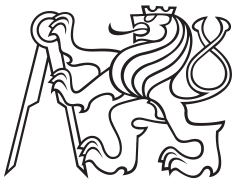
Bachelor's Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

MVP algorithm modification using polynomial paths

Jan Hadraba

Supervisor: doc. Ing. Tomáš Haniš, Ph.D.
Field of study: Cybernetics and Robotics
January 2025

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Hadraba Jan** Personal ID number: **508457**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Control Engineering**

Study program: **Cybernetics and Robotics**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

MVP algorithm modification using polynomial paths

Bachelor’s thesis title in Czech:

Modifikace MVP algoritmu s využitím polynomiálních trajektorií

Guidelines:

The goal of this thesis is to modify Minimum Violation Planning (MVP) algorithm using polynomial trajectories to expand and reconnect tree nodes. The algorithm shall account also for over-actuated vehicles (namely, the crab walk scenario) and lateral dynamics.
1. Get familiar with vehicle dynamic mathematical models and traction limits.
2. Get familiar with path planning algorithms. Prepare benchmark implementation.
3. Implement traction limits to planning algorithm.
4. Evaluate developed algorithm and compare computational cost and planning performance compared to the benchmark.
5. Verify implemented modification.

Bibliography / sources:

[1] Dieter Schramm, Manfred Hiller, Roberto Bardini – Vehicle Dynamics – Duisburg 2014
[2] Richter, Charles, Adam Bry, and Nicholas Roy. “Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments.” In Robotics Research, edited by Masayuki Inaba and Peter Corke, 114:649–66. Springer Tracts in Advanced Robotics. Cham: Springer International Publishing, 2016. https://doi.org/10.1007/978-3-319-28872-7_37.
[3] LaValle, Steven M.: Planning algorithms. Cambridge University Press, 2006.
[4] LaValle, Steven M., and James J. Kuffner Jr.: Rapidly-exploring random trees: Progress and prospects, (2000).
[5] Finney, Sarah, Leslie Kaelbling, and Tomas Lozano-Perez. “Predicting Partial Paths from Planning Problem Parameters,” n.d.
[6] Rajamani R. (2012) Mean Value Modeling of SI and Diesel Engines. In: Vehicle Dynamics and Control. Mechanical Engineering Series. Springer, Boston, MA. https://doi.org/10.1007/978-1-4614-1433-9_9

Name and workplace of bachelor’s thesis supervisor:

doc. Ing. Tomáš Haniš, Ph.D. Department of Control Engineering FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **16.01.2024** Deadline for bachelor thesis submission: **07.01.2025**

Assignment valid until:
by the end of summer semester 2024/2025

doc. Ing. Tomáš Haniš, Ph.D. Supervisor’s signature	doc. Ing. Zdeněk Hurák, Ph.D. Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

. _____ Date of assignment receipt

_____ Student's signature

Acknowledgements

First, I would like to thank my supervisor doc. Ing. Tomáš Haniš, Ph.D. for his advice and patience while working on this thesis.

A special thanks belong to my family and friends who supported me not only while working on this thesis, but also during my studies in general.

Declaration

I hereby declare that this work is all my own work and I have cited all sources I have used in the bibliography.

Prague, January 7, 2025

Abstract

Path planning for autonomous vehicles remains a central challenge in intelligent transportation systems, requiring algorithms that can efficiently generate safe, smooth, and feasible paths in real time. This thesis deals with implementing modifications to an already existing minimum-violation planning (MVP) algorithm. This modification is based on polynomial paths constructed under curvature constraints considering the surface friction coefficient. The developed algorithm is compared to the already existing algorithm and is then verified in several different test case scenarios.

Keywords: MVP, RRT*, Polynomial paths, Local planning, Autonomous vehicles

Supervisor: doc. Ing. Tomáš Haniš, Ph.D.

Abstrakt

Plánování trajektorií pro autonomní vozidla představuje klíčocou výzvu v oblasti inteligentnich dopravních systémů, kde je nutné zajistit generování bezpečných, plynulých a proveditelných trajektorií v reálném čase. Tato práce se zabývá implementací modifikace již existujícího (MVP) algoritmu. Tato modifikace spočívá v použití polynomiálních trajektorií vytvářených pod omezením křivosti s ohledem na koeficient tření povrchu. Vyvinutý algoritmus je porovnán s již existujícím algoritmem a následně je ověřen v několika různých testovacích případech.

Klíčová slova: MVP, RRT*, Polynomiální trajektorie, Lokální plánování, Autonomní vozidla

Překlad názvu: Modifkace MVP algoritmu s využitím polynomiálních trajektorií

Contents

Figures

Chapter 1

Introduction

The ever-advancing field of autonomous vehicles offers plenty of challenges whether it is sensing of vehicle's surroundings, motion control, decision-making or path planning, to name a few. One of the most critical challenges is efficient and safe path planning. Path planning relates to the process by which an autonomous vehicle creates an optimal route from its current location to a desired goal destination while avoiding obstacles and navigating through complex environments. A complex and dynamic environment contains unpredictable elements, such as moving objects (other vehicles, pedestrians, animals), and varying weather conditions. Besides obstacle avoidance and complex environments, many factors make path planning such a challenging task. For example, high dimensionality, because not only position but also vehicle dynamics, such as velocity and steering, must be taken into account. Or smoothness and comfort of the resulting trajectory, since the path planning algorithm should ensure that the generated trajectory respects vehicle motion constraints and weather conditions are taken into account. Lastly, the time complexity, because the algorithm should be able to work in real-time, which places even greater demands on it.

