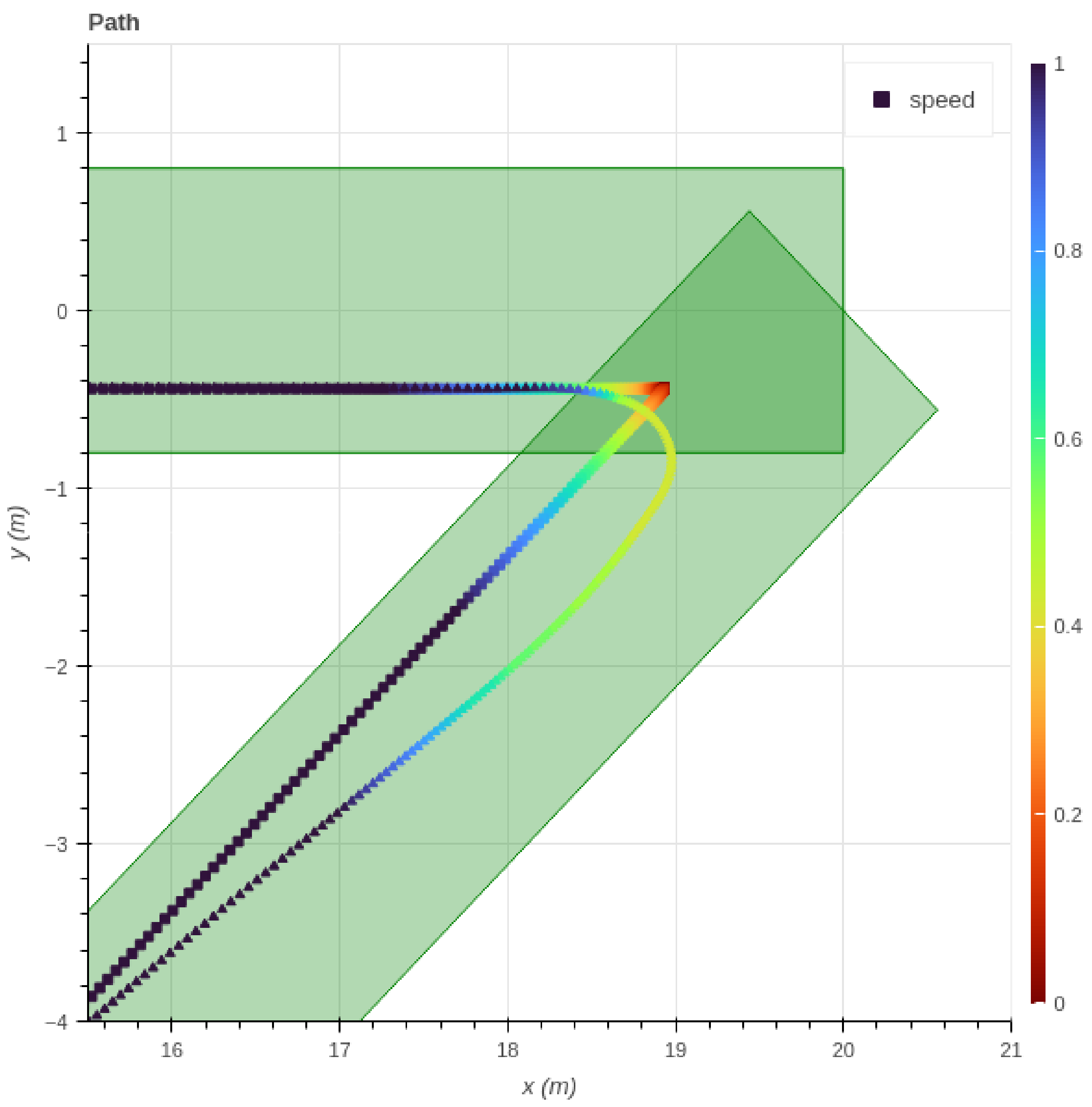


Figure 9. Case 2 fraction of max speed along the path with the existing solution with squares along the path and the proposed solution with triangles.

In the fourth case the existing solution can perform well in regards to economy (appendix 10) but the movement involves multiple lateral oscillating movements after the turn which reduces the predictability of the robot as can be seen in appendix 10. The proposed solution over prioritizes driving to the road edge creating an angle in the path taken. This makes the average cost of the proposed solution higher than the existing cost.

For the fifth case the existing solution takes a path closer to the center of the sidewalk with both inside and outside road rule as seen on appendix 11 while the proposed solution has more circular path. In the fifth case the existing solutions both inside and outside road rule of the sidewalk are relatively good speed but have multiple oscillations of speed and turning which can be seen in the uneven cost as well as fluctuations in cost in appendix

11. The average cost of proposed solution is slightly higher than the average cost of the existing solution. The proposed solution has more even cost with less spikes.

Overall the proposed solution shows more stability in speed which leads to more stable cost. In some cases the cost spikes of the existing solution reach higher than the average cost of the proposed solution. The proposed solution shows most potential in reducing cost by removing the spikes and with further development to lower the average it could provide consistent benefit.

4.2.2 CPU load comparison

To assess the feasibility of the proposed solution it is important to consider the added load on the CPU. This is important because if the computational cost becomes too great then it limits the selection of hardware and platforms that can use it. An important factor to consider when evaluating the computational cost is that there are many different platforms with different performance. Therefore not all platforms may be able to support the load required.

To evaluate the CPU load in a realistic environment the solution was run passively in background on a Starship robot in real world testing environment. The proposed solution was evaluated at different node count configurations. To give a more comprehensive overview of CPU load is measured as difference from the average percentage of core used. This shows how many percentage points more CPU core was used compared to the existing Starship solution without the proposed solution in the background. The measurements were taken while driving on sidewalk.

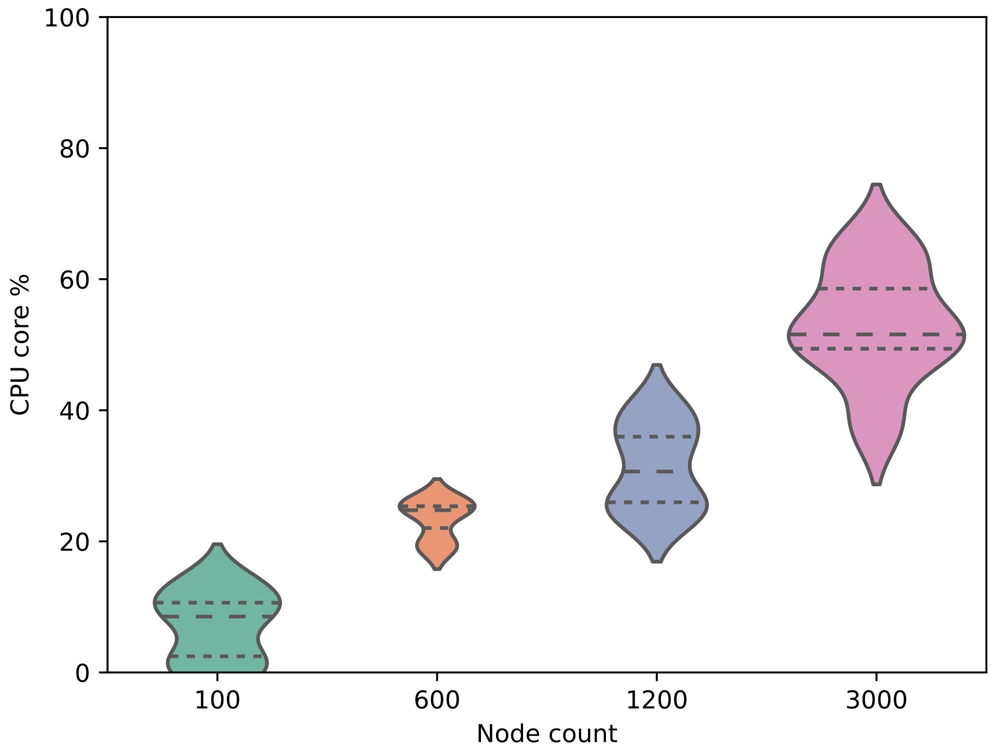


Figure 10. CPU load % with different configurations of created node counts each cycle. Long dashed lines mark the median and short dashed lines mark the quarters.

As seen in figure 10 the configuration with 600 nodes used about 25% of a core and the configuration with 3000 nodes uses about 51% of a core. This shows that running this solution is feasible in cases where at least 30% of a core is available. It should be noted that if hardware has multiple cores and the solution is implemented to use multiple threads then it is possible to use more than 100% of a core. The bigger node count also comes with a wider spread in the measured CPU loads as there is a higher possibility of reaching different trees with different average computation costs. These differences in average computation costs also get magnified by the increased node count.

4.3 Comparison to public algorithms

The solution logic was compared to A*, RRT* and costmap based solutions. The solutions were compared in following criteria: discretization on solutions, computation complexity and dependence of different conditions, extendability and support for variety of cost. The purpose of this comparison was to find the limitations as well as the reasons behind these limitations to avoid these in the proposed solution.

4.3.1 Requirements and effects of discretization

Discretization is a method of turning infinitely many continuous values into a countable set of values. The usual purpose of this is to enable approaches that require the available options to be covered through a brute force methodology. The main real numbered dimensions in path planning are the X and Y coordinate and the heading of the robot pose. In addition to this the velocity and time add additional real valued parameters to possible states acquired by the robot.

The A* depends on a graph which is a discretized representation of the world. Because of this possible poses are limited to the vertices in the graph. This creates significant reliance on the method of generation for the graph. This method needs to create a graph that would follow the movement constraints set by the platform.

RRT and RRT* use randomly generated points in the space and the poses are therefore not limited to a discrete set of values. However the step sizes in the tree are discrete values which means there is discretization of time. Therefore branching can only happen at certain intervals.

A generic costmap described by a combined function of cost has no discretization present for neither the pose nor the time. However if this costmap is represented by cells then these cells create discretization of space. Each of the values used as dimensions for the costmap will be discretized if cell representation is used.

