Trajectory planning for a differential drive robot following an N-Arc path

Roberto Morassi del Blanco

4th Year Project Report
Artificial Intelligence and Computer Science
School of Informatics
University of Edinburgh

2024

Abstract

This research investigates trajectory planning for differential drive robots, with a specific focus on optimizing N-Arc path following. Differential drive robots, characterized by their independent wheel control, present unique challenges and opportunities in motion planning due to their distinct movement dynamics. Our study aims to enhance existing trajectory planning methodologies by introducing innovative approaches tailored for N-Arc paths. We address the challenge of precise N-Arc path following without the need for complete stops at every Arc intersection, presenting solutions to mitigate this issue. Our contributions include the development of trajectory planners for Arcs and N-Arcs optimizing speed and accuracy and the implementation of algorithms to smooth out splines for collision-free traversal of N-Arc sequences.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Roberto Morassi del Blanco*)

Acknowledgements

I would like to express my sincere gratitude to my parents, friends, and family for their unwavering support and encouragement throughout this research journey. Their constant presence and belief in me have been invaluable, providing the motivation and strength needed to persevere through the challenges.

I extend my heartfelt appreciation to my supervisor, David Symmons, for his guidance and mentorship. His expertise, encouragement, and constructive feedback have been instrumental in shaping this research and fostering my academic growth.

Table of Contents

Chapter 1

Introduction

For autonomous robotics, the ability to navigate and execute complex movements with precision is essential. Planning a collision-free and dynamically feasible trajectory is fundamental in robotics, with a plethora of use cases in the real world. Differential drive robots, characterized by their two independently driven wheels, offer a unique set of challenges and opportunities in motion planning due to their distinct movement dynamics.

This research delves into the intricate process of trajectory planning for differential drive robots, focusing on the optimization of N-Arc path following. The core objective of this research is to enhance the existing methodologies of trajectory planning by introducing an innovative approach tailored for N-Arc paths. These paths, composed of consecutive Arc segments, are a natural fit for the movement patterns of differential drive robots. By optimizing the robot's trajectory, we aim to achieve a seamless transition between arcs, ensuring smooth navigation while adhering to the robot's kinematic and dynamic constraints.

We present a comprehensive study that begins with a foundational overview of general robotics and motion planning. It then progresses to a focused examination of differential drive robots and the specific techniques employed in their trajectory planning. Through rigorous experimentation and analysis, we strive to push the boundaries of robotic navigation, ultimately contributing to the advancement of autonomous robotics.

Motivations

N-Arcs, although being a natural fit for the movement patterns of differential drive robots offer a range of challenges that are not present with other path representation techniques. Particularly, as will be explained later in the report, if a differential drive robot wants to follow an N-Arc path with exact precision (without deviating), it must come to a complete stop at every Arc intersection.

In this paper we examine this pitfall and come up with solutions to mitigate it. In the process of doing so, we come up with novel ways of carrying out Trajectory Planning.

Contributions of this project

- Examined the benefits and drawbacks of using a geometric way of representing a vehicle's path.
- Created a trajectory planner for Arcs that attempts to follow the Arc as accurately and as fast as possible, without using the safety buffers.
- Created a trajectory planner for N-Arcs that attempts to follow the N-Arc as accurately and as fast as possible, without using the safety buffers.
- Created a trajectory planner for Splines that attempts to follow the original N-Arc path as accurately and as fast as possible, while using the indispensable part of the safety buffers.
- Implemented an algorithm to smooth out a spline. From this and in combination with the other aforementioned spline planner, a robot can traverse a sequence of N-Arcs in a fast and accurate manner without entering into collision with objects.

Chapter 2

Background

This chapter will briefly introduce the concepts and techniques necessary for understanding this research. This will include common robotics concepts, motion planning basics, and common path planning and trajectory planning techniques.

2.1 General robotics

The operating space of a robot is the physical space where it is located and is usually represented in $\mathbb{R}^3$ for three-dimensional worlds (x, y, and z axis).

The configuration of a robot, commonly represented as q, refers to the explicit description of the robot. The nature of the configuration varies between robots as it describes information such as the position of its joints, the location of the robot and its orientation respective to a reference frame in the operating space.

The configuration space, often denoted as C_{space}, represents all possible robot configurations.

The free space, often represented as C_{free}, denotes all possible configurations a robot can adopt if it complies with all its constraints and does not collide with an obstacle. The free space thus depends on the intricacies of both the robot and the environment it operates in.

The obstacle space, frequently denoted as C_{obs}, denotes all the configurations of the robot in which it does not comply with a constraint or enters into collision with an obstacle.

$$C_{free} + C_{obs} = C_{space}$$

When carrying out motion planning with mobile robots, it is often more convenient and efficient to use the operating space as the configuration space. A simplification of the motion planning problem is achieved, as the robot can avoid collisions with obstacles by checking if its configuration is in the free space or in the obstacle space.

2.2 Types of robots

There exists many different kinds of robots that are able to complete a variety of different tasks. The two types that are most prominent and appear most in the literature are articulated robots and mobile robots.

Articulated robots are commonly used in industry such as in manufacturing. Articulated robots have a series of links connected by joints. Actuators can move the robot's joints in order to achieve a particular motion (such as, for instance, to mimic human arm movements). Manipulators are be used to pick up, move, or otherwise manipulate objects in the C_{free} space. They are usually highly versatile given that their multiple articulated axes which provide freedom of movement. Importantly articulated robots do not necessarily have the ability to move around a space (the robot base is usually stationary). [Singh and Banga, 2022]

Mobile robots are the kinds of robots that have the ability to move themselves around a certain predefined space. There are many different kinds of mobile robots, but the vast majority of mobile robots fall into one of two categories: car-like robots and differential drive robots. Car-like robots are those that resemble cars in how they move: they usually have 4 wheels and two axles (one that can steer - front axle - and one that cannot - rear axle). Differential drive robots are those that resemble tanks: they usually have wheels on either side of the robot on fixed non-steering axles, turn by varying the relative rotation of the wheels on either side, and do not require additional steering motions. When navigating tight and cluttered spaces, differential drive robots have an advantage over car-like robots: they can turn on the spot.

For the purposes of this project, we will be focusing on differential-drive robots.

2.3 Path representation

A path is a description of consecutive configurations that a robot should adopt in order to achieve a certain task. For an articulated robot, this could a list of angles for each joint to move the manipulator from one place to another. For a mobile robot, this could a set of locations that the robot should move to in order to achieve a certain motion.

When representing movement in space for mobile robots, paths are almost always represented as a list of waypoints. These waypoints are pairs of coordinates on a grid (sometimes with orientation data as well) which the mobile robot should adopt in sequence. However, as shown by da Silva Arantes et al. [2019], representing paths as a discrete set of waypoints can lead to constraint violations or entering into collision with obstacles. This is due to the fact that, while the mobile robot might not be in collision when located at some waypoint, it might enter into collision when at an intermediary point in between two waypoints.

da Silva Arantes et al. [2019] proposes a novel approach to solve this issue, where waypoints in a path are moved around so that at all points in between waypoints are also collision-free. However, this approach can be computationally expensive and requires the use of commercial solvers, which limits the potential use cases of the solution.

An alternative way to represent a path is by using geometric shapes such as lines, circles, or arcs. By using this representation, we have a continuous representation that is infinitely precise (defined for infinitely small time steps). Therefore, one can be certain a robot does not enter into collision with an obstacle if it follows the designated path.

2.4 Motion planning

Motion planning is the general problem of finding a sequence of actions that moves a robot from an initial state to a desired goal state, while avoiding collisions with obstacles and satisfying the robot's constraints. For mobile robots, the initial and goals states are usually represented in the operating space.

In classical approaches to mobile robot motion planning, a divide-and-conquer approach is followed. The motion problem is usually divided into three sub-problems: path planning, trajectory planning, and tracking control [Rimon and Koditschek, 1992]. In these approaches each consecutive step is considered completely independent of the others. In this project, our focus will mainly be on the trajectory planning aspect.

2.4.1 Path planning

A path planning algorithm is one that generates a path from an initial starting point to a final objective point while avoiding obstacles.

This path is usually defined in the operating space of the robot, although it can also be defined in the configuration space of the robot. This is because the starting location, the end location, and the obstacles to avoid (which are all relevant to path planning) are all usually defined in the operating space rather than in the configuration space.

2.4.2 Trajectory planning

A trajectory planning algorithm adds time-dependent information to a previously generated path.

Particularly, a trajectory planning algorithm takes a path generated by a path planner, as well as the kinematic and dynamic constraints of the robot, and generates a trajectory in the form a time-dependant version of the path. This could be, for instance, represented as a list of waypoints with values for velocity and acceleration. By adding time-dependent information to a path, a trajectory planner is implicitly defining the inertial forces to which a robot is subjected to, as well as the efficiency of the overall motion [Gasparetto et al., 2015].

2.5 Path planning techniques

2.5.1 Artificial potential method

This approach treats the robot as a particle influenced by an artificial potential field, representing it as a point in the C_{space}. The primary objective of this method is to

guide the robot towards attractive regions while steering it away from repulsive fields within its operational environment. In this context, obstacles are assigned as sources of repulsive forces, while the robot's destination point is treated as a positive force.

One common issue encountered is the robot becoming trapped in a local minimum of the potential field. To address this problem, various solutions have been proposed, including techniques such as random walks and backtracking. [Campbell et al., 2020]

2.5.2 Cell decomposition method

This approach involves partitioning the C_{space} into cells. Cells containing obstacles are further divided into smaller cells, while obstacle-free cells are incorporated into the path sequence. These cells are then analysed to determine the relationships between adjacent ones, with the ultimate objective of constructing a collision-free path from the starting point to the destination. The process completed by identifying a path of cells that join the starting and ending points. [Lingelbach, 2004]

2.5.3 Roadmap method

This approach involves transforming the robot's C_{space} into a network representing feasible configurations and motions. There exist two main approaches to the roadmap method: the Visibility Graph and the Voronoi diagram.

In a Visibility Graph, the vertices represent the start and goal points as well as polygonal obstacles and the edges represent the edges of obstacles and connections between vertices that have a line of sight to each other. A common challenge with visibility graphs is that the resulting paths may lead the robot to collide with obstacles due to the proposed path's proximity to them (the path can be defined along an obstacles edges).

Voronoi diagrams, on the other hand, offer a solution to this issue. For a given set of obstacles, each Voronoi cell defines the set of points that are closer to one obstacle than to any other obstacle. The Voronoi diagram would then composed of the lines representing the perimeter of these cells. This approach inherently ensures the construction of the safest path for the robot because graph edges are positioned at the maximum distance possible from nearby obstacles. [Campbell et al., 2020]

2.6 Trajectory planning

Trajectory planning is a classical problem within robotics, with a variety of different techniques that have been recorded in the literature. Usually a path is assumed to have been previously generated by a path planner and the optimization carried out by the trajectory planner consists of finding the timing of motion under a number of constraints [Massaro et al., 2023].

The main differentiator between trajectory planning techniques is their optimality criteria.

Common optimality criteria

Time-energy optimal trajectory planning methods aim to balance execution time and energy consumption. Trajectory planning based on energy criteria offers advantages such as smoother trajectories, reduced stresses on actuators and manipulator structures, and energy conservation, crucial for applications with limited energy sources like outer space or underwater exploration [Gasparetto et al., 2015]. These methods usually incorporate constraints on velocity, acceleration, jerk, and input force/torque. This can be seen in the work done by Tokekar et al. [2014] on car-like robots, where they define an energy model to minimize that is based on aforementioned constraints.

Jerk optimal trajectory planning methods aim to limit the the total jerk (defined as the time derivative of acceleration) that a robot experiences. Trajectory planning based on jerk optimality criteria offer advantages such as alleviated stresses on structures and actuators, and reduced tracking errors [Gasparetto et al., 2015]. In studies conducted by Piazzi and Visioli [2000], an algorithm aimed at minimizing the maximum absolute value of jerk along a predefined trajectory is introduced. This technique, known as a minimax approach, imposes constraints on the trajectory execution time while seeking to optimize smoothness. By bounding the trajectory execution time and leveraging cubic splines, this algorithm strives to achieve smoother robotic trajectories, contributing to enhanced performance and reduced stress on manipulator structures and actuators.

Minimum time trajectory planning methods aim to generate a velocity profile that minimizes the time it takes the robot to traverse a certain path. Trajectory planning based on time optimality usually results in the robot accelerating and breaking with great intensity and maintaining high velocities throughout the entire traversal of the path.

For this project, it was decided to focus on creating a Trajectory planning algorithm that minimizes traversal time.

2.7 Minimum time trajectory planning across N-Arcs

To the best of our knowledge, there has been no published research that aims to create a trajectory planner for a mobile robot traversing N-Arcs. There has been some limited research done on trajectory planning with N-Arcs, but this was for agricultural articulated robots that were operating within their own task space [Boryga et al., 2015] (notably these robots are not mobile - do not move). The algorithms proposed by Boryga et al. [2015] ensure continuity in displacement, velocity, and tangential acceleration. It is also relevant to note that there has been work done on path planning for N-Arcs and car-like drive robots, such as that done by Fraichard [1991] (in this research path planning was carried out rather than trajectory planning as there is no time-bound outputs as part of the trajectory).

Okuyama et al. [2021] proposed a real-time minimum-time trajectory planning strategy for differential drive robots traversing waypoint-based parameterized paths. While they claim that their algorithm and optimization procedures are fast enough for real-time usage (trajectory planning takes less than 13.8ms in 95% of test cases when traversing

an area of 1.5m × 1.3m), we believe the runtime of the algorithm could be even faster. In their research, they proposed a novel friction model that takes into account longitudinal and lateral forces on the wheels to prevent slipping. They also proposed a novel optimization strategy based on the Resilient Propagation algorithm. We believe these two novelties could be quite computationally demanding resulting in a longer-than-ideal computation time. Especially is the case given that the proposed optimization strategy has a cost function defined for the entire curve and changes the waypoints of the path to attempt to minimize said cost - this is very time inneficient given that the changes to the waypoints occur at random.

We will take inspiration from the literature while attempting to address any known pitfalls.

Chapter 3

Design and Implementation

3.1 Introduction

To achieve the project's objectives, we adopted a divide-and-conquer approach and segmented the challenge into manageable sub-tasks. Initially, we developed a method for creating a trajectory for a single Arc (the fundamental unit of an N-Arc). Subsequently, we expanded this method to accommodate N-Arcs (multiple consecutive Arcs). Next, we adapted and built on top of the previous system to support custom Splines (functions composed of polynomial segments). Lastly, we integrated a custom version of a Spline smoothing algorithm. The completed system is capable of producing a realistic velocity profile for a differential drive vehicle, striving to generate a profile that is fast and allows the vehicle to accurately follow the path.

This layered approach enabled us to construct a system grounded in fundamental principles, which we then enhanced incrementally to create a solution that is more resilient, more effective, and more comprehensive.