1.1 Motivation

Among the various techniques developed for path planning, the use of advanced search algorithms like rapidly exploring random trees (RRT) has gained significant attention due to their ability to generate feasible and collision-free paths. This thesis aims to modify already existent local path

planning algorithm developed by Marek Boháč [1], with a new steering function based on polynomials [2]. The steering function also takes into account additional constraints, namely the adhesion (friction) coefficient μ, and is supported with a new sampling method. The inclusion of the friction coefficient means that methods for coefficient friction estimation from cameras might be utilized [3]. The most considerable advantage of this new approach is computational efficiency. Marek's algorithm needs a set of possible trajectories, which are precomputed with the use of input sampling and solving the initial value problem (IVP). IVP is solved with the use of iterative methods like Runge-Kutta on differential equations describing the motion of the vehicle. In contrast to Marek's approach, the computation of polynomial paths is less demanding and therefore, could be used in real-time and provide a more reactive planning, in the cost of the solution being a path and not a trajectory. So this algorithm with polynomial paths needs a higher level path tracking algorithm.

1.2 Thesis structure

First, in Chapter 2, I introduce state of the art planning algorithms. Then in Chapter 3 I define a goal of this thesis and describe an appropriate vehicle model selected to fulfill the goal. Then in Section 3.3, I describe how the existing algorithm is modified and what is missed out when using this approach. Lastly, in Chapter 4 I verify the performance of the designed algorithm and compare it with Marek's algorithm which is used as a benchmark.

Chapter 2

State of the art

2.1 Path planning

The problem of path planning for autonomous vehicles can be divided into two categories, local and global. The main goal of global path planning is to find an optimal, collision-free path from the vehicle's initial position to its goal using a map of the environment. The map can be created using prior knowledge or simultaneous localization and mapping (SLAM) method. Global planners operate at a higher scale and guide over longer distances. They are effective in static or well-known environments but struggle to adapt to unforeseen obstacles and dynamic changes in the environment due to the fact, that they rely on predefined maps. In contrast, local path planning focuses on the vehicle's surroundings at any given moment. Modern autonomous vehicles are equipped with a variety of sensors, such as radar, LiDAR, GPS and cameras. Data from these sensors are then utilized to detect obstacles and other changes in the environment, for example, a change in the driving surface, and navigate safely through the environment. Local planners operate in real-time during the vehicle's motion, allowing it to adapt its path in response to environmental changes and obstacles. That makes local path planning essential for navigating dynamic or incompletely known environments [4]. This thesis focuses primarily on local path planning, as one of its objectives is to incorporate the effect of different surface adhesion (friction) coefficients on path generation.

2.2 Planning algorithms

Path planning algorithms can be also divided into three main categories. In the first category are graph-based algorithms such as Dijkstra's and A* (and its other variants). The second category of path planning algorithms are algorithms that are based on sampling, RRT and probabilistic road map (PRM), and the third category are optimization-based algorithms [5], [6]. There are many other planning algorithms, but these are the most commonly used ones for autonomous vehicles [7].

The beginning of research and development of graph-based algorithms dates back to the 1960s, with Dijkstra's being first introduced in 1959 and A* later in 1968 [8], [9]. These two algorithms require discretization of the environment and don't guarantee that the resulting path will be feasible [10]. A* can be seen as an extension of Dijkstra's algorithm. It combines Dijsktra's precision with heuristic guidance, which results in much better performance while still providing optimal solutions. That makes it a popular choice in robotics, although only for problems with low dimensionality because with an increasing number of dimensions, the performance starts to stagger.

Optimization-based techniques utilize mathematical optimization methods, through defining an objective function such as minimizing travel time, fuel consumption or a weighted combination of multiple objectives. Frequently employed methods are linear (LP) and nonlinear programming (NLP), dynamic programming (DP) and model predictive control (MPC) [11].

Sampling-based methods are a class of path-planning algorithms widely used in robotics and other associated fields due to their ability to efficiently explore high-dimensional configuration spaces. These methods focus on generating feasible paths by sampling points in the environment, rather than explicitly constructing the entire configuration space. This makes them particularly effective for real-time applications in dynamic and complex scenarios. Random sampling allows the algorithms to avoid local minima and effectively navigate around obstacles. It is also ensured that generated samples do not collide with anything in the environment. This method was chosen in the original algorithm, therefore it will be described further in the work [1].

2.2.1 Sampling-based algorithms

There are two mostly used sampling-based algorithms, one of them is RRT and the other is PRM [12], [13]. With PRM, the algorithm first generates random samples throughout the space. These samples then become nodes in a graph and the algorithm may also use focused sampling strategies to join disconnected components of the graph. Once the graph is constructed, it can be reused to solve multiple different path planning queries within the same environment. RRT also randomly samples the configuration space, but in contrast to PRM, it grows a tree from a start configuration. The tree expands towards random samples until a goal is reached. Those methods are probabilistically complete, which means that they are guaranteed to find a solution if one exists given time goes to infinity. Due to the fact, that RRT creates paths between nodes during runtime and therefore can apply kinematic constraints, it is chosen as the suitable algorithm.