In the proposed solution the vertices in the tree are in continuous space and therefore the poses are not discretized. However the step lengths between poses are limited to the given options and are therefore discrete.

Table 1. Discretization overview of different algorithms.

Algorithm	Pose discretization	Time discretization
A*	Yes	No
RRT & RRT*	No	Yes
Costmap with functions	No	No
Costmap with cells	Yes	No
Proposed solution	No	Yes

As visible in table 1 the least constraining solution for discretization is a costmap that uses functions to store the cost. A* and costmap with cells can be categorized as pose discretizing which limit the achievable poses, but not the moments at which actions can be taken. RRT* and proposed solution can be categorized as time discretizing which limit the moments at which actions can be taken, but not the achievable poses. Due to space being already significantly constrained by the sidewalk we should consider further discretization of poses as a negative drawback as it may exclude all valid solution with too broad discretization. Since most control algorithms for robots already work with a given frequency we may consider the time already externally discretized and therefore as long as the solution's discretization aligns with the external discretization this should not limit the available options.

4.3.2 Complexity

Computation complexity describes the rate at which the required work increases as the given parameters increase. Space complexity describes the rate at which required memory increases as the given parameters increase. The complexity can be considered in a single path generation as well as continuously. The main parameters for the complexity are the distance to goal as well as environment features. Some of the most important features in the environment are the area to be explored as well as number of obstacles.

For A* the computational and memory complexity is $O(b^d)$ where b is the branching factor and d is the depth to goal. This means that it has exponential complexity towards distance to goal. The heuristic function is the main tool here to reduce the effective branching factor.

Complexity of RRT is at worst $O(N^2)$ where N is the number of vertices created. This can be improved by using a boxing method where vertices are grouped into grid cells based on location to reduce the number of vertices that need to be checked to connect the new vertex to the tree. The number of vertices depends on the distance between consecutive vertices as well as the distance to the goal. For the space complexity it depends on the number of vertices created and if left unlimited will keep on growing.[18] Due to the randomness of the generation of the vertices and the possibility to add bias towards the goal exact complexity can be hard to define in regards to environment parameters. However the approximate average number of vertices can be defined as linearly related to the required area to be explored.

The complexity of the costmap comes from the gradient descent. In case of a costmap defined by a grid the gradient descent direction will have to be determined by sampling. This makes the time complexity $O(knd)$ where k is number of iterations, n is number of samples and d is number of dimensions. In addition to this comes the complexity of generating the grid from which the samples are taken. This is most likely to have a linear complexity to the area considered as well as number of affecting features such as obstacles. With a costmap defined by a differentiable function the time complexity comes from differentiation. This makes the final complexity $O(ktd)$ where k is number of iterations, t is number of differentiable functions and d is number of dimensions. Expressed in this way number of iterations is linearly proportional to distance to the goal. The number of differentiable functions is most likely to have at least linear complexity towards number of features in the obstacles. The number of dimensions stays constant throughout the implementation and can therefore be disregarded in the O notation when considering changes in the environment and not the implementation. With this consideration we can get the time complexity $O(knf)$ for the grid based approach and $O(kf)$ for the differentiation based where k is the number of iterations, n is the number of samples and f is the number of features. For the space complexity the grid based approach has linear complexity towards navigable area as that defines the number of grid cells that need to be stored. For the differentiation based approach we need to store the functions which gives linear complexity towards number of features, but in most cases should remain at values where it can be considered negligible.

Since the proposed solution uses a tree-based structure and iterative generation the space and time complexities are $O(b^d)$ where b is the branching factor and d is the depth of the tree. The depth of the tree can be considered linear with regard to distance to the goal. The branching factor depends on the quality of the chosen cost functions, but will always be greater than or equal to 1.

Table 2. Complexity overview of different algorithms in regards to explorable area A, distance to goal D, number of features F and branching factor b.

Algorithm	Time complexity	Space complexity
A*	$O(F * (b^D))$	$O(A)$
RRT & RRT*	$O(F * min(A, D^2)^2)$	$O(min(A, D^2)^2)$
Costmap with functions	$O(DF)$	$O(F)$
Costmap with cells	$O(AF + D)$	$O(A)$
Proposed solution	$O(F * (b^D))$	$O(b^D)$

To compare the different solutions to each other we need to estimate their complexities relative to common parameters (table 2). When comparing the different algorithms the first thing we can notice is that all of them have linear dependence on the number of features. However that number is multiplied by different functions depending on the solution. The number of vertices for RRT and RRT* depends on the explored area which is limited by the smallest of either the explorable area or area surrounded by points at same distance to the start as the goal. RRT, RRT* and costmap with cells depend on the area for time complexity. This can be considered a significant drawback since for an intelligent and adaptive solution we want to be able to give it as much area to use as possible. Being able to provide most freedom to the system and not having to worry about performance drops is a great indicator that a system has potential for longetivity before limit of best possible performance is reached. Such solution uses the given freedom efficiently instead of exhaustively searching the space. Comparing the space complexities of the algorithms we can see that A* can avoid dependence on area in runtime only thanks to having the cost of area in the graph construction and therefore memory. It is also worth noting that both A* and the proposed solution have exponential time cost relative to the distance to the goal which means that they become unfeasible with too long distance to the goal. In terms of memory the costmap with functions performs the best since it only depends on the number of features.

4.3.3 Extendability and support for features

When comparing solutions we should also consider whether it supports taking into consideration certain preferences. We should also consider what is the expected added time and space complexity of such features. For current problem of navigating on a sidewalk we shall consider the following features: preferred sidewalk side, static obstacles, dynamic obstacles, non-holonomic movement constraints.

Having a preferred sidewalk side for path planning means better predictability for other pedestrians. This is due to people being used to walking on right or left side of the sidewalk so pedestrians moving in the opposite direction or passing others would have more space. This can be translated to preference of pose which may or may not depend on the movement direction. For all of the previously mentioned solutions adding preference based on the pose does not great significant increase in complexity.

Supporting obstacles can be one of the most complexity increasing features since there are usually many things in the detection range of the robot. For A* this can be accomplished through overlap based increase in edges. For RRT and RRT* obstacles can be handled similarly to unnavigable area as something that stops the generation and connection of vertices. However adding cost inducing but not blocking obstacles may require some creativity in implementation. Both costmap solutions would just need a cost function for the obstacles, however the cost needs to be independent of robot's pose as the costmap solution only looks at the features. For the proposed solution support for obstacles could be added by adding a cost function for these. That cost function could also contain dependence on robot's pose as that is existing information in each vertex where this would need to be evaluated.

Non-holonomic movement constraints are something that limits the available next movements depending on the path taken so far. These constraints could be soft constraints that indicate preference or hard constraints which indicate inability to perform certain actions. Since A* only looks at the shortest path to each vertex it does not possess any support for non-holonomic constraints. It is possible to add support for non-holonomic constraints to RRT by using motion primitives instead of straight lines, but that is not possible for RRT* since that also requires reconnecting of vertices. The only way to have some knowledge about past movements in costmap is by adding time and orientation as dimensions to the map. Adding new dimensions to the costmap significantly increases the time it takes to compute since the derivative needs to be calculated in each dimension. It is also important to note that non-holonomic constraints mean that there are some areas of the costmap that would be undifferentiable making only the sampling based solution viable. For the proposed solution the non-holonomic constraints are already built into the simulation part of the step prediction which makes these constraints inherently followed.

4.3.4 Conclusion on proposed solution

In conclusion the proposed solution stands out in terms of time and space complexity by being independent of navigable area while still supporting non-holonomic movement constraints. Although not implemented as a part of this thesis it has the necessary structure for adding obstacles as features. The main downside is the exponential complexity towards distance which makes planning for very long distances difficult. It also doesn't require discretization of poses which allows for smoother and more free movement in the space. It does however require discretization of time into segments which reduces the smoothness of actions and requires tuning of discretization parameters.

5. Summary

The purpose of this thesis was to develop a path planning solution that can be used online in a system with limited compute to generate a local path for movement. To achieve this a tree-based path generation algorithm was developed. To evaluate the result a black box comparison with Starship Technologie's current solution was performed on a set of constructed test cases in a simulation environment. In addition to this, comparison of logic and limitations was performed with common path planning algorithms: A*, RRT* and costmap.

During the simulations the pose and the velocity of the robot was used to gauge performance and calculate the cost of completing the task. Solutions with lower cost were considered to be better.

For the analytical comparison of limitations and capabilities the following challenges were considered: effects of discretization on solutions, computation complexity and dependence of different conditions, extendability and support for variety of cost. These properties were compared in worst and average case scenarios.

The proposed solution was superior in terms of support for continuous space and reusability of generated paths. The weakness of the proposed solution's lies within the selection of step lengths which has significant effect on computation complexity as well as quality of found results. Considering these pros and cons it can be concluded that the proposed solution is good for local path planning, but suboptimal for global path planning.

References

[1] *Amazon*. 2023. URL: http://amazon.com/scout.

[2] *Kiwibot*. 2023. URL: https://www.kiwibot.com.

[3] *Clevon*. 2023. URL: https://clevon.com.

[4] *Starship Technologies OÜ*. 2023. URL: https://www.starship.xyz.

[5] H. Eren, Chun Che Fung, and J. Evans. "Implementation of the spline method for mobile robot path control". In: *IMTC/99. Proceedings of the 16th IEEE Instrumentation and Measurement Technology Conference (Cat. No.99CH36309)*. Vol. 2. 1999, 739–744 vol.2. DOI: 10.1109/IMTC.1999.776966.

[6] M. Mahdi Ghazaei Ardakani et al. "Real-time trajectory generation using model predictive control". In: *2015 IEEE International Conference on Automation Science and Engineering (CASE)*. 2015, pp. 942–948. DOI: 10.1109/CoASE.2015.7294220.

[7] Steven M. LaValle. "Rapidly-exploring random trees : a new tool for path planning". In: *The annual research report* (1998).