The trajectory planner algorithms described in this section assume that there exists a path planner that has already generated path in the form of a sequence of N-Arcs that the differential drive robot should follow. Furthermore, it is also assumed that the N-Arc path has a defined safety buffer on either side of the path where the robot can move to without entering into collision with nearby obstacles. The output of the trajectory planner algorithms are a list of velocity values for each wheel, equally spaced in time. The reasons for this are explained in later sections.

3.1.1 The differential drive robot

In order to conduct this research we needed to chose a particular differential drive robot, as this choice might then influence design choices at later stages in the completion of the project (such as the derivation of certain constraints).

When comparing differential drive robots, we need to consider a variety of things such as the wheel configuration, wheel material and traction, control methods, velocity/acceleration characteristics and software development environment, among other

things.

The wheel configuration is extremely important as there are some significant differences between wheel configurations. Differential drive robots generally fall into one of three distinct categories: two-wheeled, three-wheeled, and four-wheeled. The differences between these are highlighted by Stefek et al. [2020].

- The two-wheeled design is mechanically simple and has the highest maneuverability, but severely lacks stability.
- The three-wheeled design can have high traction and maneuverability due to the forces being distributed through three points.
- The four-wheeled design has the highest traction and grip of all designs due to there being four points of contact with the surface, but is relatively more uncommon to find and needs powerful motors.

The wheel material is equally as important, as it dictates the traction that is the vehicle has against the surface it is moving on. The amount of traction that the wheels have directly correlates to the maneuverability that the robot has. Having a high maneuverability is paramount to making sure that the robot is capable of following all the commands we generate, such as turning fast around a tight corner (without experiencing wheel slip) [Kim et al., 2012, Kim and Lee, 2013].

Differential drive robots can be controlled using various methods. In open-loop control, the robot executes predefined commands without feedback; in closed-loop control, the vehicle uses feedback from sensors to adjust the robot's behavior; and lastly, in feed-forward control the robot anticipates disturbances and adjusts the control inputs accordingly. Given that we are carrying out research on trajectory planning, we would need to use an open-loop system. Within open-loop systems there are different ways that a robot could be controlled: for instance by providing it with torque commands, by providing it with wheel velocity commands, or by providing it with voltage commands to each motor [Kolter et al., 2010].

The kinematic characteristics of a differential drive robot, such as the acceleration and maximum velocity that the wheels of the robot can sustain, act as constrains on the trajectory that the robot can realistically follow. For instance, if a robot has a maximum acceleration of $3m/s^2$, it cannot be expected for the robot to accelerate from $2m/s$ to $10m/s$ in $1s$ - such a trajectory would be considered unrealizable. The lower the maximum acceleration of a robot is, the more challenging the task for the trajectory planner as the constraint would be tighter.

When designing a trajectory planner, choosing the right robot based on software compatibility is crucial. Since our trajectory planning algorithm should be compatible with other preexisting software such as other path planning, localisation, and mapping algorithms, we would want the robot's software development environment to be one that is commonly used in the industry.

When taking into consideration all of the aforementioned factors, one particular family of robots was chosen to be of particular interest: The family of Turtlebot differential drive robots. These are a series of differential drive robots that are small, affordable

and use a Robot Operating System (ROS) based software stack. These are among the most popular open source robotics platforms for education, research, and product prototyping.

In particular among the Turtlebot family, we opted for the Turtlebot 3 Burger model. This is a platform that is relatively new and has all the important features that are needed for this project [ROBOTIS e-Manual]:

- It has a three-wheel configuration for good traction and high maneuverability (note that one of the wheels is inactive and, given the low resistance to motion of the third inactive wheel, the vehicle dynamics are modeled to behave like a two-wheeled vehicle).
- The tire material is rubber. Their composition ensures good traction on different surfaces, allowing the robot to move effectively.
- The Turtlebot 3 Burger is compatible with different Control architectures. Importantly for this project, it is compatible with an open-loop control architecture that accepts linear velocity and angular velocity as commands.
- In terms of its kinematic characteristics, the Turtlebot 3 Burger has a low maximum linear velocity of $0.22 m/s$ and maximum angular velocity of $2.84 rad/s$, but these limits are not physical constraints and can be increased/removed in the software.
- The robots in the Turtlebot family are all compatible with the Robot Operating System (ROS) framework. ROS is a powerful and versatile software framework that provides essential functionality and tools for developing software for robots.

3.1.2 Path representation

As explained previously in the Background section of the report, there are multiple ways of representing a path. With careful consideration, it was decided that using a geometrical shape representation was best for this project. Specifically, it was chosen to represent it using N-Arcs and it was for the following two main reasons:

The first reason for representing the path with geometric shapes relates to collision with objects. Given that the path planning process should be risk-aware, having the path encoded in a discretized way such as a list of waypoints would lead to possible violations of constraints between two distinct and contiguous waypoints (such as colliding with an obstacle). This is important both for the path itself and for the safety margins on each side of the path.

The second reason for representing the path as an N-Arcs sequence is directly related to the type of robot we will be using - the differential drive robot. When a differential drive robot is moving with a constant velocity on each of its wheels, its traverses the shape of a perfect circle (as long as the wheel speeds are not equal). Thus, since a differential drive robot naturally drives in arcs, it would be natural to represent its path as an N-Arc series.

For the fair comparison between the trajectory planning methods used in this section, we will define a common NArc sequence. This NArc sequence will be used in the diagrams of this section, can be seen in Figure 3.1 and is defined by the following set of consecutive Arcs:

1. Arc of radius 1m, length 1.5m, turning RIGHT.
2. Arc of radius 2m, length 1.5m, turning LEFT.
3. Arc of radius 2m, length 2m, turning RIGHT.
4. Arc of radius 2m, length 3m, turning LEFT.

Also note that in all diagrams in this section, the maximum longitudinal acceleration and deceleration are set to 1.0m/s, the maximum lateral acceleration is set to 2.0m/s and the maximum longitudinal velocity is set to 1.2m/s.

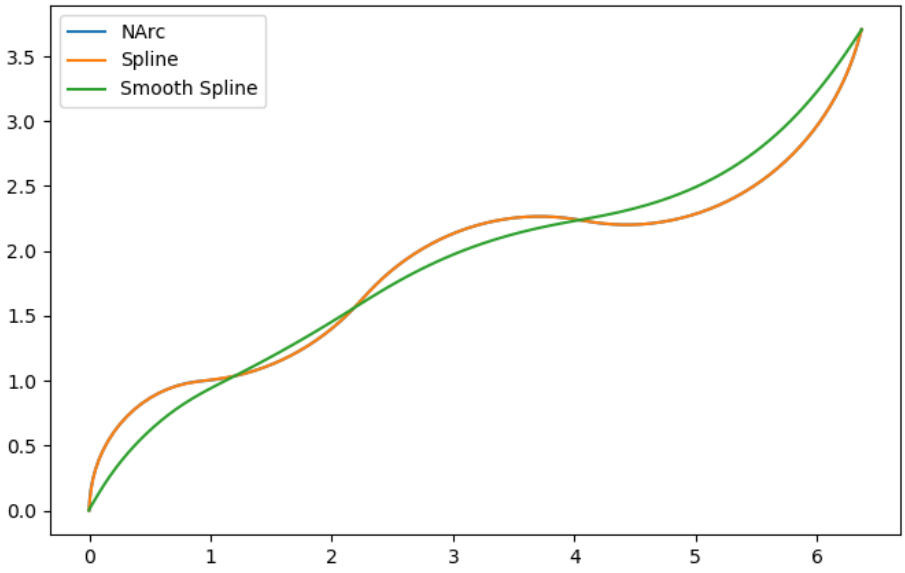

Figure 3.1: Common NArc sequence used to benchmark the methods described in this section

3.2 Arc

To generate a velocity profile for an entire N-Arc sequence, we need to generate a velocity profile for a singular Arc. In further sections we will build on top of the work carried out in this section.

3.2.1 Properties of an Arc

An arc is generally defined as a portion of the circumference of a circle. It is a smooth and continuous curve in a two-dimensional plane. In mathematics an Arc is defined

for all types of curved shapes, but for the purposes of this research we will limit our definition to just circles.

An arc has a constant radius r and length l. The angle within the circle that is traversed by the entire arc can be calculated with $\theta = \frac{l}{r}$ (given that the angle θ is in radians).

3.2.2 Methodology of trajectory planning

The goal with this section is to make the differential drive robot traverse an arc while following a best effort approach to follow the arc accurately (not deviate significantly from the arc) and quickly (traverse the arc in as little time as feasibly possible). We will do this without utilizing the given safety margins.

3.2.2.1 Constraints

Given that the trajectory will be defined as the velocities that each wheel of the differential drive robot should have to traverse the arc, we decided to build our constraints around velocities and accelerations to be more efficient (in terms of runtime) than computing the individual forces (as done in other methods).

First we will build constraints to form an 'upper boundary' on the velocity profile. Several constraints will form the upper boundary, and the lowest of each constraint at each point in the path is taken to be the upper boundary (velocities must not surpass this point).

The first constraint forming this upper boundary will account for the lateral forces that act on the robot and the tires. The reason we apply this constraint is because if a the robot goes too fast around a corner, the tires might lose grip or the robot might topple over. The formula for lateral acceleration is derived from the principals of circular motion and Newton's second law of motion. As the robot travels around a curve it experiences a centripetal force F_c (the force that is directed towards the center of the curve and is responsible for making the vehicle move in a curved path):

$$F_c = \frac{m \cdot v^2}{r}$$

where m is the mass of the vehicle, v is the linear velocity of the vehicle, and r is the radius of the curved path. From this formula, we can combine it with Newton's second law of motion ($F = m \cdot a$) to derive the lateral acceleration a_{lat}:

$$a_{lat} = \frac{v^2}{r}$$

From this lateral acceleration we can then combine it with Newton's third law of motion (centrifugal force that is acted on the wheels) and we can finally derive our second first:

$$v \leq \sqrt{maxA_{lat} \cdot r} \tag{3.1}$$

where $maxA_{lat}$ is the maximum lateral acceleration that can be sustained by the differential drive robot before starting to lose grip due to centrifugal forces, or if it starts to

tip over. This will be again applied to each wheel (thus r will differ slightly between each wheel due to the wheelbase of the robot).

The second constraint forming the upper boundary was introduced after discovering some instabilities during testing. After a certain velocity was reached, the robot would not accelerate in an expected way, sometimes even accelerating more one wheel than another thus leading the differential drive robot to unexpectedly turn to one side. This will be discussed later in the Experimentation chapter. To mitigate this issue, it was decided that the velocity of each wheel should be limited:

$$v \leq maxV_{long} \tag{3.2}$$

where $maxV_{long}$ is the maximum longitudinal velocity that can be sustained by one wheel while still being able to accelerate and decelerate as expected.

By taking the minimum of constraints 3.1 and 3.2 we form the upper boundary of velocities in the velocity profile. Now, we will create variables that enforce acceleration and deceleration constraints and thus are able to calculate the true velocity profile.

The third set of important constraints limit the acceleration of each wheel of the differential drive robot. For any two consecutive points (p_0 and p_1) in the path, the velocities of each wheel must not surpass

$$maxA_{long} \geq \frac{v_1 - v_0}{t}$$

and

$$maxD_{long} \geq \frac{v_0 - v_1}{t}$$

where $maxA_{long}$ is the maximum longitudinal acceleration that can be sustained by one wheel, v_0 and v_1 are the velocities of the wheels on the same side (left or right) at points p_0 and p_1 respectively (p_1 is the point directly after p_0 in the curve), and t is the time taken for the robot to travel from p_0 to p_1.

To eliminate the unknown time variable t, we combine the above constraints with the following formulas: $s = \frac{v_0L+v_0R+v_1L+v_1R}{4} \cdot t$ (derived from the formula for velocity) and $\frac{v_1R}{r_1R} = \frac{v_1L}{r_1L}$ (ratio of velocities of each wheel is proportional to the radius of curvature each wheel is experiencing), where s is the distance between p_0 and p_1, v_0L and v_0R are the left and right wheel velocities at p_0 respectively, and v_1L and v_1R are the left and right wheel velocities at p_1 respectively. The final constraints are as follows:

When turning right ($r_1R \leq r_1L$),

$$v_1L \leq \frac{\sqrt{4 \cdot r_1L \cdot (r_1L + r_1R) \cdot (v_0L^2 + v_0L \cdot v_0R + 4 \cdot s \cdot maxA_{long}) + (r_1R \cdot v_0L - r_1L \cdot v_0R)^2}}{2 \cdot (r_1L + r_1R)} - \frac{r_1L \cdot v_0R + r_1R \cdot v_0L}{2 \cdot (r_1L + r_1R)} \tag{3.3}$$

When turning left ($r_1L \leq r_1R$),

$$v_1R \leq \frac{\sqrt{4 \cdot r_1R \cdot (r_1L + r_1R) \cdot (v_0R^2 + v_0L \cdot v_0R + 4 \cdot s \cdot maxA_{long}) + (r_1L \cdot v_0R - r_1R \cdot v_0L)^2}}{2 \cdot (r_1L + r_1R)} - \frac{r_1R \cdot v_0L + r_1L \cdot v_0R}{2 \cdot (r_1L + r_1R)} \quad (3.4)$$

Note that the full derivation of these constraints can be found in the appendix (appendices A.1 and A.2). The reason for only constraining the left wheel when turning right is because the left wheel's velocity is higher than the right wheel's velocity during a right turn, thus the left wheel's velocity offers a tighter constraint. Similarly, the right wheel's velocity is more tightly constrained when turning left. The velocity of the right and left wheels are bound by the ratio of the radius of curvature at that wheel ($\frac{v_1R}{r_1R} = \frac{v_1L}{r_1L}$); this can be used to calculate the missing wheel's velocity from the above constraints. These constraints are necessary so that traction is maintained and to account for explicit limitations in the control commands given to the robot.

The fourth set of constraints are limits on the deceleration of each wheel of the differential drive robot. These constraints are similar to 3.3 and 3.4 (however, in this case v_0L and v_0R are the unknown variables whereas v_1L and v_1R are given). The full derivation can be found in the appendices A.3 and A.4.