2.2.2 RRT and its variants

RRT was developed as a way to search high-dimensional or complex configuration spaces, such as those encountered in planning motions for robotic arms, mobile robots or autonomous vehicles with differential constraints. The basic idea of RRT is to grow the tree incrementally from an initial configuration (or state) by repeatedly sampling random points in the configuration space and expanding the existing tree towards these samples. Random sampling ensures, that the tree reaches unexplored regions and all feasible parts are covered over time. Pseudocode of RRT algorithm is described in Algorithm 1 based on [12]. Consider a path planning problem in an obstacle-free, continuous configuration space $\mathcal{C}$ which includes possible configurations of an autonomous vehicle. It starts with an initial configuration $q_0 \in \mathcal{C}$, and then in each iteration, the Rand function selects a random sample over uniform distribution from configuration space $\mathcal{C}$. This sample q_{rand} is passed to the Nearest function, which looks for the nearest node of the tree $\mathcal{T}$ according to a chosen distance metric. This metric can be either Euclidean distance or some more specialized metric that is used for a specific scenario, for example, weighted Euclidean distance. The nearest node q_{near} is used in the New_conf function to find a new configuration q_{new}, which is going to be added as a new node to the tree $\mathcal{T}$. The New_conf function should also check whether the path from q_{near} to q_{new} is collision-free. After that the q_{new} is added as a new vertex to the tree $\mathcal{T}$ and an edge between q_{near}, q_{new} is created. A graphical illustration of the algorithm is available in Figure 2.1.

Algorithm 1 RRT

Input: q_0 - initial configuration, N - number of vertices, ε - incremental distance, $\mathcal{C}$ - configuration space
Output: RRT graph $\mathcal{T}$ with N vertices

```
T.init(q_0)
while T.number_of_vertices() < N do
    q_rand ← Rand(C)
    q_near ← Nearest(q_rand, T)
    q_new ← New_conf(q_near, q_rand, ε)
    T.add_vertex(q_new)
    T.add_edge(q_near, q_new)
end while
return T
```

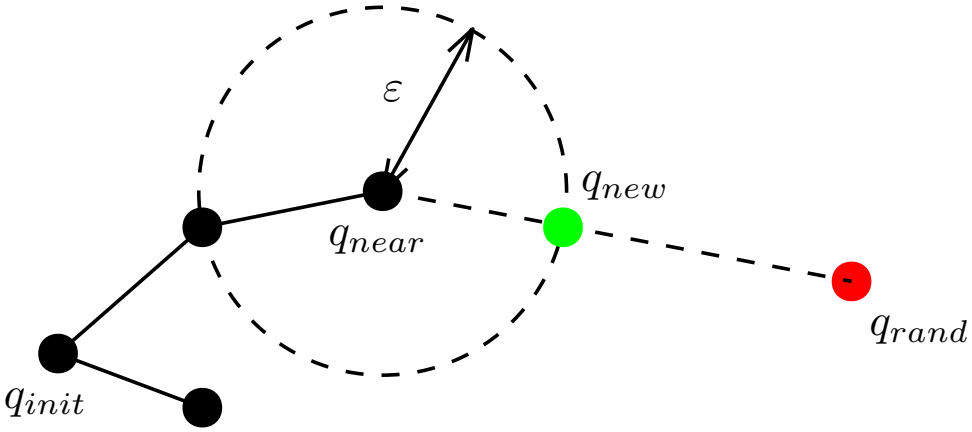


Figure 2.1: Expansion of RRT algorithm.

Even though this algorithm is probabilistically complete and effective in finding a feasible solution, it has one major issue when it comes to path planning. Paths found by this original RRT algorithm are not guaranteed to be optimal. Therefore asymptotically optimal version of this algorithm was developed, the RRT* [14]. Being asymptotically optimal means, that if the number of iterations approaches infinity, the probability that RRT* finds an optimal solution goes to 1. This is a huge step forward when it comes to autonomous vehicle path planning. In that case, optimal can mean minimal time travel, minimal distance traveled or minimal energy consumption. The optimality is accomplished mainly because of two modifications, choosing the best parent node and rewiring. Choice of the best parent is used during the selection of a new node q_{new}. Instead of simply connecting the q_{new} to its nearest neighbor (as in RRT), RRT* considers the neighborhood of existing nodes around the q_{new}. Among these neighbors, it selects the node that offers the lowest cost for the q_{new} as a parent node. The cost can represent distance traveled, time, energy, or other relevant metrics. By selecting the parent node with the minimum cost path, RRT* improves the generated route. Then

comes the process of rewiring. After attaching q_{new} to the best parent, RRT* attempts to rewire the neighborhood. Particularly, for each neighbor q_{near} in the proximity of q_{new} with radius r, the algorithm checks if passing through q_{new} would result in a lower cost path to q_{near}. If so, it rewires the parent of q_{near} to q_{new}. This step can systematically reduce the overall path cost in the tree, leading to better solutions over time. Both of these procedures can be seen in Algorithm 2.