[8] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107. DOI: 10.1109/TSSC.1968.300136.

[9] Chaymaa Lamini, Said Benhlima, and Ali Elbekri. "Genetic Algorithm Based Approach for Autonomous Mobile Robot Path Planning". In: *Procedia Computer Science* 127 (2018). PROCEEDINGS OF THE FIRST INTERNATIONAL CONFERENCE ON INTELLIGENT COMPUTING IN DATA SCIENCES, ICDS2017, pp. 180–189. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2018.01.113. URL: https://www.sciencedirect.com/science/article/pii/S187705091830125X.

[10] Naim Rastgoo et al. "A critical evaluation of literature on robot path planning in Dynamic environment". In: *Journal of Theoretical and Applied Information Technology* 1070 (Jan. 2015).

[11] Alonzo Kelly and Neal Seegmiller. “A Vector Algebra Formulation of Mobile Robot Velocity Kinematics”. In: *Springer Tracts in Advanced Robotics* 92 (Dec. 2014), pp. 613–627. DOI: 10.1007/978-3-642-40686-7_41.

[12] Edsger W Dijkstra. “A note on two problems in connexion with graphs”. In: *Numerische mathematik* 1.1 (1959), pp. 269–271.

[13] Sertac Karaman and Emilio Frazzoli. “Sampling-based Algorithms for Optimal Motion Planning”. In: *CoRR* abs/1105.1186 (2011). arXiv: 1105.1186. URL: http://arxiv.org/abs/1105.1186.

[14] Samir Bouzoualegh, Elhadi Guechi, and Ridha Kelaiaia. “Model Predictive Control of a Differential-Drive Mobile Robot”. In: *Acta Universitatis Sapientiae Electrical and Mechanical Engineering* 10 (Dec. 2018), pp. 20–41. DOI: 10.2478/auseme-2018-0002.

[15] David V. Lu, Dave Hershberger, and William D. Smart. “Layered costmaps for context-sensitive navigation”. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2014, pp. 709–715. DOI: 10.1109/IROS.2014.6942636.

[16] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep Sparse Rectifier Neural Networks”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Nov. 2011, pp. 315–323. URL: https://proceedings.mlr.press/v15/glorot11a.html.

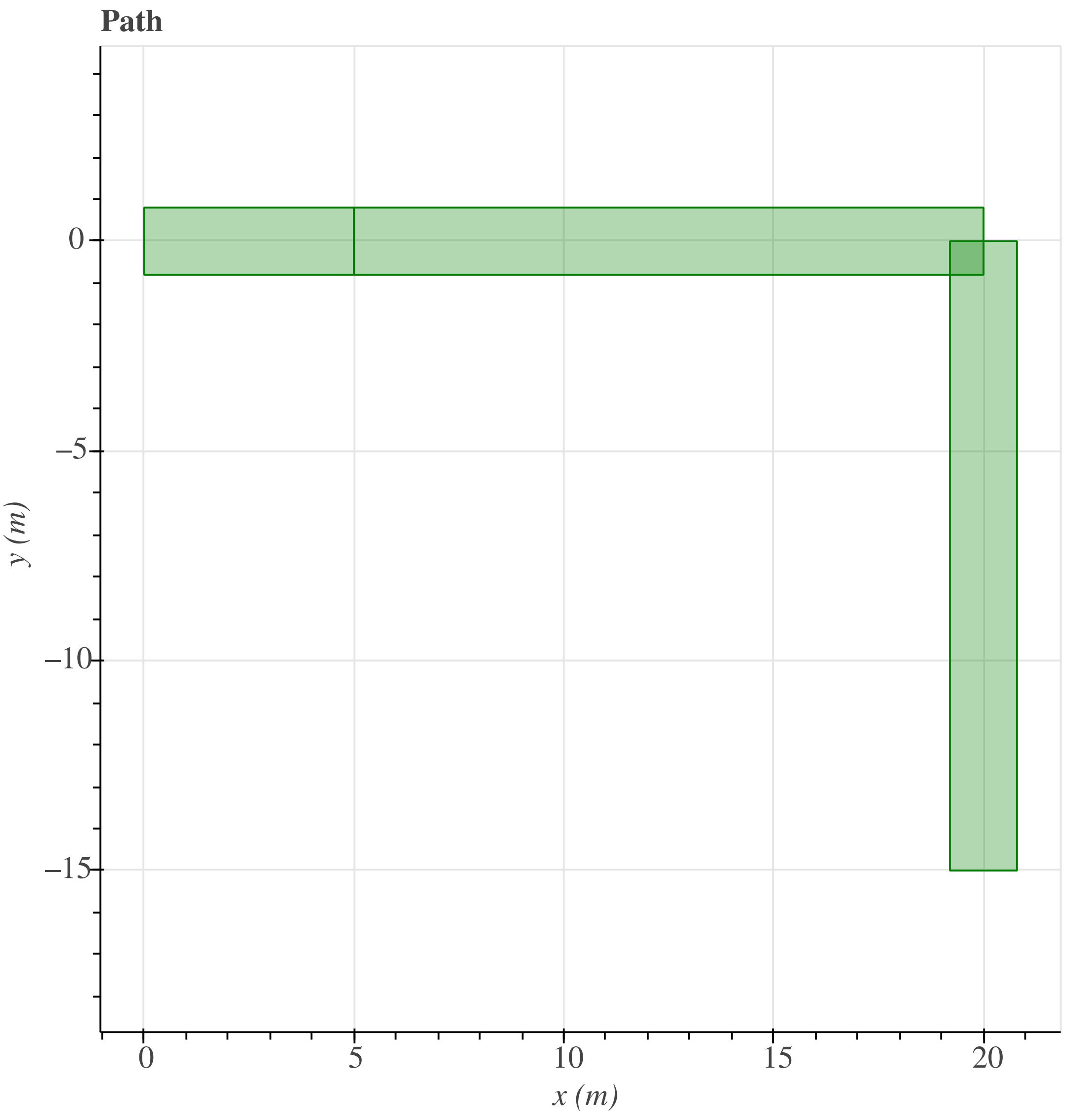
[17] P.F. Verhulst. “Recherches mathématiques sur la loi d’accroissement de la population.” In: *Nouveaux mémoires de l’Académie Royale des Sciences et Belles-Lettres de Bruxelles* 18 (1845), pp. 14–54. URL: http://eudml.org/doc/182533.

[18] Mikael Svenstrup, Thomas Bak, and Hans Andersen. “Minimising computational complexity of the RRT algorithm a practical approach”. In: May 2011, pp. 5602–5607. DOI: 10.1109/ICRA.2011.5979540.