When turning right ($r_0R \leq r_0L$),

$$v_0L \leq \frac{\sqrt{4 \cdot r_0L \cdot (r_0L + r_0R) \cdot (v_1L^2 + v_1L \cdot v_1R + 4 \cdot s \cdot maxD_{long}) + (r_0R \cdot v_1L - r_0L \cdot v_1R)^2}}{2 \cdot (r_0L + r_0R)} - \frac{r_0L \cdot v_1R + r_0R \cdot v_1L}{2 \cdot (r_0L + r_0R)} \quad (3.5)$$

When turning left ($r_0L \leq r_0R$),

$$v_0R \leq \frac{\sqrt{4 \cdot r_0R \cdot (r_0L + r_0R) \cdot (v_1R^2 + v_1L \cdot v_1R + 4 \cdot s \cdot maxD_{long}) + (r_0L \cdot v_1R - r_0R \cdot v_1L)^2}}{2 \cdot (r_0L + r_0R)} - \frac{r_0R \cdot v_1L + r_0L \cdot v_1R}{2 \cdot (r_0L + r_0R)} \quad (3.6)$$

Constraints 3.1, 3.2, 3.3, 3.4, 3.5, and 3.6 are to be applied to each wheel and should not be violated throughout the entire trajectory. Is important to note that $maxA_{long}$, $maxD_{long}$, $maxA_{lat}$, and $maxV_{long}$ will be estimated later during the experimentation phase.

3.2.2.2 Methodology

In order to generate a velocity profile, we first discretize the path into distinct points that can be iterated over. We know that the robot will not enter into collision with any object because at any point in the arc the robot is not at collision with an object (discretized at very small timesteps).

To generate the velocity profile we propose a three step process, where each consecutive layer will aim to reduce the velocity further to stay within the aforementioned constraints.

The first and second steps will set upper boundaries in the velocity for the lateral acceleration (constraint 3.1) and the maximum velocity (constraint 3.2). Given that the radius is constant across the entire arc, the lateral acceleration constraint will be constant and thus both constraints act as upper boundaries in the velocity profile.

The third and fourth steps will make sure that the acceleration and deceleration constraints are not violated (constraints 3.3, 3.4, 3.5, and 3.6).

1. We will first do a forwards pass (iterate through all the points in the arc) to enforce constraints 3.3 and 3.4. In this step, the velocities v_0L and v_0R are used to calculate the velocities v_1L and v_1R that would maximize the acceleration. If the resulting velocity values are lower than the previous velocity values resulting from step 1 and 2, then they are kept.
2. We will then do a backwards pass to enforce constraint 3.5 and 3.6. Similarly to the previous step, the velocities v_1L and v_1R are used to calculate the velocities v_0L and v_0R that would maximize the deceleration. Again, if the resulting velocity values are lower than what was previously calculated, then they are kept.

The four described steps make sure that the differential drive robot drives fast and accurately through the arc.

3.2.3 Implementation insights

It is important to note why we iterate over points in space rather than iterating over points in time. The reason for this is that each of the aforementioned steps modify the velocity of the robot at each point in the curve and thus the time it takes to traverse the curve. It would be infeasible to iterate over time as the time dimension would be constantly changing. However, given that the output of the trajectory planner needs to be the velocity of each wheel in time, after all four steps are carried out, the trajectory is interpolated over time.

In algorithm 1 you can find a simplified pseudocode of our implementation of the code. It is assumed that all of the constants are given.

In practice, this was implemented in Python. Within the implementation, mathematical libraries such as numpy were used to speed up the runtime of the algorithm.

3.3 N-Arc

In this section we will build on top of the work done in the previous section in order to generate a velocity profile for an entire N-Arc sequence. We will do this without utilizing the given safety margins (the robot should follow the N-Arc sequence as accurately as possible).

Algorithm 1 Trajectory planning for Arc

Require: $maxA_{long}, maxD_{long}, maxA_{lat}, maxV_{long}, arc, wheelbase$
  $r \leftarrow arc.radius$
  $rL \leftarrow r + \frac{1}{2} \cdot arc.direction \cdot wheelbase$ ▷ Radius of left wheel
  $rR \leftarrow r - \frac{1}{2} \cdot arc.direction \cdot wheelbase$ ▷ Radius of right wheel
  $profile_1L \leftarrow \sqrt{maxA_{lat} \cdot rL}$
  $profile_1R \leftarrow \sqrt{maxA_{lat} \cdot rR}$
  if $arc.direction = 1$ **then** ▷ If turning right, the left wheel will travel faster
    $profile_2L \leftarrow maxV_{long}$
    $profile_2R \leftarrow profile_2L \cdot \frac{rR}{rL}$
  else
    $profile_2R \leftarrow maxV_{long}$
    $profile_2L \leftarrow profile_2R \cdot \frac{rL}{rR}$
  end if
  $profile_2L \leftarrow minimum(profile_1L, profile_2L)$
  $profile_2R \leftarrow minimum(profile_1R, profile_2R)$
  for $i = 0; i <= len(profile_2) - 2; i++$ **do**
    if $arc.direction = 1$ **then**
      $profile_3L[i] \leftarrow minimum(profile_2L[i], \textbf{CONSTRAINT 3.3})$
      $profile_3R[i] \leftarrow profile_3L[i] \cdot \frac{rR}{rL}$
    else
      $profile_3R[i] \leftarrow minimum(profile_2R[i], \textbf{CONSTRAINT 3.4})$
      $profile_3L[i] \leftarrow profile_3L[i] \cdot \frac{rL}{rR}$
    end if
  end for

  for $i = len(profile_2) - 1; i >= 1; i--$ **do**
    if $arc.direction = 1$ **then**
      $profile_3L[i] \leftarrow minimum(profile_2L[i], \textbf{CONSTRAINT 3.5})$
      $profile_3R[i] \leftarrow profile_3L[i] \cdot \frac{rR}{rL}$
    else
      $profile_3R[i] \leftarrow minimum(profile_2R[i], \textbf{CONSTRAINT 3.6})$
      $profile_3L[i] \leftarrow profile_3L[i] \cdot \frac{rL}{rR}$
    end if
  end for

  $trajectoryL, trajectoryR \leftarrow TimeInterpolate(profile_4L, profile_4R)$
  return $trajectoryL, trajectoryR$

3.3.1 Properties of an N-Arc

An N-Arc is defined as being a consecutive sequence of Arcs. The total length l of the N-Arc sequence is the sum of the lengths of the individual arcs. The radius r at a particular point in the N-Arc sequence depends on the arc that point is located in:

$$Arc(t) = \begin{cases} arc1 & \text{if } 0 \leq t \leq arc1.length, \\ arc2 & \text{if } arc1.length < t \leq arc1.length + arc2.length, \\ ... & \end{cases}$$

where $0 \leq t \leq narc.length$ is the step along the N-Arc sequence. From this definition, one can determine the radius r and the direction of the curve at any point in the N-Arc sequence.

For the purposes of this research we will use N-Arcs that are C^1 continuous. This means that the N-Arc sequences are continuous even if differentiated once, but not necessarily continuous if differentiated more than once. This is showcased in the below diagrams where we can see an N-Arc sequence of two arcs with different radii.

In practice this means that there can exist discontinuity in the acceleration between two consecutive arcs. Specifically, we can observe that this happens when two consecutive arcs have different radii or different directions.

3.3.2 Methodology of Trajectory planning

3.3.2.1 Constraints

The discontinuity of the acceleration between two consecutive arcs of different radii creates a significant problem: the differential drive robot cannot traverse the point that joins both arcs without violating the wheel acceleration and deceleration constraints (3.3, 3.4, 3.5, and 3.6), unless it comes to a complete stop.

The following situation will illustrate the problem; Suppose a differential drive robot (with a wheelbase of 0.2m) is traversing an N-Arc sequence of two Arcs - one turning right with a radius of 1m and another turning left with a radius of 0.5m. When traversing the first arc, the left wheel of the differential drive robot is traversing on an arc with a radius of 1.1m while the right wheel is traversing on an arc of 0.9m in radius. When traversing the second arc, the left wheel traverses an arc with radius 0.4m while the right wheel traverses an arc with radius of 0.6m. At the point where both arcs meet, the differential drive robot is incapable of not violating constraint 3.3, 3.4, 3.5, and 3.6 as it cannot accelerate/decelerate both wheels fast enough, unless it drives to a complete stop at that point.

3.3.2.2 Methodology

Given the aforementioned problem due to the discontinuity of the acceleration across consecutive arcs, we will force the velocity of both wheels to be 0 at the point where

the arcs join. This will make sure that no constraints are violated for the entirety of the generated trajectory.

Since we force the velocity to be 0 at each Arc intersection, we will essentially create a velocity profile for each Arc individually and then combine it to create a velocity profile for the entire N-Arc sequence. For this we will use algorithm 1 as described in the previous section.

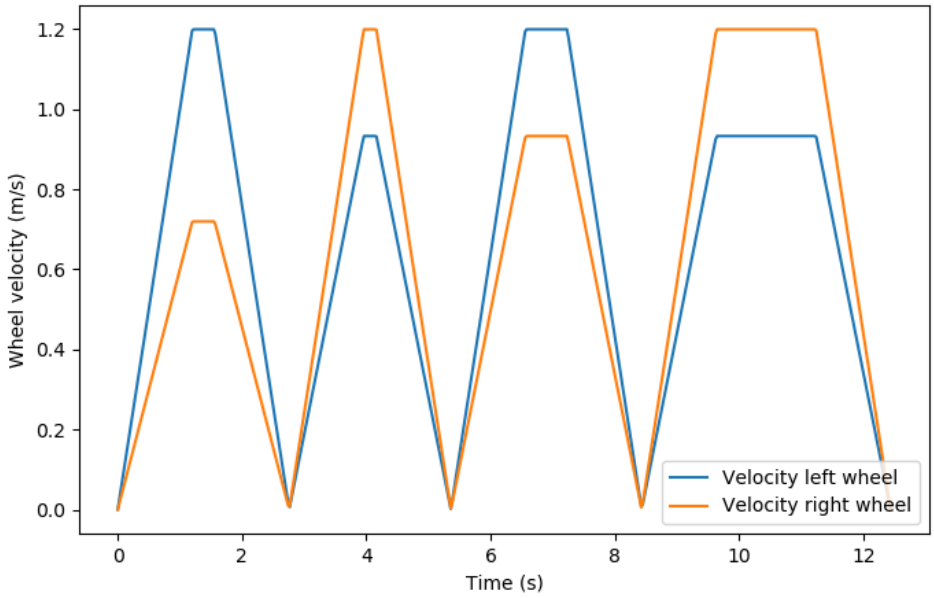

Figure 3.2: Velocity profile created by NArc trajectory planner

3.3.3 Implementation insights

For the implementation, we reused algorithm 1 given the similarities between both solutions. This can be seen in algorithm 2.

Algorithm 2 Trajectory planning for Arc

Require: $maxA_{long}$, $maxD_{long}$, $maxA_{lat}$, $maxV_{long}$, $arcs$, $wheelbase$
  $trajectoryL \leftarrow$ empty list
  $trajectoryR \leftarrow$ empty list
  for arc in $arcs$ **do**
    $profileL, profileR \leftarrow Algorithm1(maxA_{long}, maxD_{long}, maxA_{lat}, maxV_{long}, arc, wheelbase)$
    $trajectoryL \leftarrow trajectoryL + profileL$
    $trajectoryR \leftarrow trajectoryR + profileR$
  end for
  return $trajectoryL, trajectoryR$

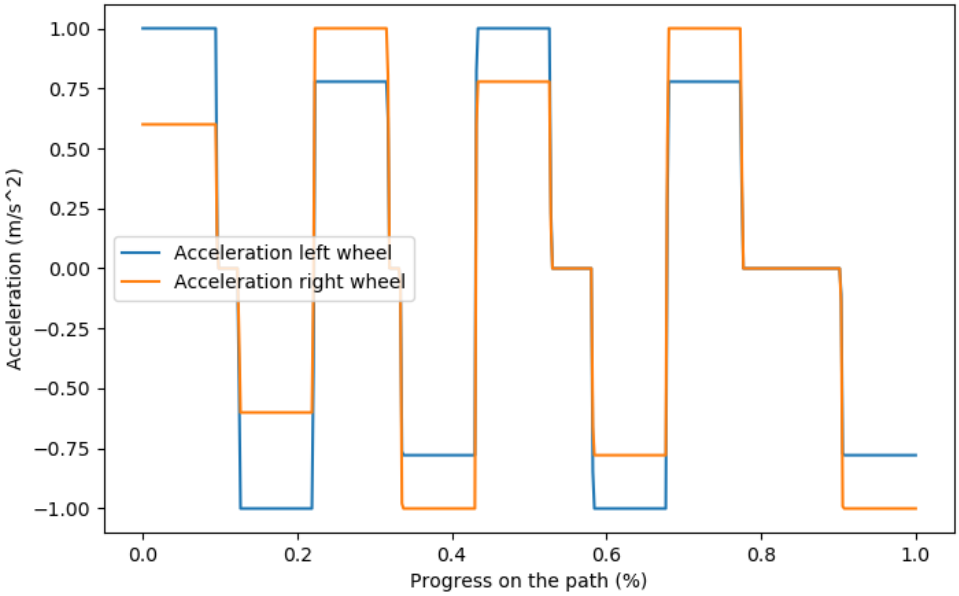

Figure 3.3: Accelerations of velocity profile created by NArc trajectory planner

3.4 Spline

In this section we will address the pitfalls of the previous methodology of generating a trajectory for N-Arcs - specifically we will generate a trajectory that does not require the differential drive robot to come to a complete stop at the intersection of every pair of Arcs in the N-Arc sequence.

For this, it is necessary to modify the path that the robot needs to follow. Particularly, in order not to violate the wheel acceleration/deceleration constraints, we need to modify the path so that it becomes C^2 continuous. At this point, we will start using the safety boundaries at each side of the original N-Arc sequence.

3.4.1 Properties of a Cubic Spline

As previously discussed, N-Arc sequences are only C^1 continuous. An N-Arc path can only be C^2 continuous if and only if all the Arcs in the sequence have the same radius of curvature and turn to the same direction. Therefore, given that we are looking for our path to have C^2 continuity, we need to change our path representation from now on - from the originally given N-Arc sequence to a new representation.

When exploring different new ways to represent the given path, we considered:

1. Similarity to the original N-Arc sequence in terms of path location (deviation from its original path) and smoothness.
2. Computational efficiency when converting the N-Arc sequence to this new representation and when calculating all the points in the path, along with their first and second derivatives.

3. ”Malleability” in terms of how computationally efficient it is to slightly alter the location of sections of the path (this will become useful when attempting to make the trajectory more time efficient - done in the next section).

Ultimately, it was decided to represent the path as a continuous Cubic Spline. A Cubic Spline is a parameterized smooth curve that is defined piece-wise by polynomials. Crucially for this research, a Cubic Spline is C^2 continuous. A Cubic Spline is created from the coordinates of points that should be intersected by the curve.

1. A Cubic Spline is smooth in nature and can very similarly approximate the original N-Arc sequence.
2. Given the piece-wise parameterization of Cubic Splines, it is very time efficient to calculate all points in the path as well as the first and second derivatives.
3. It is computationally efficient to alter the locations of sections of the path - this is done by just modifying the location of the coordinates of a point along the curve.

3.4.2 Methodology of Trajectory Planning

3.4.2.1 Constraints

The trajectory will need to be subject to the same constraints as with previous steps. For this we will directly reuse constraint 3.1 for the lateral acceleration, 3.2 for the longitudinal velocity, 3.3 and 3.4 for the longitudinal acceleration, 3.5 and 3.6 for the longitudinal deceleration.