Algorithm 2 RRT*

Input: q_0 - initial configuration, N - number of vertices, ε - incremental distance, $\mathcal{C}$ - configuration space
Output: RRT* graph $\mathcal{T}$ with N vertices

$\mathcal{T}$.init(q_0);
while $\mathcal{T}$.number_of_vertices() $< N$ **do**
  $q_{rand} \leftarrow$ Rand($\mathcal{C}$);
  $q_{nearest} \leftarrow$ Nearest($q_{rand}, \mathcal{T}$);
  $q_{new} \leftarrow$ New_conf($q_{nearest}, q_{rand}, \varepsilon$);
  $Q_{near} \leftarrow$ Near($q_{new}, \mathcal{T}$);
  $\mathcal{T}$.add_vertex(q_{new});
  $q_{min} \leftarrow q_{nearest}$;
  $c_{min} \leftarrow$ Cost($q_{nearest}$) + c(Line(q_{new}, q_{near});
  for $q_{near} \in Q_{near}$ **do**
    if Cost(q_{near}) + c(Line(q_{near}, q_{new}) $< c_{min}$ **then**
      $q_{min} \leftarrow q_{near}$;
      $c_{min} \leftarrow c$(Line(q_{near}, q_{new});
    end if
  end for
  $\mathcal{T}$.add_edge(q_{min}, q_{new});
  for $q_{near} \in Q_{near} \setminus q_{min}$ **do**
    if Cost(q_{new}) + c(Line(q_{new}, q_{near}) $<$ Cost(q_{near}) **then**
      $q_{parent} \leftarrow \mathcal{T}$.get_parent($q_{near}$);
      $\mathcal{T}$.remove_edge(q_{parent}, q_{near});
      $\mathcal{T}$.add_edge(q_{new}, q_{near});
    end if
  end for
end while
return $\mathcal{T}$;

2.2.3 Minimum-violation planning

Now until this section, algorithms able to generate collision-free paths were introduced. However, many real-world tasks require balancing multiple constraints that may not always be completely satisfiable, for example, temporal logic, environmental specifications and dynamic obstacles. In these cases,

it may be impossible to comply with all of the constraints simultaneously. Minimum-violation planning (MVP) handles this challenge by allowing small deviations from ideal constraints while prioritizing overall feasibility and safety. Instead of treating the problem in a binary manner, feasible vs infeasible, MVP tries to find a path that violates constraints the least when absolute fulfillment is not achievable [15]. The concept of MVP merged with RRT* is described in [16]. It combines the abilities of MVP to satisfy possible constraints while still finding feasible solutions and the ability of RRT* to effectively search in a complex environment and result in an MVP-RRT* algorithm described in Algorithm 3. A new Steer function is added in contrast to Algorithm 2. The Steer function returns a path between two states considering vehicle kinematics and constraints satisfaction if such a path exists.

Algorithm 3 MVP-RRT*

Input: q_0 - initial configuration, N - number of states, $\mathcal{K}$ - Kripke structure
Output: Kripke structure $\mathcal{K}$ with N states

```
K.init(q_0);
while K.count_states() < N do
  q_new ← Rand();
  Q_near ← Near(K, q_new);
  q_min ← null;
  min_cost ← ∞ ;
  for q_near ∈ Q_near do
    if Steer(q_near, q_new) ≠ ∅ then
      c' ← Cost(q_near) + c(q_near, q_new);
      if c' ≺ min_cost then
        q_min ← q_near;
        min_cost ← c';
      end if
    end if
  end for
  K.add_transition(q_min, q_new, min_cost);
  K.add_state(q_new);
  for q_near ∈ Q_near \ {q_min} do
    if Steer(q_near, q_new) ≠ ∅ then
      c' ← Cost(q_new) + c(q_new, q_near;
      if c' ≺Cost(q_near) then
        q_parent ← K.get_parent(q_near);
        K.remove_transition(q_parent, q_near);
        K.add_transition(q_new, q_near, c');
      end if
    end if
  end for
end while
return K;
```

Chapter 3

Implementation

3.1 Problem definition

The aim of this work is to implement a local path planning algorithm that utilizes polynomial paths under kinematic constraints caused by vehicle motion and the environment (adhesion coefficient). Regarding polynomial paths, several requirements had to be considered. Namely what degree of a polynomial is required to have a smooth path, how to describe the polynomial and how to apply curvature limitation on the path generation. And all that while still maintaining effectiveness so that the algorithm can be ran real-time.

3.2 Vehicle model selection

To apply kinematic constraints defined by the vehicle, the vehicle needs to be modeled somehow. The most suitable model for this matter is the kinematic model. The kinematic model captures a mathematical description of the vehicle's motion without considering the forces that affect it. The model uses a "bicycle" representation, effectively merging the left and right wheels into a single wheel at the center of each axle. A key assumption is that the velocity vectors of $\mathbf{v_f}$ and $\mathbf{v_r}$ align with the orientation of the front and rear wheels, respectively. This implies that there is no lateral slip at either wheel as the vehicle moves. This assumption is reasonable at lower speeds, where the tires

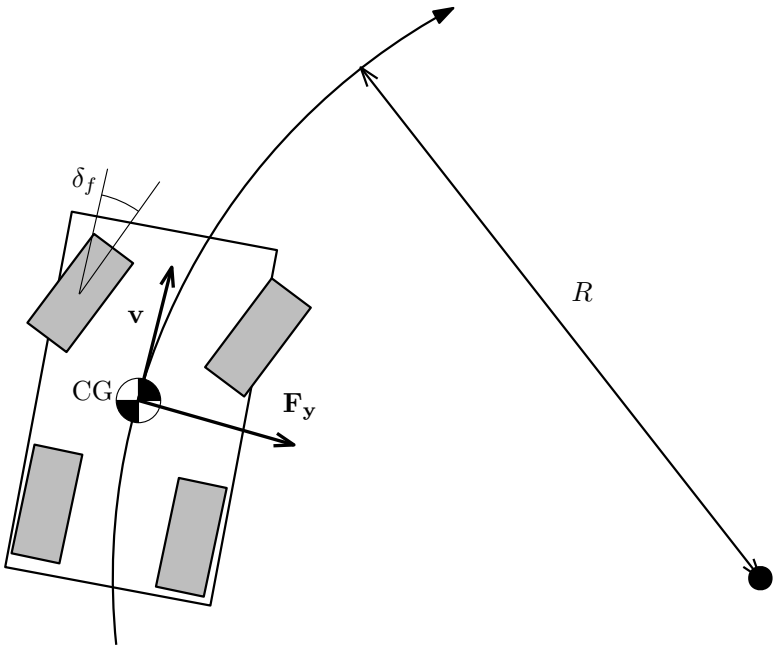


Figure 3.1: Lateral force acting as centripetal force.

generate only minimal lateral force.