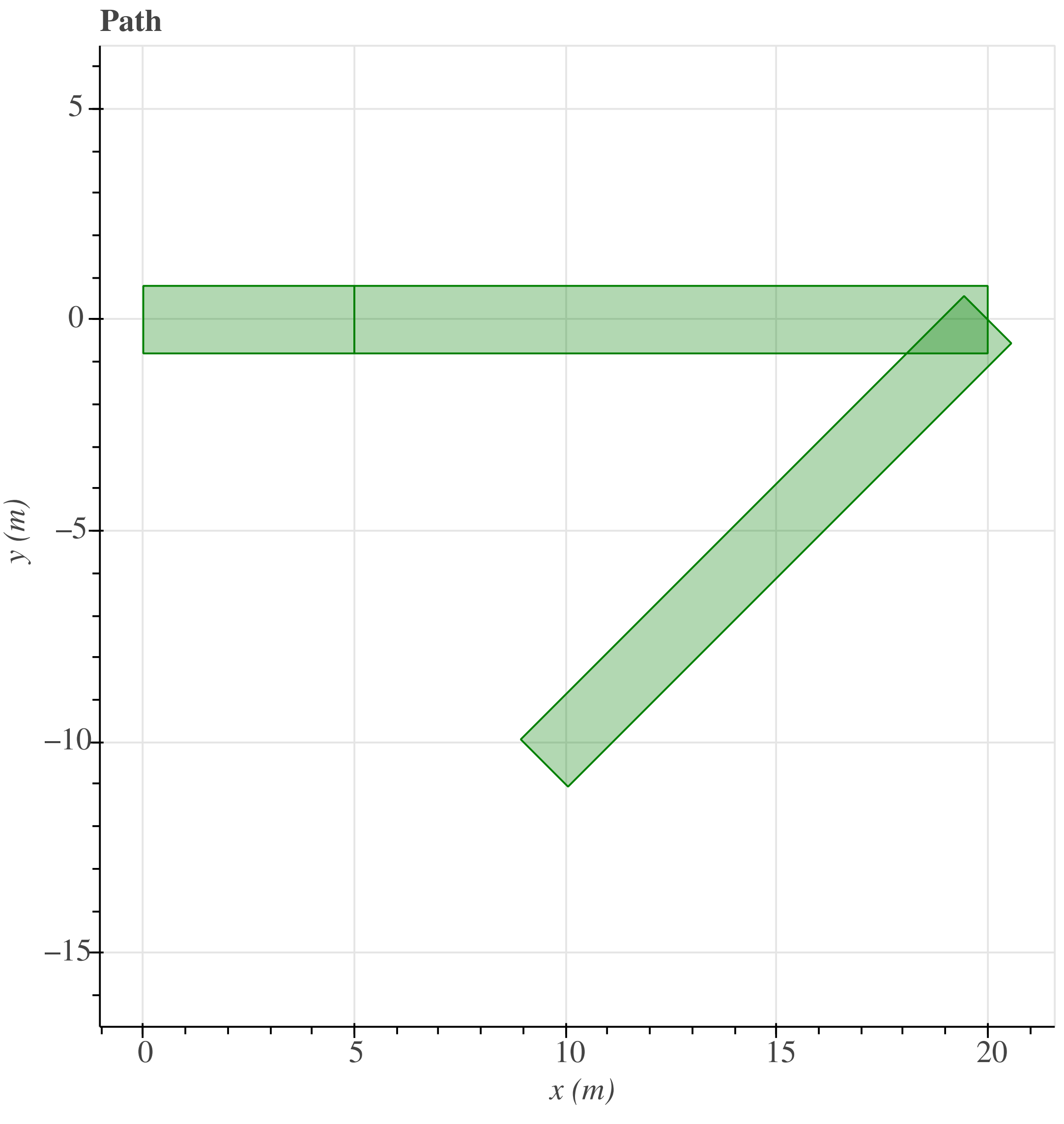
Appendices

Appendix 1 - Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

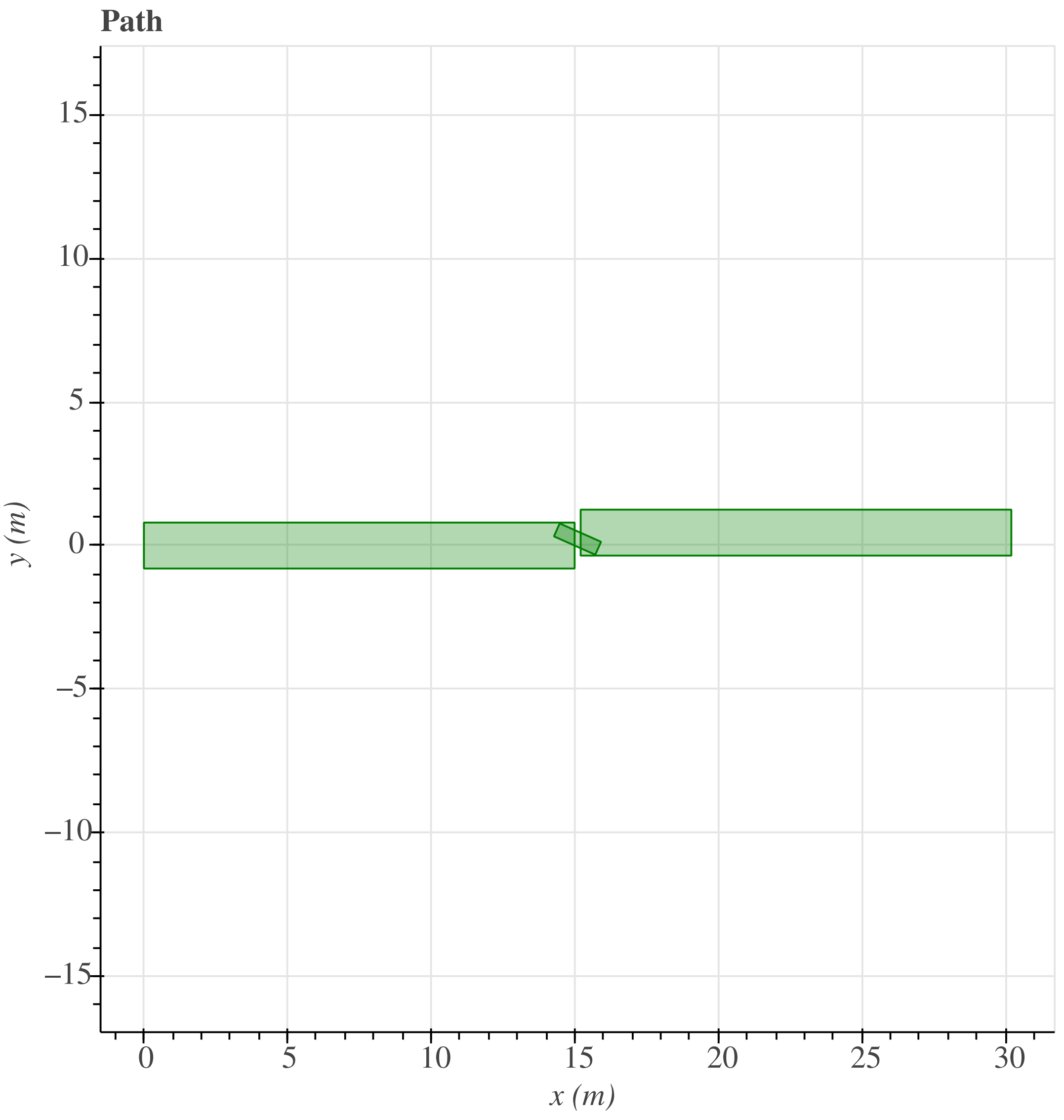
Appendix 2 - Case 1 sidewalk structure



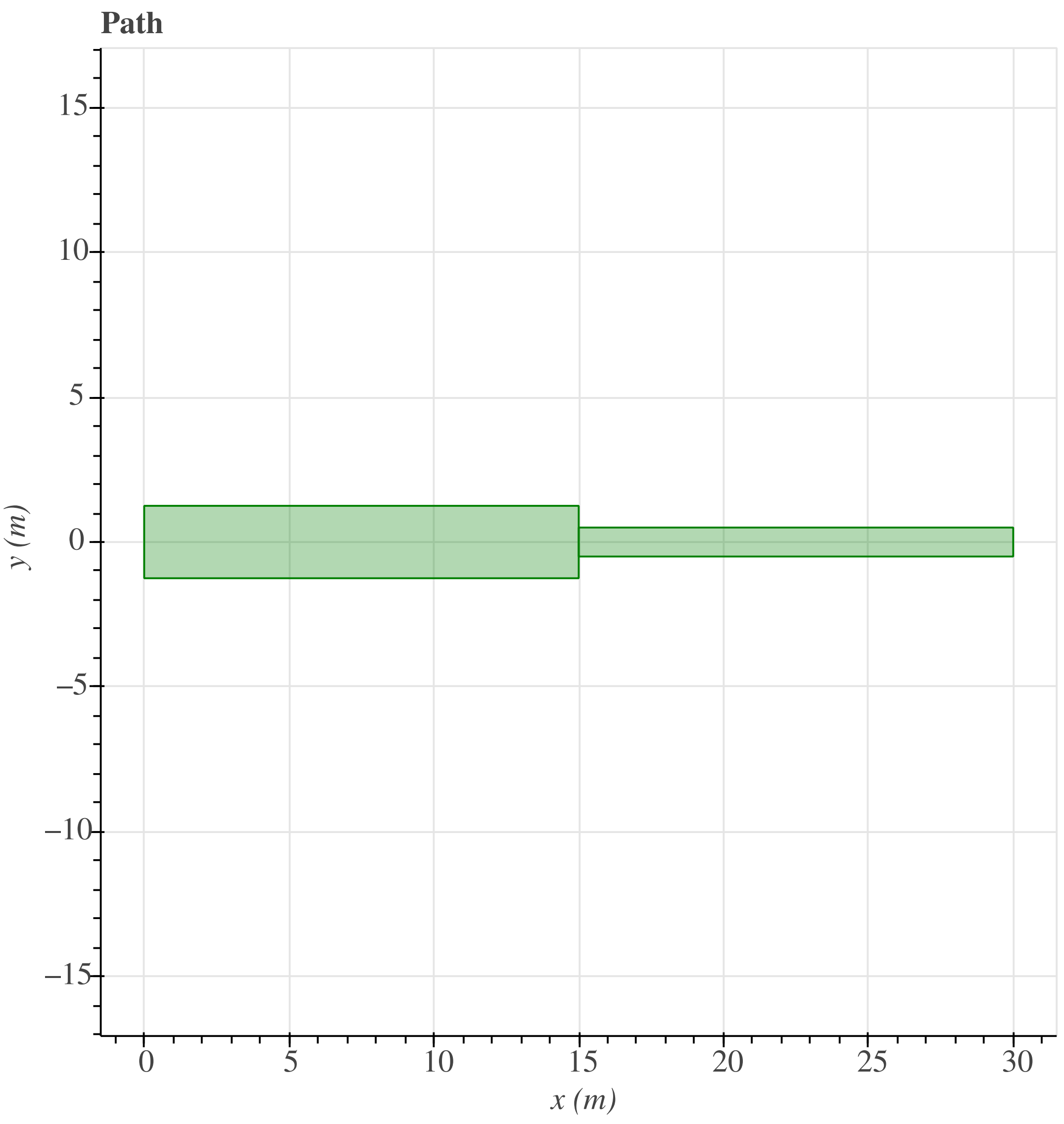