However, an additional set of constraints is needed to account for the possible change change in the curve's radius between two distinct points (this situation did not arise with Arcs, for instance, as they have a constant radius of curvature). Specifically, the simultaneous acceleration of one wheel and deceleration of another wheel (resulting from a change in radius) is not accounted for up until this point. The derivation for this can be found in appendix A.5.

If:

$$0 \leq s \cdot {r_1}^2 \cdot {r_0}^2 \cdot (r_1L \cdot r_0R - r_1R \cdot r_0L) \cdot (maxA_{long} \cdot (r_1 \cdot r_0R + r_0 \cdot r_1R) + maxD_{long} \cdot (r_1 \cdot r_0L + r_0 \cdot r_1L))$$

Then:

$$v_0 \leq \frac{\sqrt{2} \cdot s \cdot r_1 \cdot {r_0}^2 \cdot (r_1L \cdot maxD_{long} + r_1R \cdot maxA_{long})}{\sqrt{s \cdot {r_1}^2 \cdot {r_0}^2 \cdot (r_1L \cdot r_0R - r_1R \cdot r_0L) \cdot (maxA_{long} \cdot (r_1 \cdot r_0R + r_0 \cdot r_1R) + maxD_{long} \cdot (r_1 \cdot r_0L + r_0 \cdot r_1L))}} \tag{3.7}$$

$$v_1 \leq \frac{\sqrt{2} \cdot s \cdot {r_1}^2 \cdot r_0 \cdot (r_0L \cdot maxD_{long} + r_0R \cdot maxA_{long})}{\sqrt{s \cdot {r_1}^2 \cdot {r_0}^2 \cdot (r_1L \cdot r_0R - r_1R \cdot r_0L) \cdot (maxA_{long} \cdot (r_1 \cdot r_0R + r_0 \cdot r_1R) + maxD_{long} \cdot (r_1 \cdot r_0L + r_0 \cdot r_1L))}} \tag{3.8}$$

Else:

$$v_0 \leq \frac{\sqrt{2} \cdot s \cdot r_1 \cdot {r_0}^2 \cdot (r_1 L \cdot maxA_{long} + r_1 R \cdot maxD_{long})}{-s \cdot {r_1}^2 \cdot {r_0}^2 \cdot (r_1 L \cdot r_0 R - r_1 R \cdot r_0 L) \cdot (maxA_{long} \cdot (r_1 \cdot r_0 L + r_0 \cdot r_1 L) + maxD_{long} \cdot (r_1 \cdot r_0 R + r_0 \cdot r_1 R))} \tag{3.9}$$

$$v_1 \leq \frac{\sqrt{2} \cdot s \cdot {r_1}^2 \cdot r_0 \cdot (r_0 L \cdot maxA_{long} + r_0 R \cdot maxD_{long})}{-s \cdot {r_1}^2 \cdot {r_0}^2 \cdot (r_1 L \cdot r_0 R - r_1 R \cdot r_0 L) \cdot (maxA_{long} \cdot (r_1 \cdot r_0 L + r_0 \cdot r_1 L) + maxD_{long} \cdot (r_1 \cdot r_0 R + r_0 \cdot r_1 R))} \tag{3.10}$$

Therefore, now the upper boundary is formed by constraints 3.1, 3.2, 3.7, 3.8, 3.9, 3.10, while the actual trajectory generation is done by constraints 3.3, 3.4, 3.5, and 3.6.

3.4.2.2 Methodology

Before generating the velocity profile we need to convert the N-Arc sequence into a Cubic Spline. Since a Cubic Spline is created from a set of way-points, we will need to sample the N-Arc sequence (take a point every certain number of metres) in order to create the Spline. The closer the points are sampled to one another, the greater the resemblance will be between the original N-Arc sequence and the newly created Spline.

To generate the velocity profile, we will follow a similar approach as we did for a singular Arc, with some modifications to account for the now not-constant radius and the addition of a step within the creation of the upper ceiling to accommodate constraints 3.7, 3.8, 3.9, and 3.10.

The first and second steps will set upper boundaries in the velocity due to limits on the lateral acceleration (constraint 3.1) and the maximum longitudinal velocity (constraint 3.2). Given that the radius is not constant across the entire curve, the lateral acceleration constraint will need to be computed for every point in the curve. This means that for one particular point in the curve, both of the aforementioned constraints might differ (at some points in the curve constraint 3.1 might be more restrictive than constraint 3.2 or vice-versa).

The third step will consider the newly added constraints (3.7, 3.8, 3.9, and 3.10). In this step, we would iterate over all pairs of consecutive points in the curve and calculate v_0 and v_1 such that they are maximized within the constraints. Up to this point we have created the upper ceiling.

The fourth and fifth steps will make sure that the acceleration and deceleration constraints are not violated (constraints 3.3, 3.4, 3.5, and 3.6). For this we will use the same logic as for the third and fourth steps for the Arc.

The five described steps make sure that the differential drive robot drives fast and accurately through the a curve that has changing radius of curvature.

It is important to note why we carry out steps 1-3 strictly before steps 4 and 5. The reason for this is that steps 1-3 create upper boundaries for the velocities of the wheels at each point in the curve, however, the velocity change between consecutive points in

the curve might still violate the longitudinal acceleration and deceleration constraints. Therefore, it is necessary to complete step 4 and 5 after steps 1-3 to make sure that the trajectory can realistically be followed by the differential drive robot.

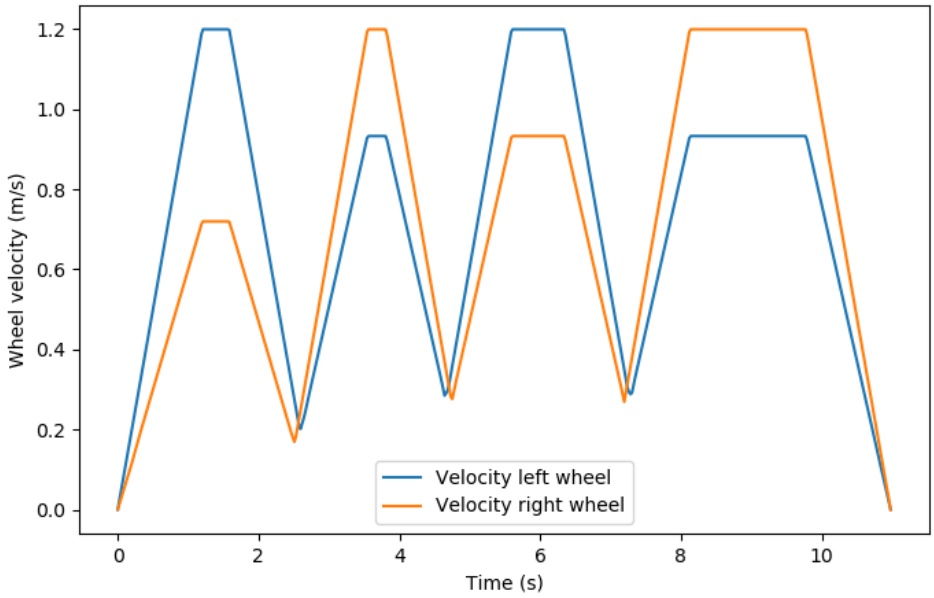

Figure 3.4: Velocity profile created by Spline trajectory planner

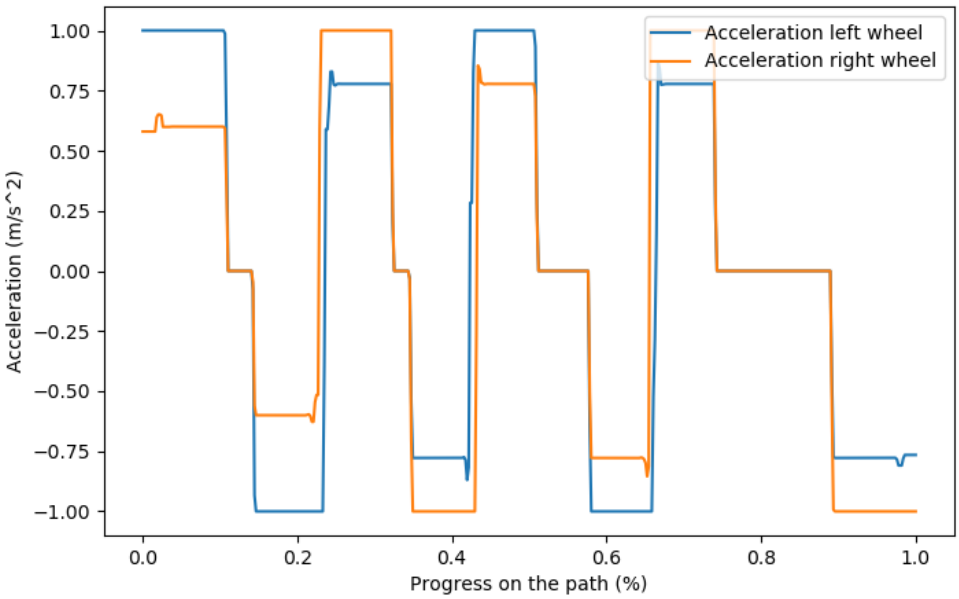

Figure 3.5: Accelerations of velocity profile created by Spline trajectory planner

3.4.3 Implementation insights

In algorithm 3 you can find a simplified pseudocode of our implementation of the code. It is assumed that all of the constants are given.

In practice, this was implemented in Python. Within the implementation, mathematical libraries such as numpy were used to speed up the execution of mathematical calculations along an entire array of data.

Creating the Spline from an N-Arc is very time efficient thanks to the use of the library ”CubicSmoothingSpline” from CSAPS. This package implements Cubic Spline algorithm proposed by Carl de Boor in his book “A Practical Guide to Splines”.

3.5 Smoothed Spline

In this section we will address the inefficiencies in the trajectory that were still present in the previous methodology of generating a trajectory for Cubic Splines. In particular, we will attempt to generate a trajectory that does not make the differential drive robot slow down significantly at each Arc intersection. For this we will utilize as much of the safety buffers as necessary.

3.5.1 Properties of a Smooth Spline

As seen in the curvature graph of the previous section, the Cubic Spline created from an N-Arc still has a rapid change in the curvature of the path at the intersection between arcs of different radius or direction (very pronounced change in curvature in this last case). This very rapid change forces the differential drive robot to slow down significantly in order to still be complaint with the longitudinal wheel acceleration/deceleration constraints.

To minimize the rate of change of the radius/curvature along all points in the curve, it is necessary to modify and straighten (smoothen) the curve. In order to do this we will utilize the safety margins. By straightening the curve we also achieve another positive effect: minimizing the total length of the curve. The combination of having a curve that is more straight and less long should make it possible for the curve to be traversed faster by the differential drive robot.

The problem of smoothing a curve is not new and has been extensively studied in literature. We chose to implement the Convex Elastic Smoothing (CES) algorithm created by Zhu et al. [2015]. This is an algorithm that was originally intended to smoothen the paths generated from sampling-based path planners and to be used with car-like robots. This heuristic algorithm was chosen for its speed (hundreds of miliseconds) and simplicity over other algorithms. The algorithm literately performs shape and speed optimization.

Algorithm 3 Trajectory planning for Cubic Spline

Require: $maxA_{long}, maxD_{long}, maxA_{lat}, maxV_{long}, arc, wheelbase$
  $r \leftarrow arc.radius$
  $rL \leftarrow r + \frac{1}{2} \cdot arc.direction \cdot wheelbase$ ▷ Radius of left wheel
  $rR \leftarrow r - \frac{1}{2} \cdot arc.direction \cdot wheelbase$ ▷ Radius of right wheel
  $profile_1L \leftarrow \sqrt{maxA_{lat} \cdot rL}$
  $profile_1R \leftarrow \sqrt{maxA_{lat} \cdot rR}$
  if $arc.direction = 1$ **then** ▷ If turning right, the left wheel will travel faster
    $profile_2L \leftarrow maxV_{long}$
    $profile_2R \leftarrow profile_2L \cdot \frac{rR}{rL}$
  else
    $profile_2R \leftarrow maxV_{long}$
    $profile_2L \leftarrow profile_2R \cdot \frac{rL}{rR}$
  end if
  $profile_2L \leftarrow minimum(profile_1L, profile_2L)$
  $profile_2R \leftarrow minimum(profile_1R, profile_2R)$
  for $i = 0; i <= len(profile_2) - 2; i++$ **do**
    if $arc.direction = 1$ **then**
      $profile_3L[i] \leftarrow minimum(profile_2L[i], \textbf{CONSTRAINT 3.3})$
      $profile_3R[i] \leftarrow profile_3L[i] \cdot \frac{rR}{rL}$
    else
      $profile_3R[i] \leftarrow minimum(profile_2R[i], \textbf{CONSTRAINT 3.4})$
      $profile_3L[i] \leftarrow profile_3L[i] \cdot \frac{rL}{rR}$
    end if
  end for

  for $i = len(profile_2) - 1; i >= 1; i--$ **do**
    if $arc.direction = 1$ **then**
      $profile_3L[i] \leftarrow minimum(profile_2L[i], \textbf{CONSTRAINT 3.5})$
      $profile_3R[i] \leftarrow profile_3L[i] \cdot \frac{rR}{rL}$
    else
      $profile_3R[i] \leftarrow minimum(profile_2R[i], \textbf{CONSTRAINT 3.6})$
      $profile_3L[i] \leftarrow profile_3L[i] \cdot \frac{rL}{rR}$
    end if
  end for

  $trajectoryL, trajectoryR \leftarrow TimeInterpolate(profile_4L, profile_4R)$
  return $trajectoryL, trajectoryR$

Shape optimization with the CES algorithm

The best way to visualize how the shape optimization aspect of the CES algorithm is to view the curve as a elastic band whose starting and ending points are fixed. The elastic band attempts to contract itself and straightens itself in the process.

Points along the curve are modeled to be inside ”collision-free bubbles” and are subject to artificial forces. There are two types of artifical forces that are applied to the points in the curve:

1. Tensile forces connect adjacent points in the curve and attempt to pull them closer together, hence straightening the curve.
2. Balancing forces connect points in the smoothed curve to the respective points in the original curve and attempts to pull them closer together, hence making sure the optimized curve does not deviate too much from the original curve.

The weighted sum of the aforementioned forces is then applied to each point (this is adjusted through a set of coefficients that are analogous to spring constants in Hooke’s Law). Between iterations of the CES algorithm, these forces move the points that make up the curve.