The other variant of how the vehicles can be modeled is twin-track models, but they are way too complex for this subject as they work with vehicle dynamics. The kinematic model is suitable because it is relatively simple, yet it captures all the information that is needed for the maximum curvature constraint. That is, when the vehicle with mass m is driving on any circular road with radius R with a constant speed V, the total lateral force generated by all tires is

$$F_y = \frac{mV^2}{R}, \tag{3.1}$$

As demonstrated, the assumption about velocity vectors $\mathbf{v_f}$ and $\mathbf{v_r}$ remains valid at lower speeds and smaller steering angles because lateral force grows quadratically with speed and is inversely proportional to the vehicle's turning radius. However, at higher speeds and more aggressive maneuvers, the model no longer provides an accurate representation. Nevertheless, it still provides a reasonably good validation tool [17], [18]. Now, as stated in 3.1, this work aims to create polynomial paths under the vehicle's kinematic constraints and adhesion coefficient. The lateral force $\mathbf{F_y}$ is acting as a centripetal force, as seen in Figure 3.1, where CG is the center of gravity (c.g.), δ_f is the front wheel's steering angle and $\mathbf{v}$ is the velocity at vehicle's center of gravity. It was mentioned before that this force is caused by friction of the tires, which means that the vehicle turns because of friction. Friction force can be

described as

$$F_f = \sum_{i=1}^{4} F_{\text{wheel},\, i} = \mu F_N, \tag{3.2}$$

where μ is coefficient of friction (or adhesion coefficient) and F_N is normal force in the c.g. of the vehicle. Now, if the car has to make the turn and not slide off, the friction force F_f must be greater or equal, to the lateral force F_y. So now there is a relation between those two forces looking like this

$$F_f \geq F_y, \tag{3.3}$$

$$\mu F_N \geq \frac{mV^2}{R}. \tag{3.4}$$

Assuming that the vehicle is driving on a flat road and all its mass is distributed to the tires evenly, the normal force has a relation to the weight of the vehicle

$$F_N = F_g = mg, \tag{3.5}$$

where m is mass of the vehicle and g is gravitational acceleration with approximate value of $g = 9.81\,\mathrm{m\,s^{-2}}$.

Looking at this, the Equation 3.4 can be rewritten and simplified as

$$\mu mg \geq \frac{mV^2}{R}, \tag{3.6}$$

$$\mu g \geq \frac{V^2}{R}, \tag{3.7}$$

which takes out the mass of the vehicle from the equation. Before adding curvature to the equation, it needs to be said, what curvature is. Curvature κ is a measure of how sharply a curve bends at a given point, so the larger the curvature, the sharper the turn. Equivalently, it can be understood as the reciprocal of the radius of the osculating circle—the circle that best approximates the curve near this point. That yields this equation

$$\kappa = \frac{1}{R}. \tag{3.8}$$

Equation 3.8 allows curvature to be included in the constraints in Equation 3.7. Curvature can be then expressed as a function of μ

$$\kappa \leq \frac{\mu g}{V^2}, \tag{3.9}$$

and maximum curvature is limited by

$$\kappa_{max} = \frac{\mu g}{V^2}. \tag{3.10}$$

This will be employed to generate a path under curvature constraints using a "surface mask" with estimated coefficients of friction of the road ahead of the vehicle. As can be seen in Equation 3.10, the curvature is directly proportional to the friction coefficient of a surface the vehicle is driving, The lower the value of coefficient friction is, the lower the maximum curvature of polynomial paths generated can be.

3.3 Algorithm design and implementation

Using the theoretical foundations from preceding chapters, path planning algorithm modification is formulated in this section. From a computational time perspective, path planning must be very time efficient, which in the case of an RRT*-like algorithm means generating more samples and therefore a finer solution due to the properties of probabilistic completeness and asymptotic optimality. Modified MVP-RRT* with polynomial paths is presented in Algorithm 4. Only those methods specifically adapted for generating polynomials are discussed in the subsequent sections. The others are already greatly described in [1].

A goal bias has been added to Algorithm 4 in the selection of $q_r and$ to help navigate the tree towards the goal by using the RandomSample method (see Algorithm 6) to extend the node that is closest to the goal.

3.3.1 Modified steering method

Steering functions used in most of the RRT algorithm modifications are simple line segments. The downside is that the path provided by these algorithms must be smoothed in postprocessing using splines, Bézier curves or some other kind of optimization [19] for the sake of satisfying kinematic constraints. In contrast, using polynomials in the process of connecting new nodes to the tree provides a smooth path instantaneously, if the degree of polynomial is chosen accordingly.