Appendix 3 - Case 2 sidewalk structure



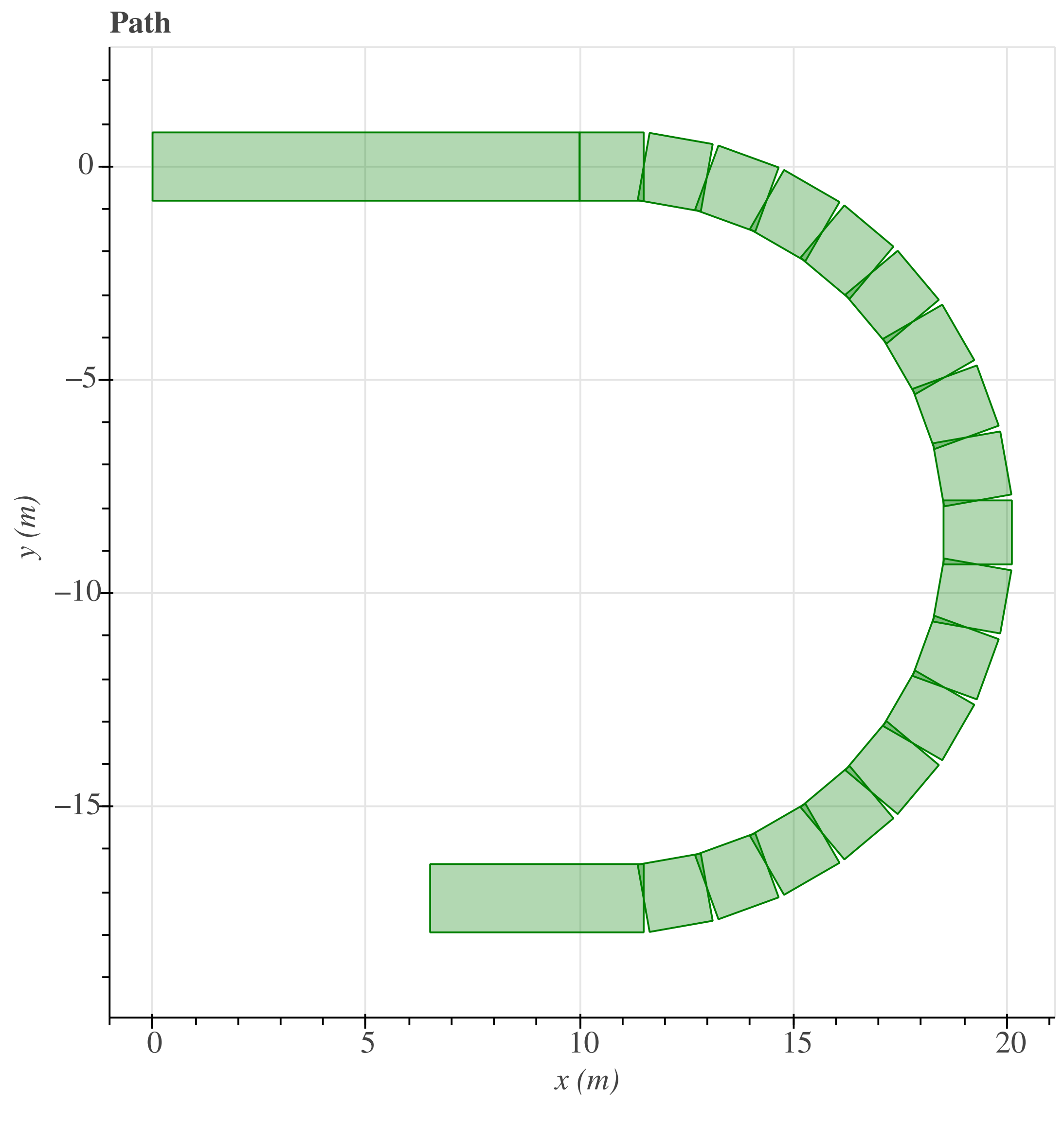
Appendix 4 - Case 3 sidewalk structure



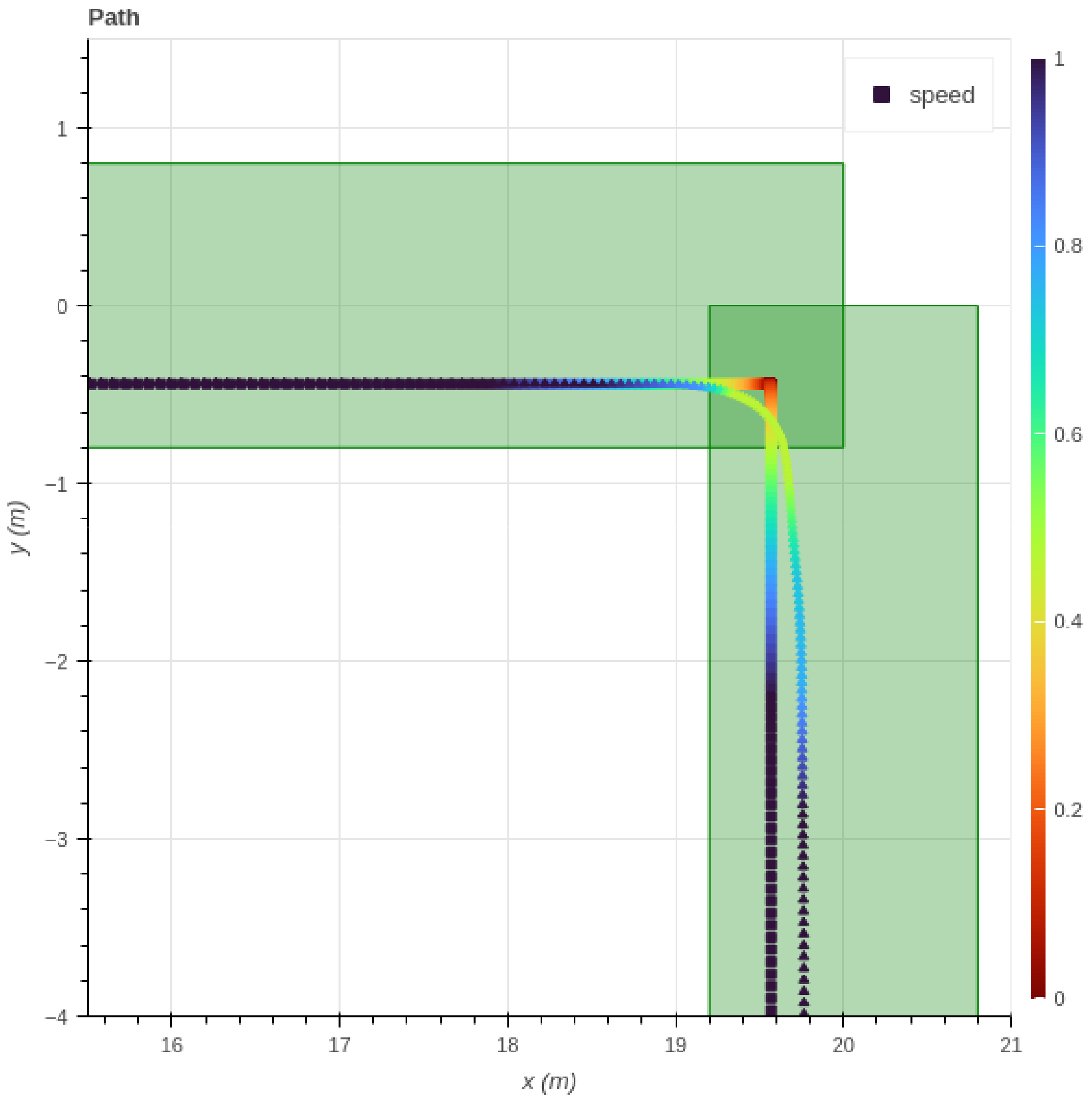
Appendix 5 - Case 4 sidewalk structure

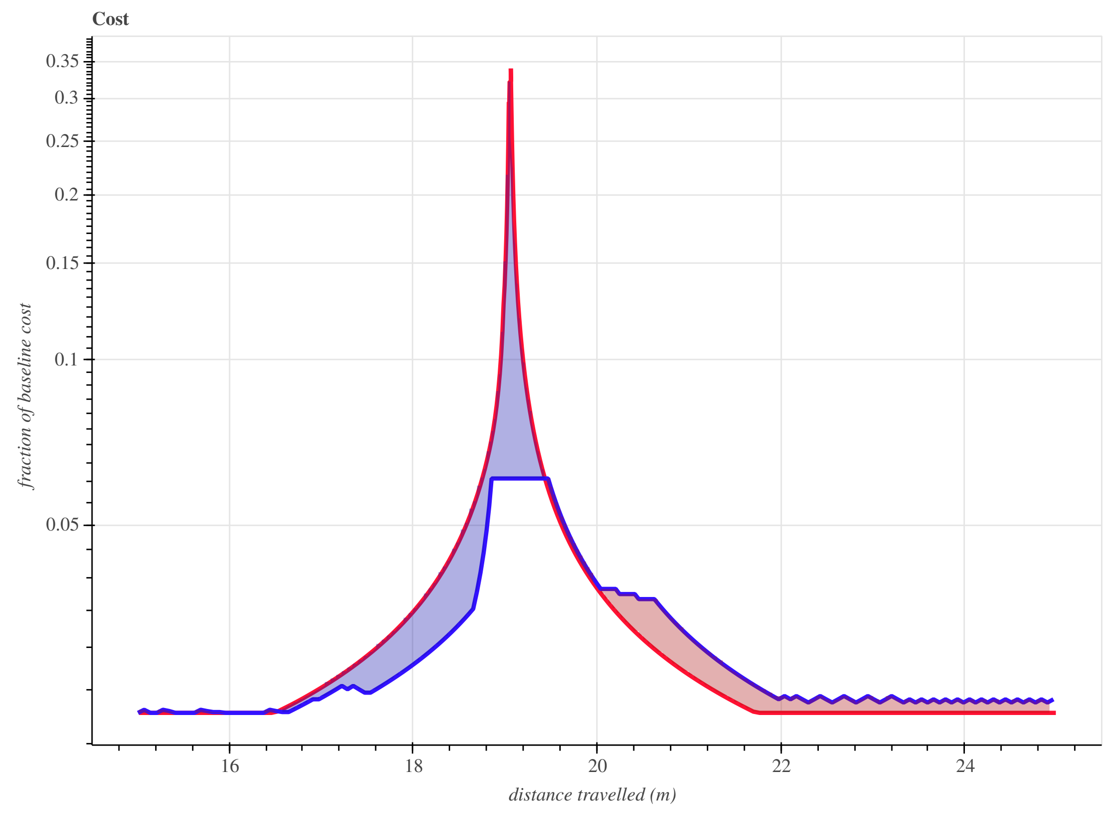