Modifications to the CES algorithm

In my implementation I made several modifications to the CES algorithm to better suit the needs to this project:

1. The CES algorithm literately performs shape and speed optimization, but we chose to only implement the shape optimization aspect of the algorithm (the speed optimization is not compatible with our approach since the trajectory planning part is to be executed after the path smoothing process has been completed).
2. The collision-avoidance features were maintained but instead of modelling the collision-free space as bubbles around original points, it was modelled to be a tunnel around the original curve. The width of this ”tunnel” is constant along the entire curve to take into account the width of the robot and mimic the safety boundaries. This is the safety buffer from the original N-Arc sequence.
3. After the forces were applied to each point in the curve and a new point was calculated, the new point was projected to be in the normal line to the original curve at the original point. This is done to ensure that the aforementioned collision-avoidance features worked as intended.
4. The running of the algorithm was halted once the sum of the forces on the curve was lower than a pre-defined threshold. This should speed up the algorithm by eliminating later iterations that brought largely insignificant changes to the curve.

After running our tweaked version of the CES algorithm, the resulting curve should be straighter, have less intense changes in curvature, and be shorter length.

3.5.2 Methodology of Trajectory Planning

Given that the trajectory planning is to still be done on a Spline, it was decided to run the same algorithm as the one used for a normals Spline. However, given the improvements to the curve that is fed to the trajectory planning algorithm, the new velocity profile should not make the differential drive robot slow down as much at the intersection of arcs and should be faster given the shorter length of the curve.

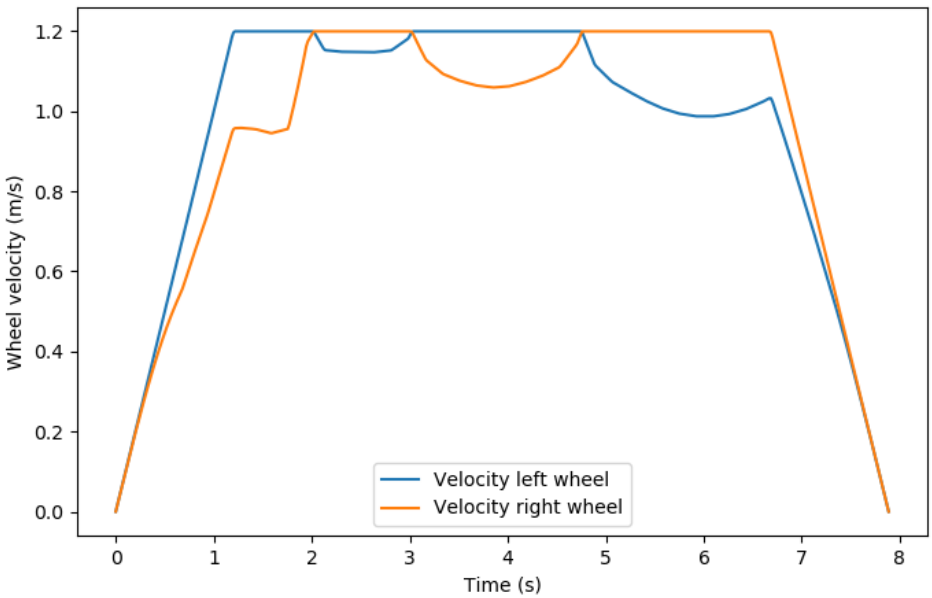

Figure 3.6: Velocity profile created by Smooth Spline trajectory planner

3.5.3 Implementation insights

The curve smoothing algorithm was also implemented in Python and relied on numpy for the quick and efficient numerical computations that were needed.

Algorithm 4 Optimization Algorithm

Require: $originalCurve$, $transforms$, $SAFETY\_MARGIN$, $SPRING\_CONSTANT$, $DAMPING\_CONSTANT$, MAX_ITERS, $THRESHOLD$, arc

$points \leftarrow originalCurve$
$optimizedCurve \leftarrow \text{Copy}(points)$
for $iter$ **in** $range(MAX\_ITERS)$ **do**
    Variables used:
    $prevPoints \leftarrow optimizedCurve[:-2]$ ▷ Previous points in optimized curve
    $currPoints \leftarrow optimizedCurve[1:-1]$ ▷ Current points in optimized curve
    $nextPoints \leftarrow optimizedCurve[2:]$ ▷ Next points in optimized curve
    $originalPoints \leftarrow points[1:-1]$ ▷ Original points in non-altered curve
    $normals \leftarrow transforms[1:-1]$ ▷ normals to non-altered curve
    Calculate forces:
    $prevForces \leftarrow SPRING\_CONSTANT \times (prevPoints - currPoints)$
    $nextForces \leftarrow SPRING\_CONSTANT \times (nextPoints - currPoints)$
    $dampForces \leftarrow DAMPING\_CONSTANT \times (originalPoints - currPoints)$
    $totalForces \leftarrow prevForces + nextForces + dampForces$
    $calculatedForces \leftarrow \text{Dot}(totalForces, normals)$
    Restrict forces:
    $newPoint \leftarrow currPoints + \text{Dot}(calculatedForces, normals)$
    $factor \leftarrow \frac{SAFETY\_MARGIN - \text{Norm}(currPoints - originalPoints)}{\text{Norm}(newPoint - originalPoints)}$
    $factor[factor > 1] \leftarrow 1$
    $adjustedForces \leftarrow 0$ **If** $(factor < 0)$ **Else** $(calculatedForces \times factor)$
    Concatenate forces:
    $forces \leftarrow [0] + adjustedForces + [0]$ ▷ First and last points are not modified
    $optimizedCurve \leftarrow optimizedCurve + \text{Dot}(forces, normals)$
    if $(\text{Sum}(\text{Norm}(forces)) < THRESHOLD)$ **then**
        break For
    end if
end for

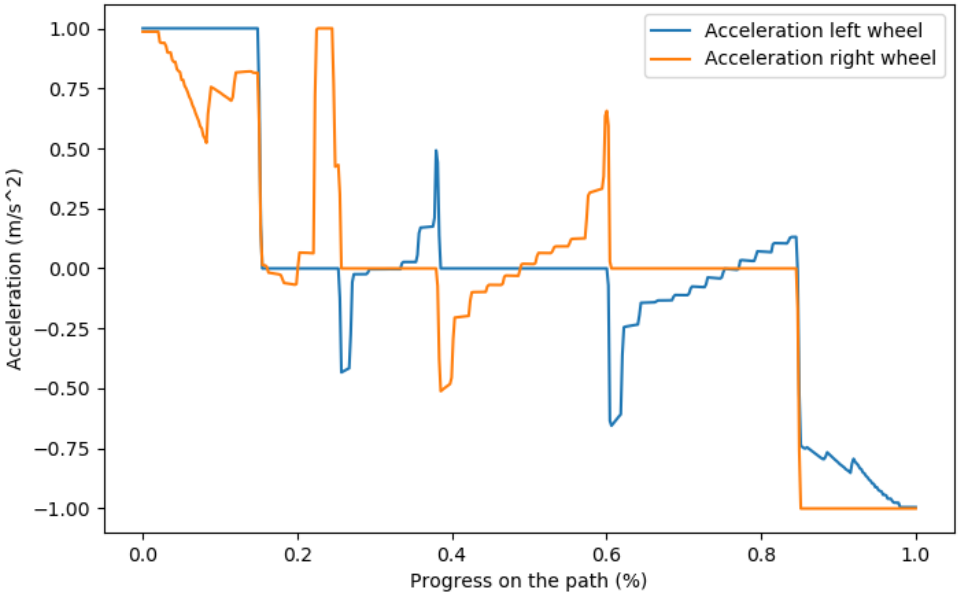

Figure 3.7: Accelerations of velocity profile created by Smooth Spline trajectory planner

Chapter 4

Experimentation

In this section, we conduct experiments to benchmark our trajectory planning algorithms and also offer a point of comparison to other trajectory planning algorithms in the literature.

Choice of simulation environment

In order to carry out experimentation with a virtual Turtlebot 3 Burger, it is necessary to select a simulation environment. When chosing the the simulation environment we considered the following set of characteristics:

1. Features and capabilities: Since we are carrying out research it is necessary that the chosen environment has a specific suite of features that cater our needs. The Physics engine should be able to accurately model the robot's dynamics. The actuators (in our case we are interested in the independent wheel speeds) on the robot should be able to be controlled in a realistic manner. The simulation environment should have extensive integration with other tools (such as ROS, Python, C++, etc...) for ease of use. Customizability is also an important factor, as it might be necessary to alter the default behaviour of the robot or the environment.
2. Performance and speed: The simulation environment should be able to run on our personal computers as we unfortunately do not have access to other hardware tools.
3. Community and support: It is indispensable for the chosen environment to have a strong history of support both in terms of software (bug fixes) and knowledge share (forums, documentation, etc...) to speed up the rate of testing.

Taking into consideration the above characteristics, we narrowed down our selection to Gazebo, PyBullet, and Webots: all of these are robotics software simulations that are capable of accurately modelling the physics of the robot in its environment, have good performance, and are used in the world of research.

Ultimately, the open-source robot simulator Gazebo was chosen as the simulation environment to use for testing our research. The reason for this was due to its extensive

set of features (accurate physics modelling, realistic actuator behaviour, integration with other tools, customizability) and its large community and documentation base.

Conveniently, there exists an open source Turtlebot 3 Burger model that is compatible with ROS and Gazebo. By using ROS with the Turtlebot 3 Burger model, we can control the robot by issuing linear velocity (in m/s) and angular velocity (in rad/s) commands to control the robot.

Note that we are using Gazebo version 11.11.0 and ROS 2 Galactic.

Integration with the simulation

The trajectory planning software we created with this research is not completely compatible out-of-the box with the ROS/Gazebo software stack. The main reason for this is due to the fact that the virtual version of the robot accepts commands in linear velocity and angular velocity, whereas the software we want to test creates commands for individual wheel speeds (wheel speeds for the left wheel and wheel speeds for the right wheel).

Fortunately, the conversion between both modes of representation is straightforward with the following formulae:

$$v = \frac{v_R + v_L}{2}$$

$$\omega = \frac{v_R - v_L}{wb}$$

where v and ω are the linear and angular velocities respectively, v_R and v_L are the linear velocities of each wheel (right and left respectively), and wb is the wheelbase of the robot (distance between both wheels).

To test if the Turtlebot 3 Burger is capable of following our produced velocity profiles, we publish control commands at a constant frequency and record the location of the robot in the simulation. The frequency command publishing is chosen to be high enough so that the motion of the robot appears to be smooth and in line with the planned path.

4.1 Estimation of parameters

In order to run the trajectory planning software that we created throughout this project, it is necessary have the constants that were defined in the constraints. Specifically, these are the maximum longitudinal acceleration $maxA_{long}$, the maximum longitudinal deceleration $maxD_{long}$, the maximum lateral acceleration $maxA_{lat}$, and the maximum longitudinal velocity $maxV_{long}$. Some but not all of these constants depend on the friction that the differential drive robot has on the surface it is riding on.

Given that the Turtlebot specification does not offer any guidance on these constants, we had to estimate them ourselves by using the simulation to test the limits of the car.

The maximum longitudinal acceleration was estimated by repeatedly issuing commands of very high linear velocity to the robot and measuring how it performed. Some of the tests were carried out with the robot at initial stanstill (before accelerating) while other tests were carried out with the robot already in motion. It was concluded that the

Turtlebot 3 Burger has a hard limit of $1.0m/s^2$ but is unstable at higher accelerations (wheels accelerate at different rates and thus the robot is incapable of following the straight-line intended path). The Turtlebot 3 Burger robot behaved as expected when it was issued commands that were no higher than $0.4m/s^2$ in longitudinal acceleration.

The maximum longitudinal deceleration was estimated using a very similar methodology. The robot initially started from a certain linear velocity and was instructed to quickly slow down. Similar to the longitudinal acceleration, the simulation has a hard limit of $1.0m/s^2$ of deceleration but the wheels do not decelerate at expected rates if the longitudinal deceleration is higher than $0.4m/s^2$.

To calculate the maximum lateral acceleration, we relied on the lateral acceleration formula that was defined in an earlier section of the report.

$$a_{lat} = \frac{v^2}{r}$$

The robot was issued commands of constant linear velocity and the radius of curvature was iteratively reduced until the robot stopped following the intended curve - the velocity and radius values where recorded. This was repeated several times and with different linear velocities. The maximum lateral acceleration was then computed by plotting all the results of this experiment and calculating the gradient ($\frac{v^2}{r}$). The calculated maximum lateral acceleration was around $2.0m/s^2$.

The Turtlebot 3 Burger specifications dictate that the robot is not capable of going faster than $0.22m/s$ in longitudinal velocity, but we deemed this value to be too low to showcase the true potential of the trajectory planning software we created throughout this project. It was deemed from the previous experiments that the robot behaved erratically when its linear velocity exceeded $1.2m/s$. This value was used as the new upper limit for the longitudinal velocity.

Note that all of these were deemed as limits ($maxA_{long} = 0.4m/s^2$, $maxD_{long} = 0.4m/s^2$, $maxA_{lat} = 2.0m/s^2$, $maxV_{long} = 1.2m/s$) as the robot behaved erratically when they were surpassed (robot would not maintain the expected linear or angular velocities).

4.2 Methodology of testing and experiments

The problem space of a trajectory planner defined by the possible states of an NArc sequence. An NArc sequence can adopt an infinite number of states: there can be any number of narcs, with any length, any radius, etc... In fact, the state of an NArc sequence is in the subspace of $\mathbb{R}^5$ (number of arcs, arc lengths, arc radii, arc directions, safety buffer size). Given this high dimensionality, it is very hard for us to accurately analyze the performance of our trajectory planner solutions.

Due to the high dimensionality of the possible problem space, we decided to utilize the Monte Carlo approach to obtain metrics about how our algorithms perform. This also brings our testing methods in line with other research so it will allow us to compare results more easily.

In Monte Carlo simulations the problem space is sampled randomly and subsequently the software to test is run. Results are gathered for each iteration and they are used to create a distribution of some informative metrics. The NArc problem space was randomized to its fullest, while adhering to some sensible limits:

1. The length of the NArc sequence was randomized but limited to be $3 \leq narcLength \leq 7$. This limit was chosen because, when considering that ultimately the trajectory planner will generate a trajectory that will only be obeyed at most a couple of seconds, a path length of longer than 7m would result in a really long trajectory in terms of physical time taken to traverse by the robot. Note that the arc length was limited to be an integer for easier comparison in the results stage.
2. The number of Arcs present in the NArc sequence was randomized but limited to $1 \leq nArcs \leq 8$. This limit was introduced so that, given the limit on the length of the entire Narc sequence, there wouldn't be too many short arcs or long arcs.
3. The safety buffer of the NArc sequence was randomized but limited to be $wheelbase \leq bufferSize \leq 2.5 \cdot wheelbase$ so that there was a minimum buffer of half the wheelbase of the car and at mose two wheelbases (consider that the robot is as wide as its wheelbase).
4. The lengths of the individual Arcs within the NArc sequence were randomized by first sampling the space $0 < point < narcLength$ a number of $nArcs - 1$ times (these will be used as arc intersection points), then sorting the samples, and finally taking the arc lengths as the differences of these samples.
5. The radius of the individual Arcs within the Narcs sequence were randomized but limited to $\frac{wheelbase}{2} < radius < 20 \cdot wheelbase$.
6. The direction of each Arc was randomized by simply randomly choosing between $LEFT$ and $RIGHT$.