In the case of this thesis, the algorithm is developed to be used on an autonomous vehicle platform. This means that the available state space variables are position in x [m], position in y [m] and heading ψ[rad] $\in [-\pi, \pi)$ which is the angle that the direction of travel of the vehicle makes with the

Algorithm 4 Modified MVP-RRT*

Input: q_0 - initial configuration, N - number of vertices, $\mathcal{T}$ - tree, p_{goal} - goal bias, q_{goal} - goal configuration
Output: tree $\mathcal{T}$ with N nodes

```
T.init(q_0);
while T.count_nodes() < N do
    r ← RandomUniform(0, 1);
    if r < p_goal then
        q_rand ← q_goal;
    else
        q_rand ← Rand();
    end if
    Q_near ← Near(T, q_rand);
    q_parent ← SelectNode(Q_near);
    q_new ← RandomSample(q_parent);
    if PolySteer(q_parent, q_new) ≠ ∅ then
        cost ← T.get_cost(q_parent) + c(q_parent, q_new);
        T.add_edge(q_parent, q_new, cost);
        T.add_node(q_new);
        Q_near ← Near(T, q_new);
        for q_near ∈ Q_near \ q_parent do
            Rewire(T, q_new, q_near);
        end for
    end if
end while
return T;
```

x-axis. A possible look of some configuration q_i is

$$q_i = (x_i, y_i, \psi_i). \tag{3.11}$$

Now the polynomial curve of a degree n can be either parametrized in a form of

$$x(s) = a_0 + a_1 s + a_2 s^2 + \cdots + a_n s^n, \tag{3.12}$$
$$y(s) = b_0 + b_1 s + b_2 s^2 + \cdots + b_n s^n, \tag{3.13}$$
$$\tag{3.14}$$

or it can be defined explicitly as y being a function of x or otherwise as x being a function of y

$$y(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n, \tag{3.15}$$

or

$$x(y) = a_0 + a_1 y + a_2 y^2 + \cdots + a_n y^n. \tag{3.16}$$

From now on, everything written about the polynomial in the form of $y(x)$ is valid for $x(y)$ too. An explicit definition of a polynomial curve was chosen

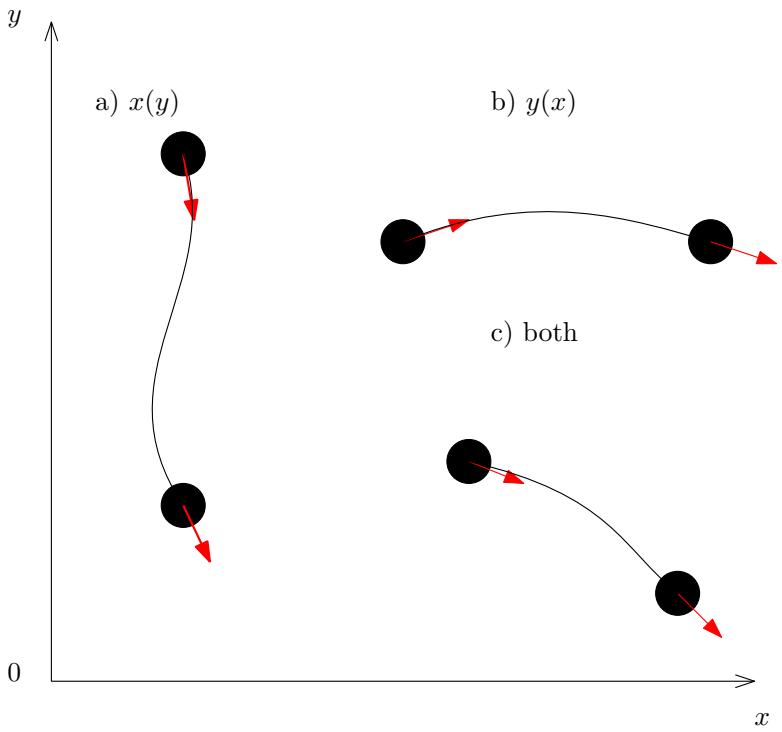


Figure 3.2: Example of configurations which can be connected by only one of the polynomials in a) and b) and a configuration which can be connected by both.

because compared to the parametric it is much easier to handle its derivatives which are used to calculate the heading from the slope of the polynomial. Both definitions $x(y)$ and $y(x)$ have to be considered because each segment of the path has to be a graph of either function $y(x)$ or $x(y)$. Sometimes one of the polynomials isn't able to connect the two configurations, but the other one solves it as seen in an example in Figure 3.2.

Because the RRT* algorithm uses rewiring, the polynomial must be able to connect two configurations q_s and q_g with known headings. The lowest-degree polynomial which can connect such configurations is the third-degree polynomial

$$y(x) = ax^3 + bx^2 + cx + d. \tag{3.17}$$

To get coefficients of a polynomial which connects two configurations q_s and q_g, as seen on Figure 3.3, four linear equations with 4 variables emerge

$$y_s = ax_s^3 + bx_s^2 + cx_s + d, \tag{3.18}$$
$$y_g = ax_g^3 + bx_g^2 + cx_g + d, \tag{3.19}$$
$$\tan(\psi_s) = y'(x_s) = 3ax_s^2 + 2bx_s + c, \tag{3.20}$$
$$\tan(\psi_g) = y'(x_g) = 3ax_g^2 + 2bx_g + c.. \tag{3.21}$$

Equation 3.20 and 3.21 are different for $x(y)$ due to the fact that its coordinate