Appendix 6 - Case 5 sidewalk structure

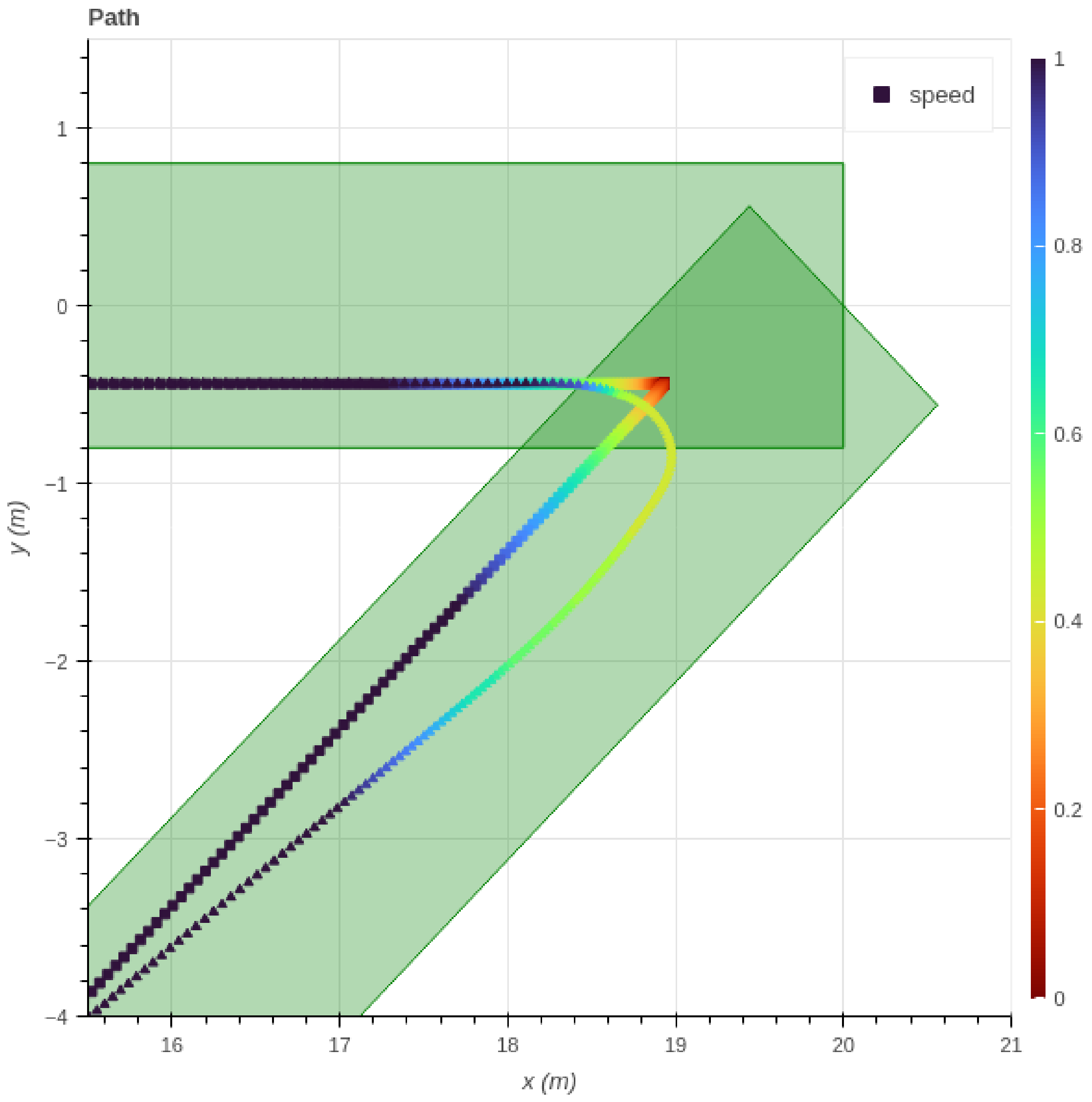


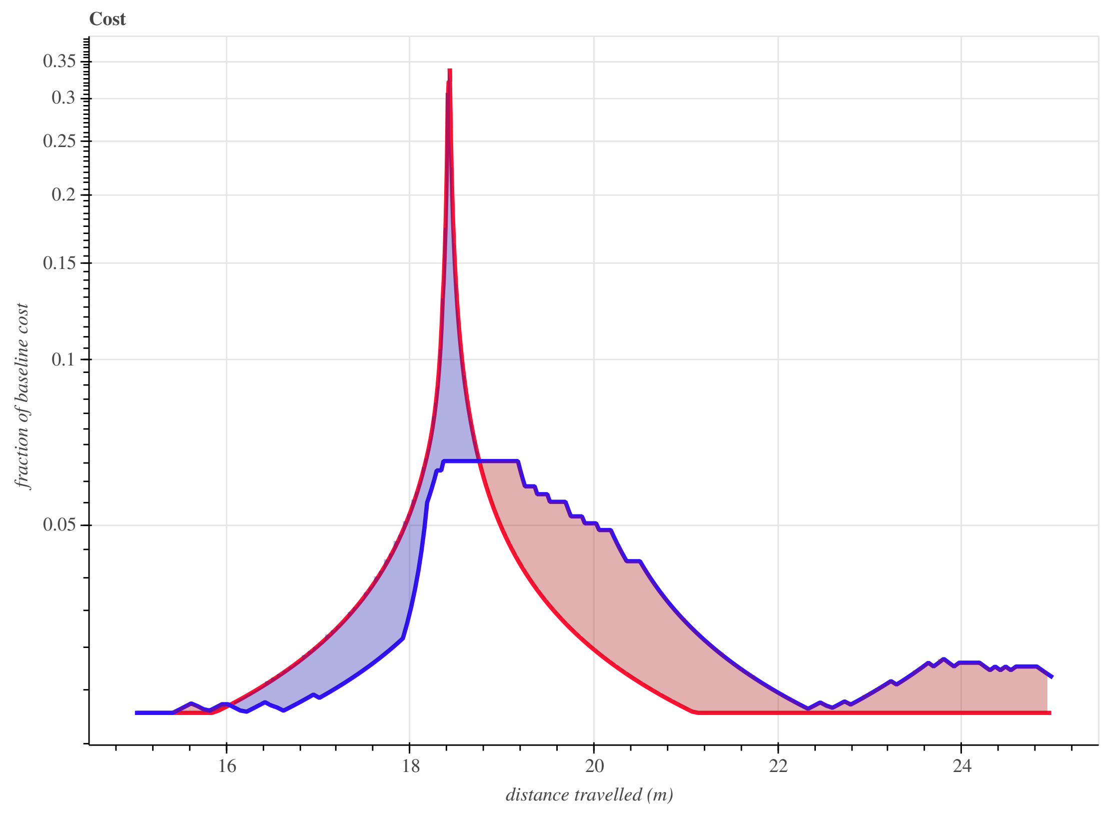
Appendix 7 - Case 1 performance




Cost
0.35
0.3
0.25
0.2
0.15
0.1
0.05
fraction of baseline cost
16
18
20
22
24
distance travelled (m)