It was very important to maintain randomness when sampling the problem space, but it was equally as important to limit the scope of the tested problem space to be realistic to avoid unnecessarily testing unrealistic scenarios. Therefore the above limits were introduced.

We took 5000 samples from the problem space to create random NArcs, which were then used to benchmark our trajectory planning algorithms. Comparisons and conclusions were drawn from the algorithms' performance within the simulation as well as outside of it.

4.3 Results

The trajectory planning methods compared in below experiments are:

1. NArc curve with NArc trajectory planner (stop at intersection of arcs).
2. Spline curve with Spline trajectory planner.
3. Smoothed spline curve with Spline trajectory planner.

Firstly, each of the randomized NArcs (problem space) was used to generate velocity profiles using all three of aforementioned trajectory planning methods. The results from these include runtime comparisons, as well as trajectory time length and path length comparisons. We will call these the ”algorithmic results”.

Then, a subset of the generated velocity profiles was used to test the feasibility of the trajectory planning methods in the simulation. We will refer to these as the ”simulation results”.

4.3.1 Algorithmic results

These are the results of our trajectory planning algorithms before being tested in the ROS2 Gazebo simulation. These offer a point of comparison between trajectory planning methods.

Curve improvements analysis

We first compared the differences in curvature between the Spline and the Smooth Spline curves. The results can be seen in Figure 4.1. Note that the length-normalized curvature was calculated by computing the area under the curve of curvature against point in the path (integrated) and was subsequently divided by the total length of the path for comparison.

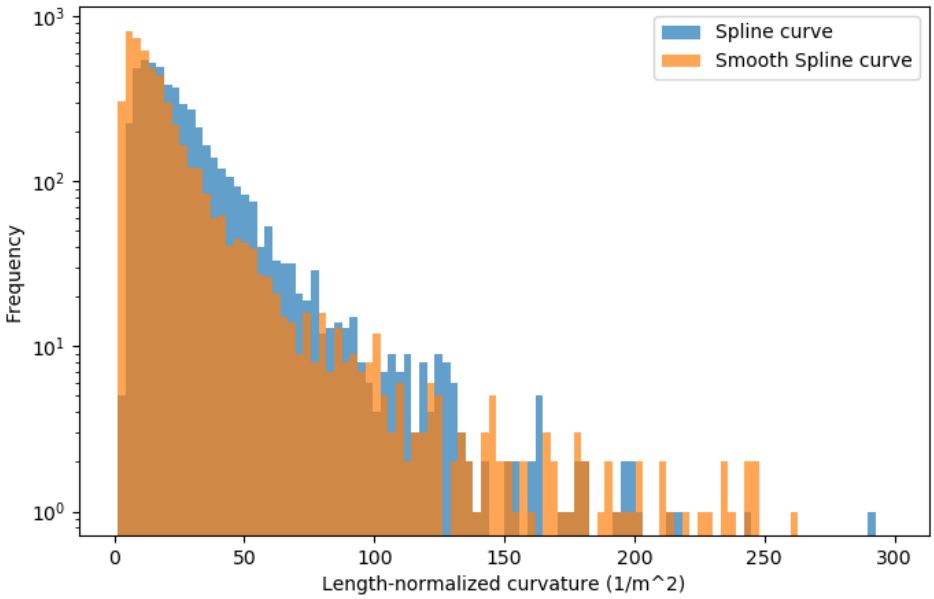

Figure 4.1: Curvature comparison

We can observe that the curvature of Spline curves is generally slightly higher than the curvature of the Smooth Spline curves. In fact the average curvature of Spline curves was $29.37m^{-2}$ whereas the average curvature of Smooth Spline curves was $22.53m^{-2}$ (note how a smaller curvature value means a straighter curve). This suggests that our

Table 4.1: Traversal Time Statistics

	Mean traversal (s)	Standard deviation traversal (s)	Slowest traversal (s)
NArc trajectory planner	8.90	2.51	15.82
Spline trajectory planner	6.87	1.54	12.87
Smoothed spline trajectory planner	5.47	1.17	7.60

modified implementation of the CES algorithm works as intended: the curvature and rapid changes in curvature are reduced.

On average, 2.63 ms to create a Spline from a NArc sequence. Interestingly our modified implementation of the CES algorithm was much faster than the original: it took 3.03 ms to Smoothen a Spline (average of 74.07 iterations per run). The original CES algorithm ran within 13.8 ms ms in 95% of cases where as our implementation only took 5.03 ms in 95% of cases.

Trajectory traversal time analysis

We compared the traversal time (time taken for the robot to travel along the entire trajectory) between trajectory planning techniques. This can be seen in Figure 4.2 and summarised in Table 4.1.

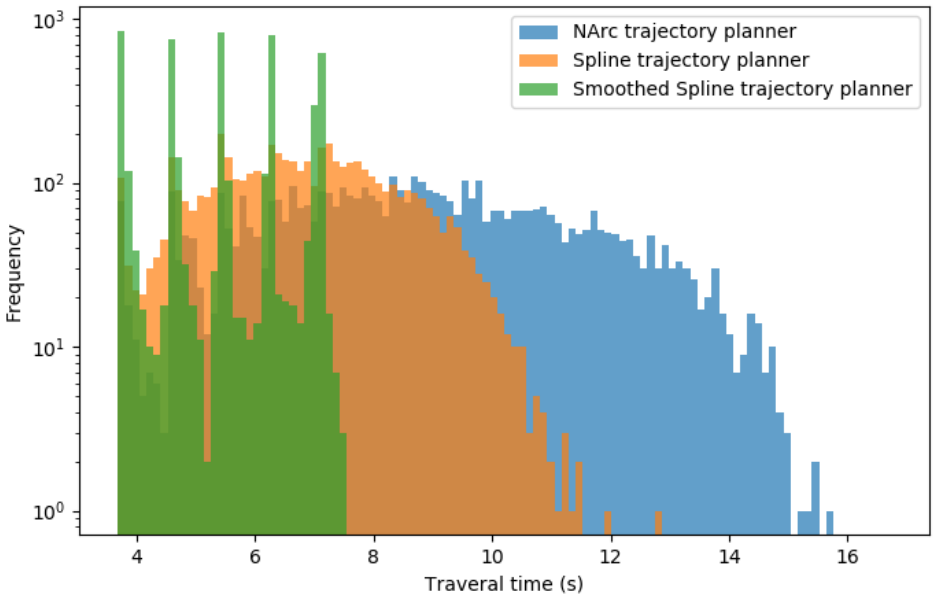

Figure 4.2: Traversal time analysis of trajectories

From the figure and and the table it can be clearly seen that the Smoothed Spline trajectory planner consistently creates the fastest trajectory and the NArc trajectory planner creates the slowest trajectory. In the worst-case scenario (slowest recorded

Table 4.2: Runtime Statistics

	Mean runtime (ms)	Standard deviation runtime (ms)
NArc trajectory planner	6.88	3.71
Spline trajectory planner	3.82	0.65
Smoothed spline trajectory planner	3.74	0.64

traversal) the Spline planner creates a trajectory that is 18.6% faster than the NArc alternative, where as the Smooth Spline trajectory is 52.0% faster. The fastest traversal is the same across all three planners (all within 0.01s of 3.75s).

We believe that the improvements in traversal time of the Spline planners - especially the Smoothed Spline planner - is due to the much decreased curvature and minimized changes in curvature that these have. When a path has a high curvature, especially if there are rapid changes in curvature such as in an NArc sequence, the differential drive robot has to slow down so that no the constraints are violated (especially limiting are the longitudinal acceleration and deceleration constraints).

It is interesting to note that there are there are five peaks in the traversal time of the Smoothed Spline. These peaks align with the 5 different NArc path lengths that the curve can have. This is a by-product of our sampling of the problem space (if we sampled floating point NArc lengths they would not appear). Given that these peaks appear at the Smooth Spline Trajectories and not at the other trajectories it suggests that the robot is traveling close to its maximum longitudinal velocity - due to the straighter curve.

Trajectory planning runtime analysis

First we compared the runtime between trajectory planning techniques. This can be seen in Figure 4.3 and summarised in Table 4.2.

All trajectory planners were very fast, with most of the time generating velocity profiles within 10ms. This increases their suitability for real time use in autonomous systems, especially for the Spline and Smooth Spline Trajectory Planners. However, it can be observed that the NArc trajectory planning had a higher mean runtime and a much larger standard deviation in terms of runtime than the spline trajectory planner.

This is slightly counter-intuitive as there are some constraints that remain constant for the duration of entire Arcs in the Arc Trajectory Planner (used within the NArc Trajectory Planner). This should make the planner faster as less computations are needed (no need to iterate through the entire velocity profile within the planner for parts 1-2 as described in the Methodology section). Therefore, we believe that the longer mean runtime and higher standard deviations in the runtime for the NArc trajectory planner are attributed to overhead needed object orientation within Python. The NArc trajectory planner needs to calculate the velocity profile for each arc independently (where as the Spline trajectory planner does all the curve at once) thus needing to switch between processes and objects.

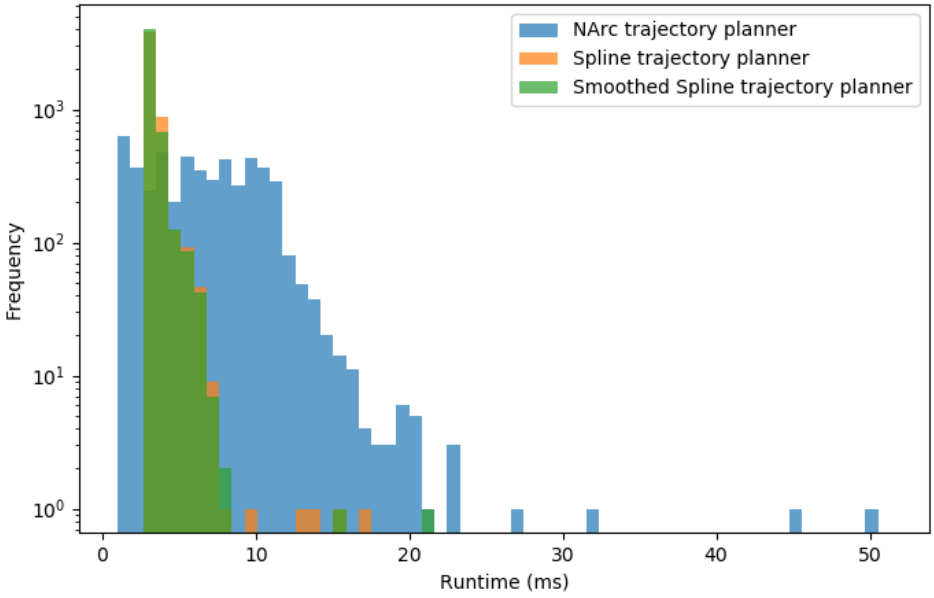

Figure 4.3: Runtime analysis of trajectory planners

It is important to note that our solutions are all considerably faster than those proposed by Okuyama et al. [2021]. Particularly, in 95% of cases our Smooth Spline trajectory planner took less than 4.81ms while the one proposed by Okuyama et al. [2021] took 13.8ms.

4.3.2 Simulation results

Unlike for the algorithimic results experiments from the previous section, it was infeasible to run the simulation for all of the 5000 randomly generated NArc sequences. This was due to the fact that the Gazebo simulation could not be sped up much faster than the real time. To test all 5000 NArc sequences 3 times (for comparing NArc, Spline, and Smooth Spline trajectory planning) it would have taken aproximately 84 hours. Instead it was decided to randomly choose a subset of 500 random NArc sequences and run the tests on them.

To test the feasibility of the generated trajectories, each of the 500 NArc paths was converted into trajectories using the NArc, Spline, and Smooth Spline trajectory planners and subsequently run in the simulation. The actual location of the robot was compared to the planned location of the robot, across the entire path. We also recorded if the robot deviated from the path beyond the stipulated safety buffer for that path (and thus could have entered into a collision with an obstacle).

Examples of comparisons between the intended path and the actual path of the robot can be seen in Figures 4.4, 4.5, and 4.6. In these three trajectories the robot is trying to follow the following NArc sequence:

1. Arc with radius 1m and length 2m turning right.

2. Arc with radius 1m and length 2m turning left.

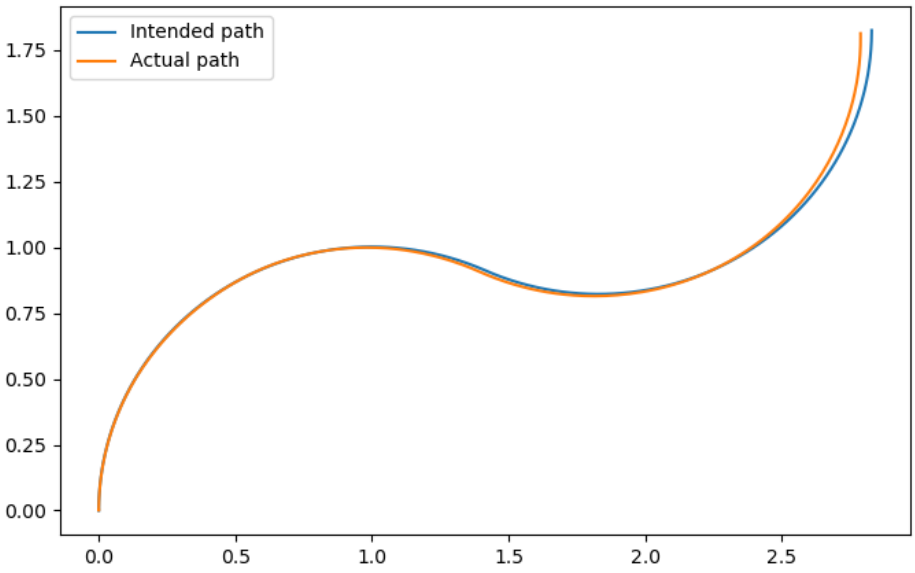

Figure 4.4: NArc trajectory example: intended path vs actual path

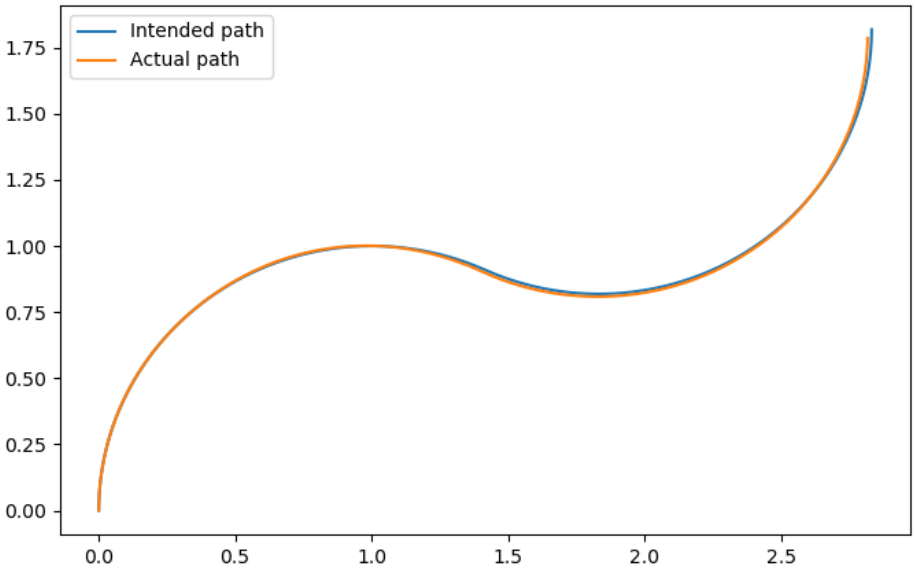

Figure 4.5: Spline trajectory example: intended path vs actual path

From the simulation experiments, it was observed that the TurtleBot 3 Burger would deviate slightly from the intended path (the path traversed by the robot would not be exactly the same as the path it was intended to traverse). It was also observed that this