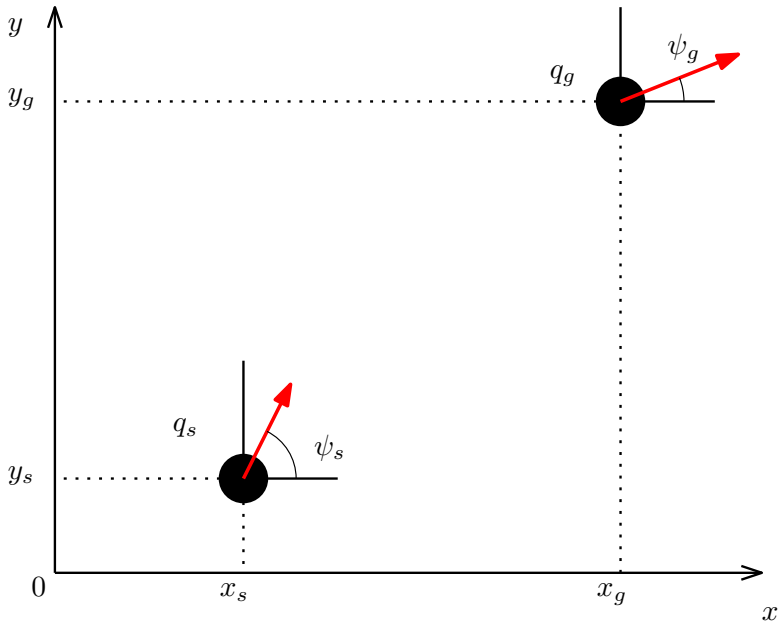


Figure 3.3: Configurations q_s and q_g in state space.

system has a left-handed orientation but is based on the same principle. The slope a of a curve is given by its first derivative as well as a tangent function of heading angle ψ

$$a = \frac{dy}{dx} = \tan(\psi). \tag{3.22}$$

In the case of $x(y)$ $\pi/2$rad has to be either added or subtracted, before calculating the slope of the curve from the heading angle.

While establishing a polynomial path between two nodes is a key step, it is equally important to evaluate the curvature along the resulting path. The formula for calculating the curvature of a two-dimensional parametrized curve $c(t) = (x(t), y(t))$ is

$$\kappa = \frac{x'y'' - y'x''}{(x'^2 + y'^2)^{3/2}}, \tag{3.23}$$

as described in [20]. Graphs of functions $y(x)$ and $x(y)$ are just special cases of parametrized curve in the form

$$x = t, \tag{3.24}$$
$$y = y(t), \tag{3.25}$$

which is then used in Equation 3.23. With the first derivative of x being 1 and the second derivative of x being 0 the formula simplifies to

$$\kappa = \frac{|y''|}{(1 + y'^2)^{3/2}}. \tag{3.26}$$

After clarifying how the polynomials are expressed and how to check their curvature, it is appropriate to introduce the PolySteer method in Algorithm 5. It requires two configurations as input and outputs a polynomial path connecting them respecting maximum curvature constraint based on a friction coefficient. First, it determines whether to use $y(x)$ or $x(y)$ expression based on headings in q_{start} and q_{goal} configurations in the DetermineAxis method. Thanks to the sampling function, one of the variations, either $y(x)$ or $x(y)$, should be applicable almost every time.

After the correct polynomial representation is chosen, axis and both configurations are passed to a GetCoefficients method. This function calculates the coefficients of the polynomial by solving a system of four linear equations with four variables (can be seen in Equations 3.18–3.21).

Following that, the GeneratePath method creates a sequence of configurations from q_{start} to q_{goal}. Additionally, it guarantees that the path's maximum curvature, as computed using Equation 3.26, stays below the threshold imposed by surface friction in Equation 3.10. It is worth mentioning, that the velocity along the path is considered to be constant, but its value can be set either manually, or it can be passed on as an optional input to the PolySteer method and then used to recalculate maximum curvature accordingly.

Algorithm 5 PolySteer method used in Algorithm 4

Input: q_{start} - initial configuration, q_{goal} - goal configuration, $mask$ - surface friction mask
Output: path $\mathcal{P}$ between q_{start} and q_{goal}
$axis \leftarrow$ DetermineAxis(q_{start}, q_{goal});
$coeffs \leftarrow$ GetCoefficients$(q_{start}, q_{goal}, axis)$;
$\mathcal{P} \leftarrow$ GeneratePath$(q_{start}, q_{goal}, axis, coeffs, mask)$;
return $\mathcal{P}$;

3.3.2 Modified sampling method

With the intention of implementing polynomial paths to the original algorithm not only the steering function must be modified, but it is also necessary to implement a new sampling function, as the polynomial steering method needs boundary points. Each segment of the path must begin at a specified initial configuration and end at a specified goal configuration.