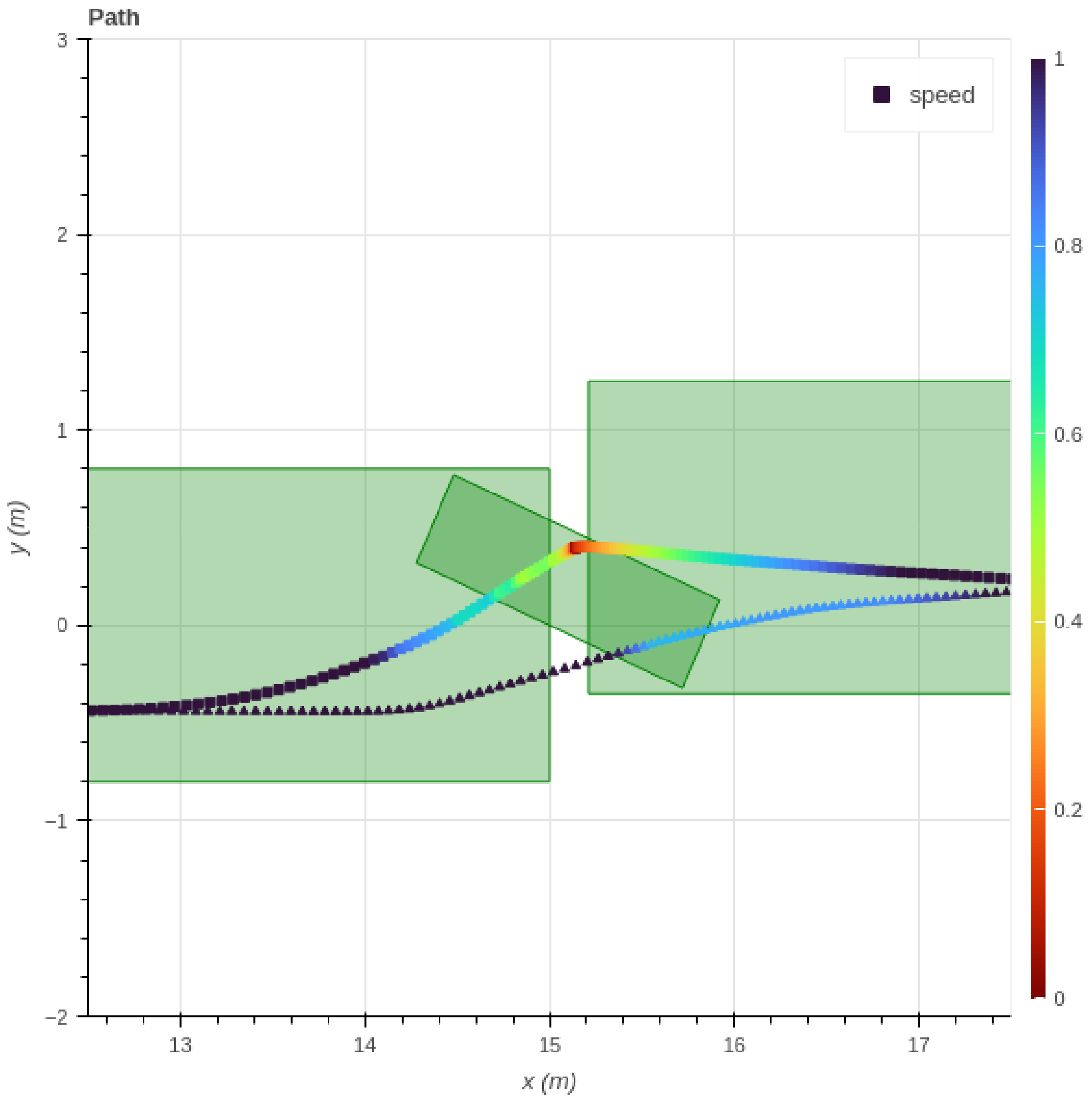
Appendix 8 - Case 2 performance

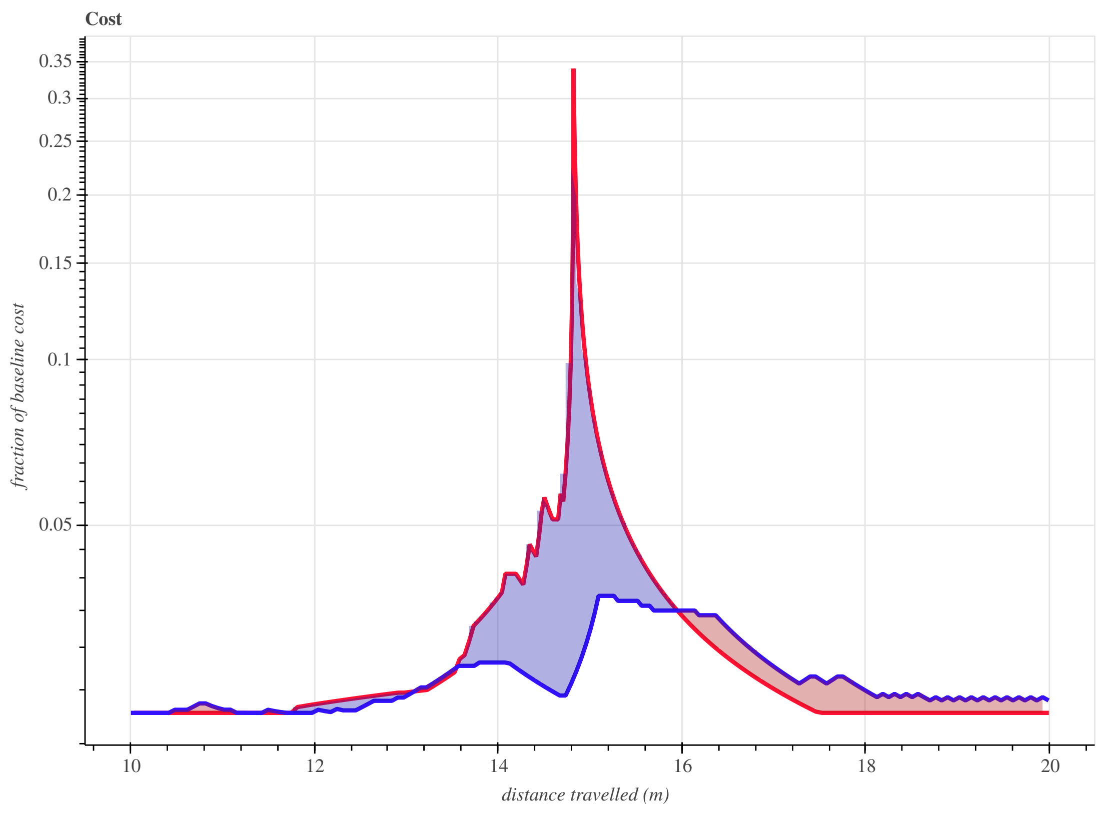




Cost
0.35
0.3
0.25
0.2
0.15
0.1
0.05
fraction of baseline cost
16
18
20
22
24
distance travelled (m)