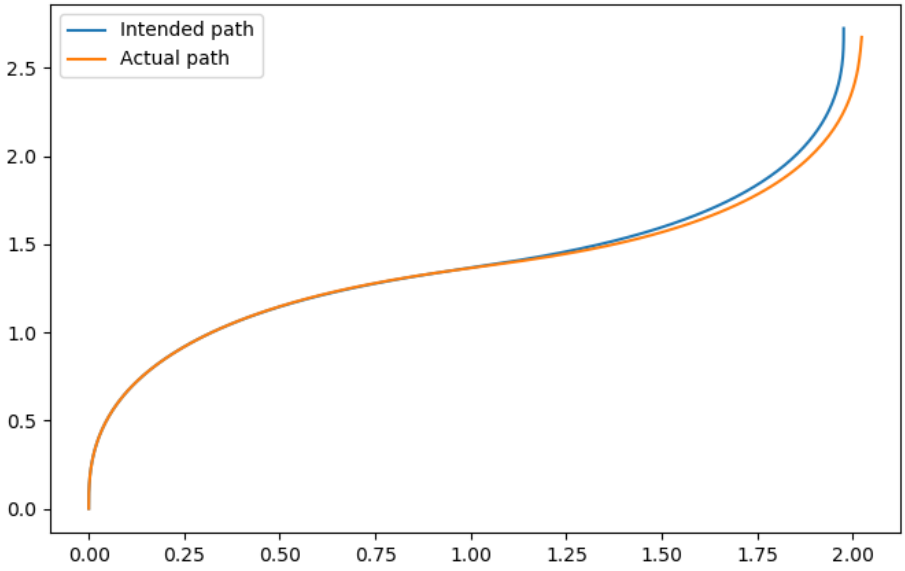

Figure 4.6: Smooth Spline trajectory example: intended path vs actual (path curve rotated counter-clockwise for easier comparison)

deviation would increase as the robot traversed the path (the farther the robot went from the starting point, the farther it deviated from the path). Usually, the robot would be at the farthest point from the path when it reached the end of the path. This is behaviour that is unfortunate although not unexpected (as long as the deviations do not become too large) given that there may exist delays in the system. For instance, if there is a slight random delay when executing a deceleration command, the robot will then increase its deviation from the intended path.

From our experimentation, the robot stayed within the specified safety boundaries in 95.4% of cases. This statistic, although quite high, is expected to be higher. We believe that part of the reason for it not being higher is due to some limitations with the ROS2 and Gazebo simulation software stack. It was observed that when running the tests the robot would sometimes spontaneously turn towards a random direction (even when ran multiple times consecutively on the same trajectory). The root cause of this behaviour was not found, but we suspect that it might be due to control commands not arriving in real time to the Turtlebot 3 Burger model. If commands do not arrive in real time, then the trajectory will not be followed as intended as the time distance between commands is not respected.

Chapter 5

Conclusions

Throughout this research paper, we have:

1. Created a trajectory planner for Arcs that attempts to follow the Arc as accurately and as fast as possible, without using the safety buffers.
2. Created a trajectory planner for N-Arcs that attempts to follow the N-Arc as accurately and as fast as possible, without using the safety buffers.
3. Created a trajectory planner for Splines that attempts to follow the original N-Arc path as accurately and as fast as possible, while using the indispensable part of the safety buffers.
4. Implemented an algorithm to smooth out a spline while using as much of the safety buffers as is required. From this, if used with the Spline trajectory planner, a robot can traverse a sequence of N-Arcs in a fast and accurate manner without entering into collision with objects.

As we have shown in the experimentation phase, all mentioned algorithms are very efficient, consistently running in the single or double digit millisecond range. The speed of the algorithms make the solutions proposed suitable for real-time use in self-driving systems. These systems could have a computationally expensive software stack and would benefit from a fast and accurate trajectory plan.

As can be seen in figure 4.6, the car tended to deviate from its intended path as it got farther from the original starting point. While some of it could be due to inconsistencies in the simulation tests, I believe some of the blame is also on the Jerk jumps that can be seen in the velocity profile. Jerk, the 2rd derivative of the velocity profile is closely related to the actuator limits of the motors. For future work, I recommend attempting to add constraints that limit the Jerk; this might make yield better results when testing in the simulation.

On a similar note, it is mentioned in Tokekar et al. [2014] that trapezoidal velocity profiles (such as the ones seen in this project) might not be very realistic due to large spikes in the Jerk.

Bibliography

Marek Boryga, Andrzej Graboś, Paweł Kołodziej, Krzysztof Gołacki, and Zbigniew Stropek. Trajectory planning with obstacles on the example of tomato harvest. *Agriculture and Agricultural Science Procedia*, 7:27–34, 2015.

Sean Campbell, Niall O'Mahony, Anderson Carvalho, Lenka Krpalkova, Daniel Riordan, and Joseph Walsh. Path planning techniques for mobile robots a review. In *2020 6th International Conference on Mechatronics and Robotics Engineering (ICMRE)*, pages 12–16. IEEE, 2020.

Marcio da Silva Arantes, Claudio Fabiano Motta Toledo, Brian Charles Williams, and Masahiro Ono. Collision-free encoding for chance-constrained nonconvex path planning. *IEEE Transactions on Robotics*, 35(2):433–448, 2019.

Th Fraichard. Smooth trajectory planning for a car in a structured world. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, volume 1, pages 318–323. Citeseer, 1991.

Alessandro Gasparetto, Paolo Boscariol, Albano Lanzutti, and Renato Vidoni. Trajectory planning in robotics. *Mathematics in Computer Science*, 6:269–279, 2012.

Alessandro Gasparetto, Paolo Boscariol, Albano Lanzutti, and Renato Vidoni. Path planning and trajectory planning algorithms: A general overview. *Motion and Operation Planning of Robotic Systems: Background and Practical Approaches*, pages 3–27, 2015.

Jayoung Kim and Jihong Lee. Predicting maximum traction to improve maneuverability for autonomous mobile robots on rough terrain. *Journal of Automation and Control Engineering*, 1(1):1–6, 2013.

W Kim, K Yi, and J Lee. An optimal traction, braking, and steering coordination strategy for stability and manoeuvrability of a six-wheel drive and six-wheel steer vehicle. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of automobile engineering*, 226(1):3–22, 2012.

J Zico Kolter, Christian Plagemann, David T Jackson, Andrew Y Ng, and Sebastian Thrun. A probabilistic approach to mixed open-loop and closed-loop control, with application to extreme autonomous driving. In *2010 IEEE International Conference on Robotics and Automation*, pages 839–845. IEEE, 2010.

Frank Lingelbach. Path planning using probabilistic cell decomposition. In *IEEE*

International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004, volume 1, pages 467–472. IEEE, 2004.

Matteo Massaro, Stefano Lovato, Matteo Bottin, and Giulio Rosati. An optimal control approach to the minimum-time trajectory planning of robotic manipulators. *Robotics*, 12(3):64, 2023.

IF Okuyama, Marcos ROA Maximo, and Rubens JM Afonso. Minimum-time trajectory planning for a differential drive mobile robot considering non-slipping constraints. *Journal of Control, Automation and Electrical Systems*, 32(1):120–131, 2021.

Aurelio Piazzi and Antonio Visioli. Global minimum-jerk trajectory planning of robot manipulators. *IEEE transactions on industrial electronics*, 47(1):140–149, 2000.

Elon Rimon and Daniel Koditschek. Exact robot navigation using artificial potential functions. In *IEEE Transactions on Robots and Automation Vol 8 No 5*, pages 501–518. IEEE, 1992.

ROBOTIS e-Manual. *TurtleBot3*, 2024.

Gurjeet Singh and VK Banga. Robots and its types for industrial applications. *Materials Today: Proceedings*, 60:1779–1786, 2022.

Alexandr Stefek, Thuan Van Pham, Vaclav Krivanek, and Khac Lam Pham. Energy comparison of controllers used for a differential drive wheeled mobile robot. *IEEE Access*, 8:170915–170927, 2020.

Pratap Tokekar, Nikhil Karnad, and Volkan Isler. Energy-optimal trajectory planning for car-like robots. *Autonomous Robots*, 37:279–300, 2014.

Zhijie Zhu, Edward Schmerling, and Marco Pavone. A convex optimization approach to smooth trajectories for motion planning with car-like robots. In *2015 54th IEEE conference on decision and control (CDC)*, pages 835–842. IEEE, 2015.

Appendix A

Constraint derivations

A.1 Longitudinal acceleration limit on the left wheel (constraint 3.3)

Problem statement

The following variables are given: v_0L, v_0R, r_1L, r_1R, $maxA_{long}$, s.

Assume that we have two consecutive points p_0 and p_1 on the curve. Then, v_0L and v_0R are the left and right wheel's velocities at p_0, r_1L and r_1R are the left and right wheel path's radius of curvature at p_1, $maxA_{long}$ is the maximum longitudinal acceleration that can be sustained by any particular wheel, and s is the distance between p_0 and p_1.

We are attempting to find relationships (constraints) for the following variables: v_1L and v_1R which are the left and right wheel's velocities at p_1.

The following relationships are given and assumed to always be true:

$$\frac{v_1R}{r_1R} = \frac{v_1L}{r_1L}$$

$$t = \frac{4 \cdot s}{v_0L + v_0R + v_1L + v_1R}$$

where t is the time taken for the robot to move from p_0 to p_1.

$$\frac{v_1L - v_0L}{t} \leq maxA_{long}$$

Derivation

The following is the step-by-step derivation of the constraint, built from the aforementioned relationships:

$$v_1L \leq t \cdot maxA_{long} + v_0L$$

$$v_1L \leq \frac{4 \cdot s \cdot maxA_{long}}{v_0L + v_0R + (1 + \frac{r_1R}{r_1L}) \cdot v_1L} + v_0L$$

$$[v_0L + v_0R + (1 + \frac{r_1R}{r_1L}) \cdot v_1L] \cdot v_1L \leq 4 \cdot s \cdot maxA_{long} + [v_0L + v_0R + (1 + \frac{r_1R}{r_1L}) \cdot v_1L] \cdot v_0L$$

$$[v_0L + v_0R] \cdot v_1L + [1 + \frac{r_1R}{r_1L}] \cdot v_1L^2 \leq 4 \cdot s \cdot maxA_{long} + v_0L^2 + v_0L \cdot v_0R + [1 + \frac{r_1R}{r_1L}] \cdot v_0L \cdot v_1L$$

$$[1 + \frac{r_1R}{r_1L}] \cdot v_1L^2 + [v_0L + v_0R - (1 + \frac{r_1R}{r_1L}) \cdot v_0L] \cdot v_1L - [4 \cdot s \cdot maxA_{long} + v_0L^2 + v_0L \cdot v_0R] \leq 0$$

Applying the quadratic formula (negative root ignored):

$$v_1L \leq \frac{\sqrt{[v_0L + v_0R - (1 + \frac{r_1R}{r_1L}) \cdot v_0L]^2 + 4 \cdot [1 + \frac{r_1R}{r_1L}] \cdot [4 \cdot s \cdot maxA_{long} + v_0L^2 + v_0L \cdot v_0R]}}{2 \cdot [1 + \frac{r_1R}{r_1L}]} - \frac{[v_0L + v_0R - (1 + \frac{r_1R}{r_1L}) \cdot v_0L]}{2 \cdot [1 + \frac{r_1R}{r_1L}]}$$

Simplification:

$$v_1L \leq \frac{\sqrt{4 \cdot r_1L \cdot (r_1L + r_1R) \cdot (v_0L^2 + v_0L \cdot v_0R + 4 \cdot s \cdot maxA_{long}) + (r_1R \cdot v_0L - r_1L \cdot v_0R)^2}}{2 \cdot (r_1L + r_1R)} - \frac{r_1L \cdot v_0R + r_1R \cdot v_0L}{2 \cdot (r_1L + r_1R)}$$

A.2 Longitudinal acceleration limit on the right wheel (constraint 3.4)

Problem statement

The following variables are given: v_0L, v_0R, r_1L, r_1R, $maxA_{long}$, s.

Assume that we have two consecutive points p_0 and p_1 on the curve. Then, v_0L and v_0R are the left and right wheel's velocities at p_0, r_1L and r_1R are the left and right wheel path's radius of curvature at p_1, $maxA_{long}$ is the maximum longitudinal acceleration that can be sustained by any particular wheel, and s is the distance between p_0 and p_1.

We are attempting to find relationships (constraints) for the following variables: v_1L and v_1R which are the left and right wheel's velocities at p_1.

The following relationships are given and assumed to always be true:

$$\frac{v_1R}{r_1R} = \frac{v_1L}{r_1L}$$

$$t = \frac{4 \cdot s}{v_0L + v_0R + v_1L + v_1R}$$

where t is the time taken for the robot to move from p_0 to p_1.

$$\frac{v_1R - v_0R}{t} \leq maxA_{long}$$

Derivation

The following is the step-by-step derivation of the constraint, built from the aforementioned relationships:

$$v_1R \leq t \cdot maxA_{long} + v_0R$$

$$v_1R \leq \frac{4 \cdot s \cdot maxA_{long}}{v_0L + v_0R + (1 + \frac{r_1L}{r_1R}) \cdot v_1R} + v_0R$$

$$[v_0L + v_0R + (1 + \frac{r_1L}{r_1R}) \cdot v_1R] \cdot v_1R \leq 4 \cdot s \cdot maxA_{long} + [v_0L + v_0R + (1 + \frac{r_1L}{r_1R}) \cdot v_1R] \cdot v_0R$$

$$[v_0L + v_0R] \cdot v_1R + [1 + \frac{r_1L}{r_1R}] \cdot v_1R^2 \leq 4 \cdot s \cdot maxA_{long} + v_0R^2 + v_0L \cdot v_0R + [1 + \frac{r_1L}{r_1R}] \cdot v_0R \cdot v_1R$$

$$[1 + \frac{r_1L}{r_1R}] \cdot v_1R^2 + [v_0L + v_0R - (1 + \frac{r_1L}{r_1R}) \cdot v_0R] \cdot v_1R - [4 \cdot s \cdot maxA_{long} + v_0R^2 + v_0L \cdot v_0R] \leq 0$$

Applying the quadratic formula (negative root ignored):

$$v_1R \leq \frac{\sqrt{[v_0L + v_0R - (1 + \frac{r_1L}{r_1R}) \cdot v_0R]^2 + 4 \cdot [1 + \frac{r_1L}{r_1R}] \cdot [4 \cdot s \cdot maxA_{long} + v_0R^2 + v_0L \cdot v_0R]}}{2 \cdot [1 + \frac{r_1L}{r_1R}]} - \frac{[v_0L + v_0R - (1 + \frac{r_1L}{r_1R}) \cdot v_0R]}{2 \cdot [1 + \frac{r_1L}{r_1R}]}$$

Simplification:

$$v_1R \leq \frac{\sqrt{4 \cdot r_1R \cdot (r_1L + r_1R) \cdot (v_0R^2 + v_0L \cdot v_0R + 4 \cdot s \cdot maxA_{long}) + (r_1L \cdot v_0R - r_1R \cdot v_0L)^2}}{2 \cdot (r_1L + r_1R)} - \frac{r_1R \cdot v_0L + r_1L \cdot v_0R}{2 \cdot (r_1L + r_1R)}$$

A.3 Longitudinal deceleration limit on the left wheel (constraint 3.5)

Problem statement

The following variables are given: v_1L, v_1R, r_0L, r_0R, $maxD_{long}$, s.

Assume that we have two consecutive points p_0 and p_1 on the curve. Then, v_1L and v_1R are the left and right wheel's velocities at p_1, r_0L and r_0R are the left and right wheel path's radius of curvature at p_0, $maxD_{long}$ is the maximum longitudinal deceleration that can be sustained by any particular wheel, and s is the distance between p_0 and p_1.

We are attempting to find relationships (constraints) for the following variables: v_0L and v_0R which are the left and right wheel's velocities at p_0.

The following relationships are given and assumed to always be true:

$$\frac{v_0R}{r_0R} = \frac{v_0L}{r_0L}$$

$$t = \frac{4 \cdot s}{v_1L + v_1R + v_0L + v_0R}$$

where t is the time taken for the robot to move from p_0 to p_1.

$$\frac{v_0L - v_1L}{t} \leq maxD_{long}$$

Derivation

The following is the step-by-step derivation of the constraint, built from the aforementioned relationships:

$$v_0L \leq t \cdot maxD_{long} + v_1L$$

$$v_0L \leq \frac{4 \cdot s \cdot maxD_{long}}{v_1L + v_1R + (1 + \frac{r_0R}{r_0L}) \cdot v_0L} + v_1L$$

$$[v_1L + v_1R + (1 + \frac{r_0R}{r_0L}) \cdot v_0L] \cdot v_0L \leq 4 \cdot s \cdot maxD_{long} + [v_1L + v_1R + (1 + \frac{r_0R}{r_0L}) \cdot v_0L] \cdot v_1L$$

$$[v_1L + v_1R] \cdot v_0L + [1 + \frac{r_0R}{r_0L}] \cdot v_0L^2 \leq 4 \cdot s \cdot maxD_{long} + v_1L^2 + v_1L \cdot v_1R + [1 + \frac{r_0R}{r_0L}] \cdot v_1L \cdot v_0L$$

$$[1 + \frac{r_0R}{r_0L}] \cdot v_0L^2 + [v_1L + v_1R - (1 + \frac{r_0R}{r_0L}) \cdot v_1L] \cdot v_0L - [4 \cdot s \cdot maxD_{long} + v_1L^2 + v_1L \cdot v_1R] \leq 0$$

Applying the quadratic formula (negative root ignored):

$$v_0L \leq \frac{\sqrt{[v_1L+v_1R-(1+\frac{r_0R}{r_0L})\cdot v_1L]^2+4\cdot[1+\frac{r_0R}{r_0L}]\cdot[4\cdot s\cdot maxD_{long}+v_1L^2+v_1L\cdot v_1R]}}{2\cdot[1+\frac{r_0R}{r_0L}]} - \frac{[v_1L+v_1R-(1+\frac{r_0R}{r_0L})\cdot v_1L]}{2\cdot[1+\frac{r_0R}{r_0L}]}$$

Simplification:

$$v_0L \leq \frac{\sqrt{4\cdot r_0L\cdot(r_0L+r_0R)\cdot(v_1L^2+v_1L\cdot v_1R+4\cdot s\cdot maxD_{long})+(r_0R\cdot v_1L-r_0L\cdot v_1R)^2}}{2\cdot(r_0L+r_0R)} - \frac{r_0L\cdot v_1R+r_0R\cdot v_1L}{2\cdot(r_0L+r_0R)}$$

A.4 Longitudinal deceleration limit on the right wheel (constraint 3.6)

Problem statement

The following variables are given: v_1L, v_1R, r_0L, r_0R, $maxD_{long}$, s.

Assume that we have two consecutive points p_0 and p_1 on the curve. Then, v_1L and v_1R are the left and right wheel's velocities at p_1, r_0L and r_0R are the left and right wheel path's radius of curvature at p_0, $maxD_{long}$ is the maximum longitudinal deceleration that can be sustained by any particular wheel, and s is the distance between p_1 and p_0.

We are attempting to find relationships (constraints) for the following variables: v_0L and v_0R which are the left and right wheel's velocities at p_0.

The following relationships are given and assumed to always be true:

$$\frac{v_0R}{r_0R} = \frac{v_0L}{r_0L}$$

$$t = \frac{4\cdot s}{v_1L+v_1R+v_0L+v_0R}$$

where t is the time taken for the robot to move from p_0 to p_1.

$$\frac{v_0R-v_1R}{t} \leq maxD_{long}$$

Derivation

The following is the step-by-step derivation of the constraint, built from the aforementioned relationships:

$$v_0R \leq t \cdot maxD_{long} + v_1R$$

$$v_0R \leq \frac{4 \cdot s \cdot maxD_{long}}{v_1L + v_1R + (1 + \frac{r_0L}{r_0R}) \cdot v_0R} + v_1R$$

$$[v_1L + v_1R + (1 + \frac{r_0L}{r_0R}) \cdot v_0R] \cdot v_0R \leq 4 \cdot s \cdot maxD_{long} + [v_1L + v_1R + (1 + \frac{r_0L}{r_0R}) \cdot v_0R] \cdot v_1R$$

$$[v_1L + v_1R] \cdot v_0R + [1 + \frac{r_0L}{r_0R}] \cdot v_0R^2 \leq 4 \cdot s \cdot maxD_{long} + v_1R^2 + v_1L \cdot v_1R + [1 + \frac{r_0L}{r_0R}] \cdot v_1R \cdot v_0R$$

$$[1 + \frac{r_0L}{r_0R}] \cdot v_0R^2 + [v_1L + v_1R - (1 + \frac{r_0L}{r_0R}) \cdot v_1R] \cdot v_0R - [4 \cdot s \cdot maxD_{long} + v_1R^2 + v_1L \cdot v_1R] \leq 0$$

Applying the quadratic formula (negative root ignored):

$$v_0R \leq \frac{\sqrt{[v_1L + v_1R - (1 + \frac{r_0L}{r_0R}) \cdot v_1R]^2 + 4 \cdot [1 + \frac{r_0L}{r_0R}] \cdot [4 \cdot s \cdot maxD_{long} + v_1R^2 + v_1L \cdot v_1R]}}{2 \cdot [1 + \frac{r_0L}{r_0R}]} - \frac{[v_1L + v_1R - (1 + \frac{r_0L}{r_0R}) \cdot v_1R]}{2 \cdot [1 + \frac{r_0L}{r_0R}]}$$

Simplification:

$$v_0R \leq \frac{\sqrt{4 \cdot r_0R \cdot (r_0L + r_0R) \cdot (v_1R^2 + v_1L \cdot v_1R + 4 \cdot s \cdot maxD_{long}) + (r_0L \cdot v_1R - r_0R \cdot v_1L)^2}}{2 \cdot (r_0L + r_0R)} - \frac{r_0R \cdot v_1L + r_0L \cdot v_1R}{2 \cdot (r_0L + r_0R)}$$

A.5 Simultaneous Acceleration & deceleration limit due to changing radiuson the left wheel

Problem statement

The following variables are given: r_0, r_0L, r_0R, r_1, r_1L, r_1R, $maxA_{long}$, $maxD_{long}$, s.

Assume that we have two consecutive points p_0 and p_1 on the curve. r_0, r_0L, r_0R are the centre of vehicle velocity, left, and right wheel's velocities at p_0 respectively; r_1, r_1L, r_1L are the centre of vehicle velocity, left, and right wheel's velocities at p_1 respectively;

$maxA_{long}$ and $maxD_{long}$ are the maximum longitudinal acceleration and deceleration that can be sustained by any particular wheel; and s is the distance between p_0 and p_1.

We are attempting to find relationships (constraints) for the following variables: v_0 and v_1 which are velocities of the vehicle at at p_0 and p_1.

The following relationships are given and assumed to always be true:

$$\frac{v_0}{r_0} = \frac{v_0L}{r_0L} = \frac{v_0R}{r_0R}$$

$$\frac{v_1}{r_1} = \frac{v_1L}{r_1L} = \frac{v_1R}{r_1R}$$

$$t = \frac{2 \cdot s}{v_0 + v_1} = \frac{4 \cdot s}{v_0L + v_0R + v_1L + v_1R}$$

where t is the time taken for the robot to move from p_0 to p_1.

$$-maxD_{long} \leq \frac{v_1L - v_0L}{t} \leq maxA_{long}$$

$$-maxD_{long} \leq \frac{v_1R - v_0R}{t} \leq maxA_{long}$$

Derivation

The following is the step-by-step derivation of the constraint, built from the aforementioned relationships:

$$maxD_{long} \leq \frac{(r_1L \cdot r_0 \cdot v_1 - r_0L \cdot r_1 \cdot v_0) \cdot (v_0 + v_1)}{2 \cdot s \cdot r_0 \cdot r_1} \leq maxA_{long}$$

$$maxD_{long} \leq \frac{(r_1R \cdot r_0 \cdot v_1 - r_0R \cdot r_1 \cdot v_0) \cdot (v_0 + v_1)}{2 \cdot s \cdot r_0 \cdot r_1} \leq maxA_{long}$$

(substituted the time relationship and simplified)

This leaves us with two sets of simultaneous equations (since we are finding the constraints when one wheel accelerates and another decelerates):

$$maxD_{long} \leq \frac{(r_1L \cdot r_0 \cdot v_1 - r_0L \cdot r_1 \cdot v_0) \cdot (v_0 + v_1)}{2 \cdot s \cdot r_0 \cdot r_1}, \frac{(r_1R \cdot r_0 \cdot v_1 - r_0R \cdot r_1 \cdot v_0) \cdot (v_0 + v_1)}{2 \cdot s \cdot r_0 \cdot r_1} \leq maxA_{long}$$

or

$$\frac{(r_1L \cdot r_0 \cdot v_1 - r_0L \cdot r_1 \cdot v_0) \cdot (v_0 + v_1)}{2 \cdot s \cdot r_0 \cdot r_1} \leq maxA_{long}, maxD_{long} \leq \frac{(r_1R \cdot r_0 \cdot v_1 - r_0R \cdot r_1 \cdot v_0) \cdot (v_0 + v_1)}{2 \cdot s \cdot r_0 \cdot r_1}$$

From the above simultaneous equations, we solved for v_0 and v_q, the variables we are trying to find. We will not show the derivation of the below equations as it is very long winded.

If:

$$0 \leq s \cdot {r_1}^2 \cdot {r_0}^2 \cdot (r_1L \cdot r_0R - r_1R \cdot r_0L) \cdot (maxA_{long} \cdot (r_1 \cdot r_0R + r_0 \cdot r_1R) + maxD_{long} \cdot (r_1 \cdot r_0L + r_0 \cdot r_1L))$$

Then:

$$v_0 \leq \frac{\sqrt{2} \cdot s \cdot {r_1} \cdot {r_0}^2 \cdot (r_1 L \cdot maxD_{long} + r_1 R \cdot maxA_{long})}{\sqrt{s \cdot {r_1}^2 \cdot {r_0}^2 \cdot (r_1 L \cdot r_0 R - r_1 R \cdot r_0 L) \cdot (maxA_{long} \cdot (r_1 \cdot r_0 R + r_0 \cdot r_1 R) + maxD_{long} \cdot (r_1 \cdot r_0 L + r_0 \cdot r_1 L))}}$$

$$v_1 \leq \frac{\sqrt{2} \cdot s \cdot {r_1}^2 \cdot r_0 \cdot (r_0 L \cdot maxD_{long} + r_0 R \cdot maxA_{long})}{\sqrt{s \cdot {r_1}^2 \cdot {r_0}^2 \cdot (r_1 L \cdot r_0 R - r_1 R \cdot r_0 L) \cdot (maxA_{long} \cdot (r_1 \cdot r_0 R + r_0 \cdot r_1 R) + maxD_{long} \cdot (r_1 \cdot r_0 L + r_0 \cdot r_1 L))}}$$

Else:

$$v_0 \leq \frac{\sqrt{2} \cdot s \cdot {r_1} \cdot {r_0}^2 \cdot (r_1 L \cdot maxA_{long} + r_1 R \cdot maxD_{long})}{-s \cdot {r_1}^2 \cdot {r_0}^2 \cdot (r_1 L \cdot r_0 R - r_1 R \cdot r_0 L) \cdot (maxA_{long} \cdot (r_1 \cdot r_0 L + r_0 \cdot r_1 L) + maxD_{long} \cdot (r_1 \cdot r_0 R + r_0 \cdot r_1 R))}$$

$$v_1 \leq \frac{\sqrt{2} \cdot s \cdot {r_1}^2 \cdot r_0 \cdot (r_0 L \cdot maxA_{long} + r_0 R \cdot maxD_{long})}{-s \cdot {r_1}^2 \cdot {r_0}^2 \cdot (r_1 L \cdot r_0 R - r_1 R \cdot r_0 L) \cdot (maxA_{long} \cdot (r_1 \cdot r_0 L + r_0 \cdot r_1 L) + maxD_{long} \cdot (r_1 \cdot r_0 R + r_0 \cdot r_1 R))}$$