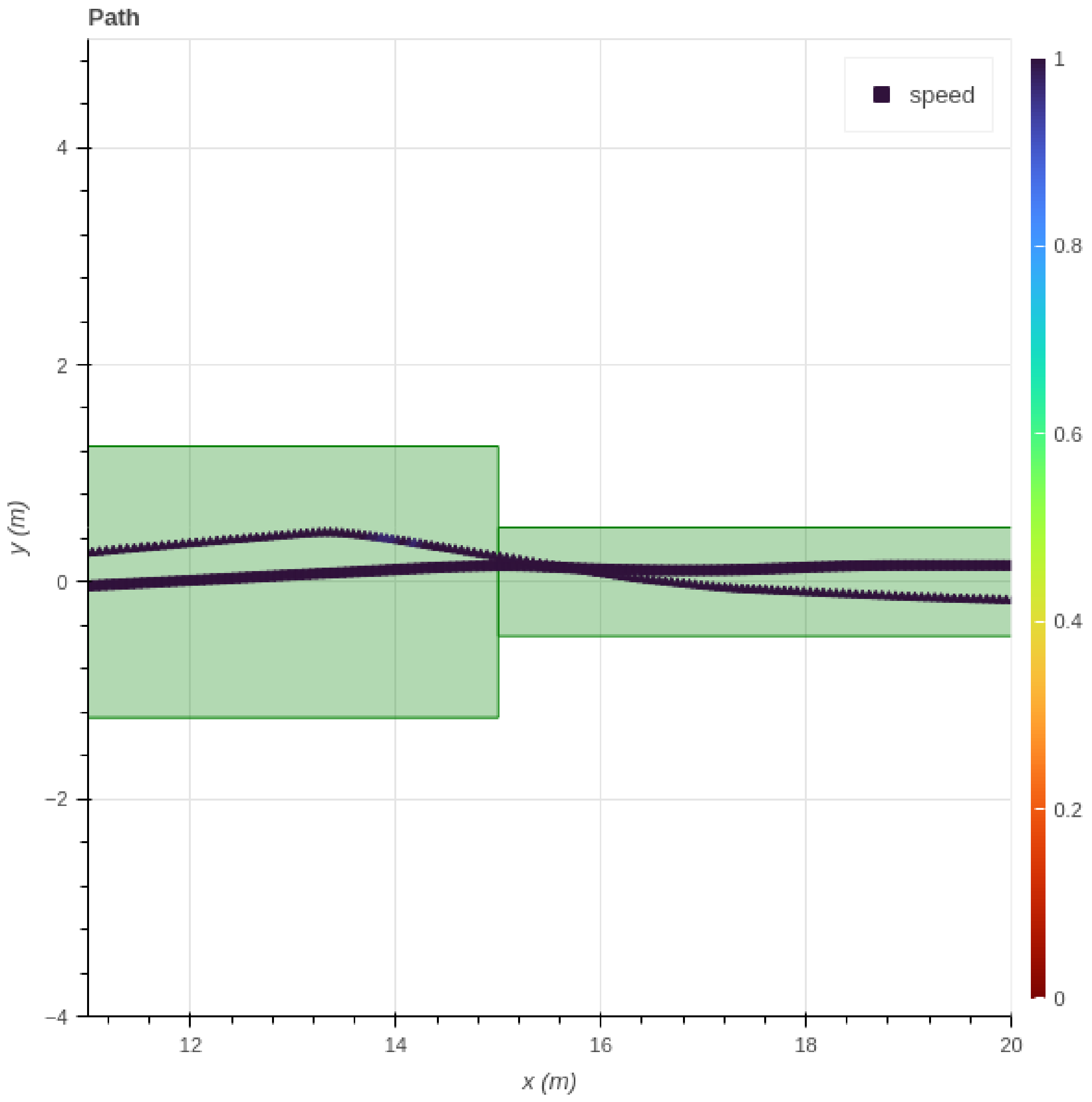
Appendix 9 - Case 3 performance

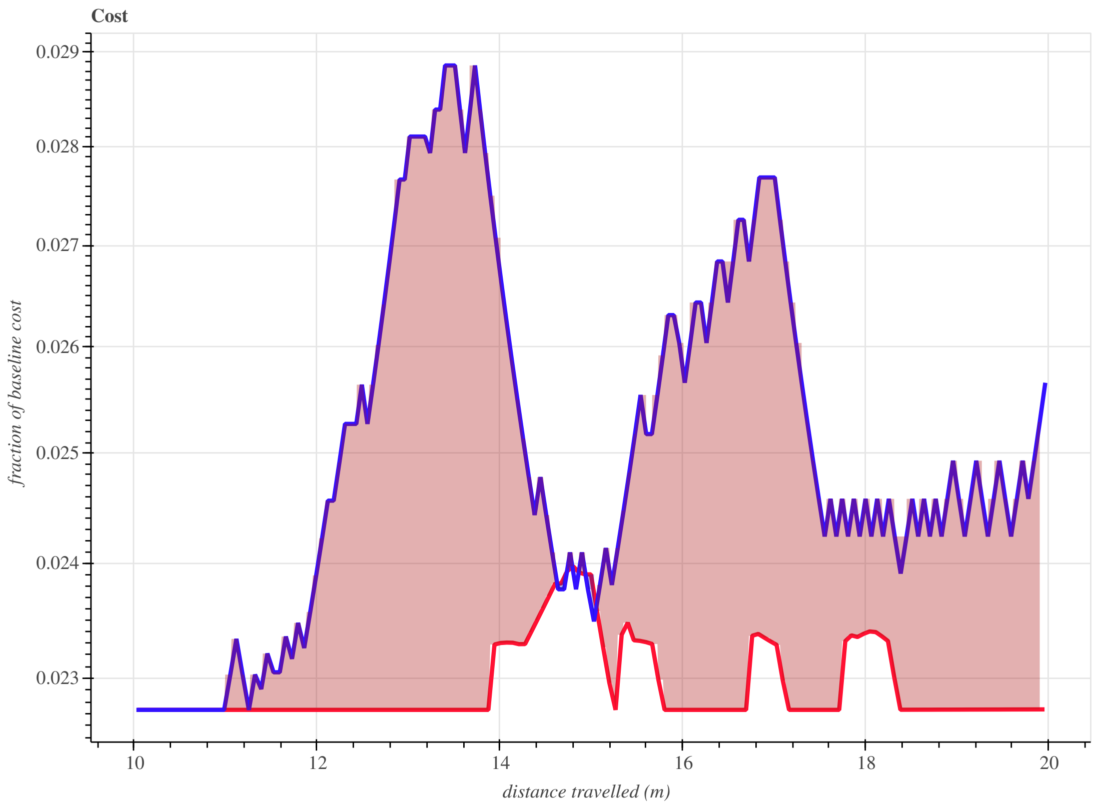




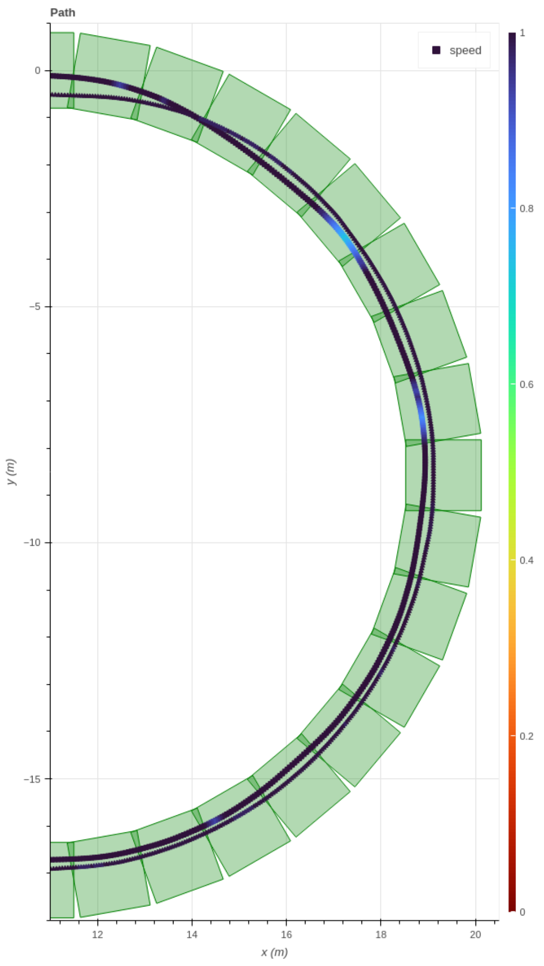
Cost
fraction of baseline cost
0.35
0.3
0.25
0.2
0.15
0.1
0.05
10
12
14
16
18
20
distance travelled (m)

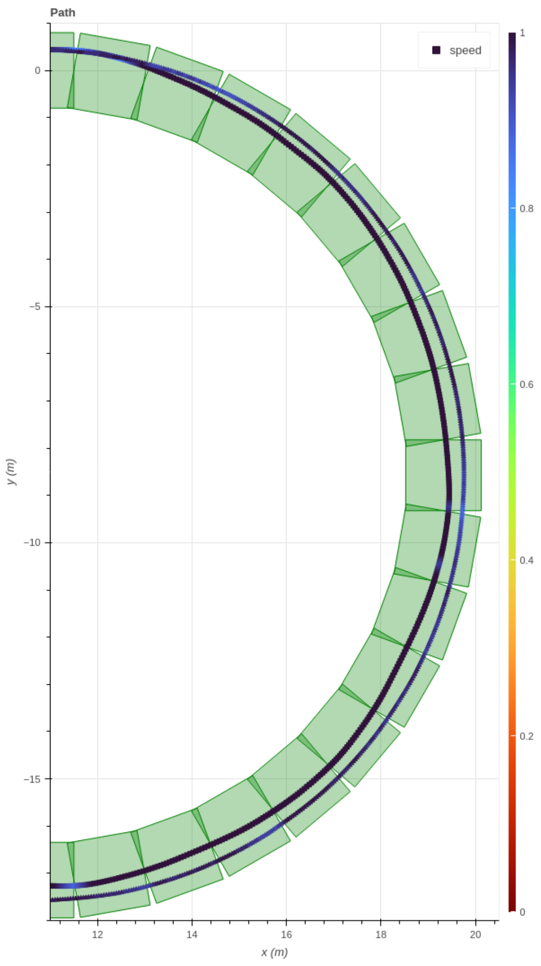
Appendix 10 - Case 4 performance

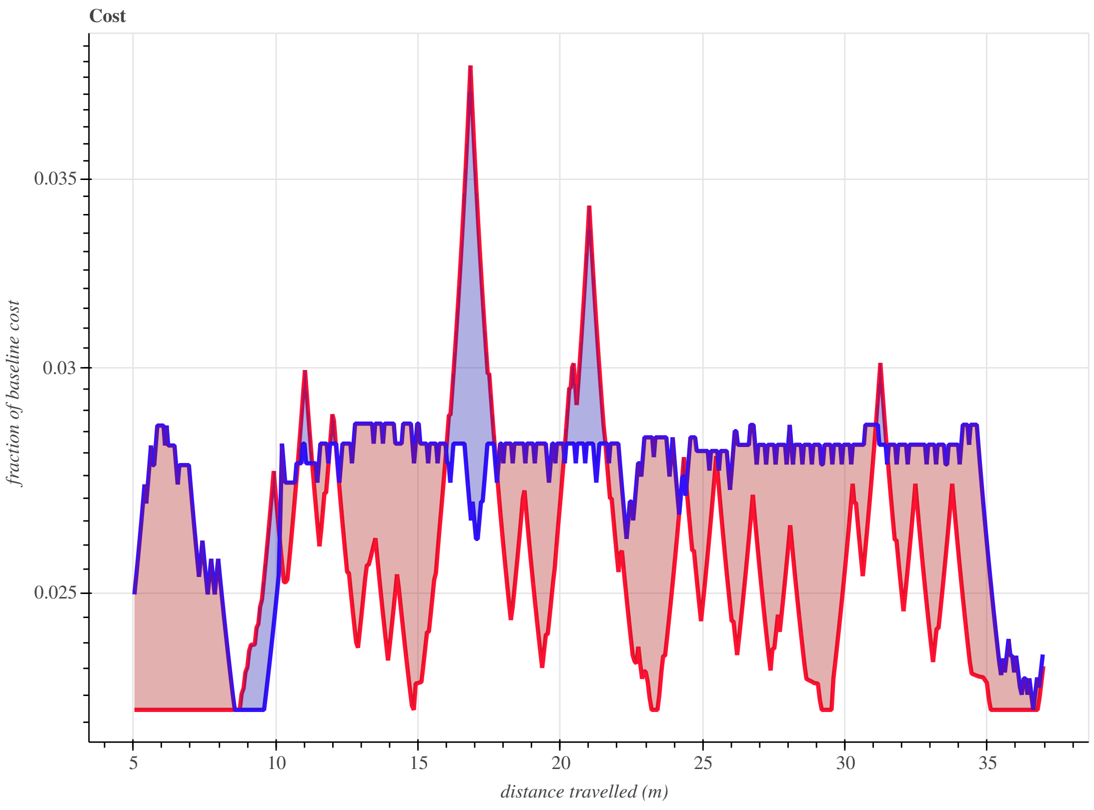


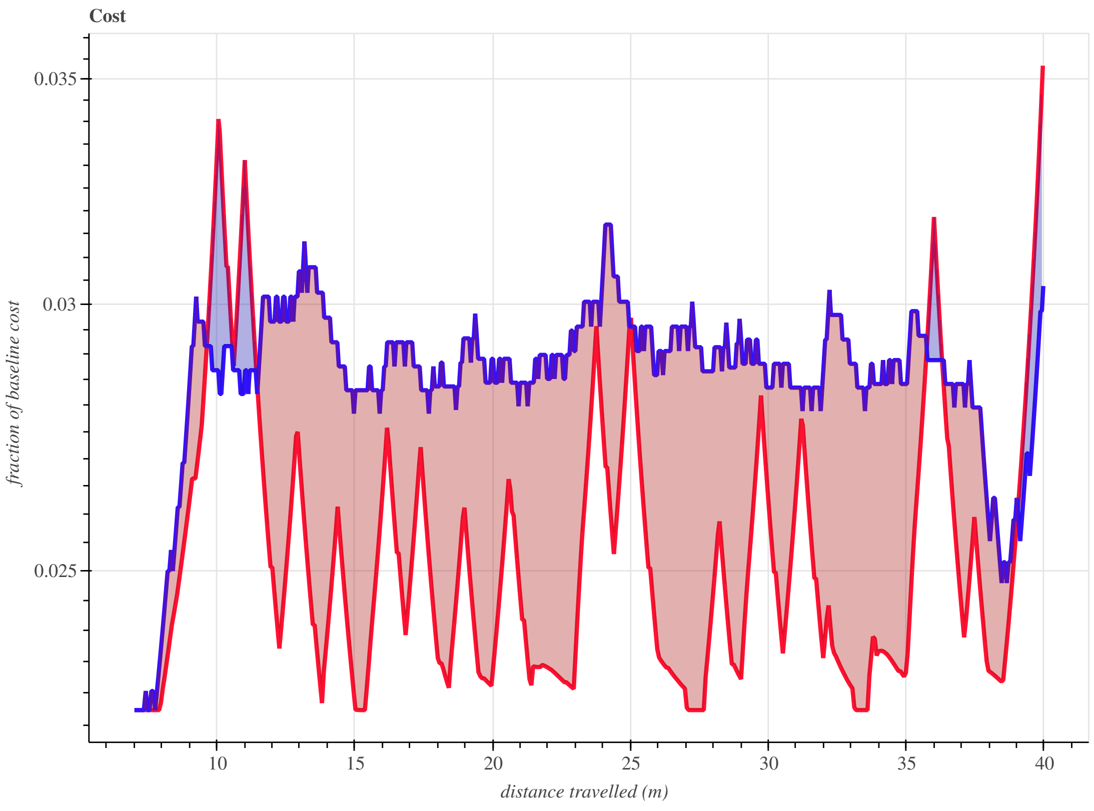

Cost
0.029
0.028
0.027
0.026
0.025
0.024
0.023
fraction of baseline cost
10
12
14
16
18
20
distance travelled (m)

Appendix 11 - Case 5 performance






Cost
fraction of baseline cost
0.035
0.03
0.025
5
10
15
20
25
30
35
distance travelled (m)


Cost
fraction of baseline cost
0.035
0.03
0.025
10
15
20
25
30
35
40
distance travelled (m)