In the process of adding a new node, the initial configuration is given as q_{parent}, but q_{new} needs to be sampled, as can be seen in Algorithm 4. While the sampling procedure must preserve randomness, it must also ensure that

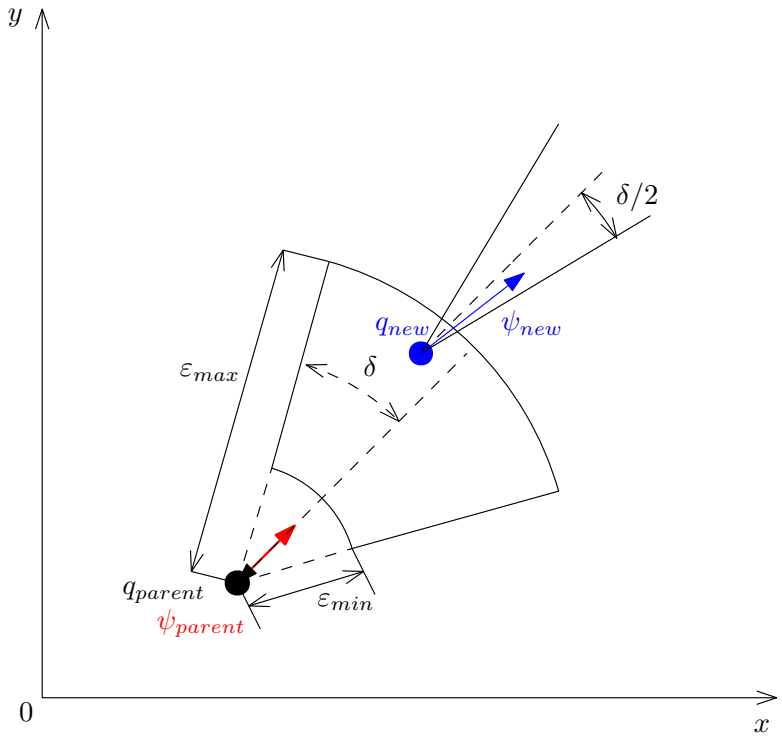


Figure 3.4: Illustration of how does sampling with RandomSample work.

the random sample remains feasible using the polynomial steering function. Generating samples that are not feasible from the randomly selected q_parent means a huge drop-off in computational efficiency. Therefore additional sampling method RandomSample is introduced in Algorithm 6.

It needs a configuration q as an input and returns a new randomly sampled configuration q_{new}, which should be feasible. It is best explained with an illustration in Figure 3.4. First, the heading from the parent is obtained so that the initial direction of sampling is given. Then random distance ε from uniform distribution and random heading from normal distribution are acquired to calculate the random sample configuration q_{new}, which is checked if it lies in the bounds of configuration space. In the next step, the heading of q_{new} is retrieved from another normal distribution and assigned to q_{new}, after that q_{new} is returned.

Algorithm 6 RandomSample method used in Algorithm 4

Input: q_{parent} - configuration for expansion
Output: q_{new} - new node to be added to the $\mathcal{T}$

$\psi_{parent} \leftarrow (q_{parent})$get_heading;
$\varepsilon \leftarrow$ RandomUniform$(\varepsilon_{min}, \varepsilon_{max})$;
$\psi_{new} \leftarrow$ RandomNormal(ψ_{parent}, δ);
$q_{new} \leftarrow$ CreateSample$(q_{parent}, \varepsilon, \psi_{new})$;
$\psi_{new} \leftarrow$ RandomNormal $(\psi_{parent}, \delta/2)$;
(q_{new}).add_heading(ψ_{new})
return q_{new};

Chapter 4

Validation and comparison to benchmark on testing scenarios

The modified algorithm with polynomial paths and Marek's original algorithm are compared in Section 4.1 and tested on scenarios adopted from [1] because of the known and desired behavior of the benchmark. All scenarios are conducted within a 4 m by 4 m square occupancy grid located directly in front of the vehicle with x being from range $[0, 4]$ and y from range $[-2, 2]$. This area represents a variously classified surface with purple being drivable and yellow being undesired for driving, e.g. grass or a sidewalk.

Then in Section 4.3, a comparison between surface masks with different surface friction coefficient values is shown and compared with the case of the coefficient value being constant $\mu = 1$ everywhere. The surface mask containing information about the friction coefficient of a given surface can be of any size and is mapped on the aforementioned area. It is worth mentioning, that as discussed in Algorithm 5, velocity along the path is considered to be constant and for this benchmark is set to $V = 2\mathrm{m\,s}^{-1}$. Goal biasing was also utilized in 4 and for this benchmark, it was set to a value of $p_{goal} = 0.2$.

4.1 Computational efficiency

Both algorithms were deployed on a laptop with a Ryzen 7 Pro 4750U processor and 32 GB of RAM. The resulting comparison between Marek's algorithm

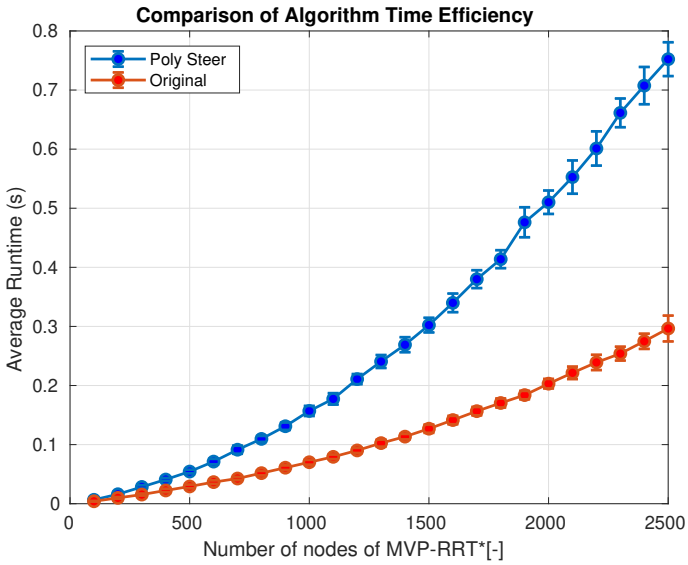


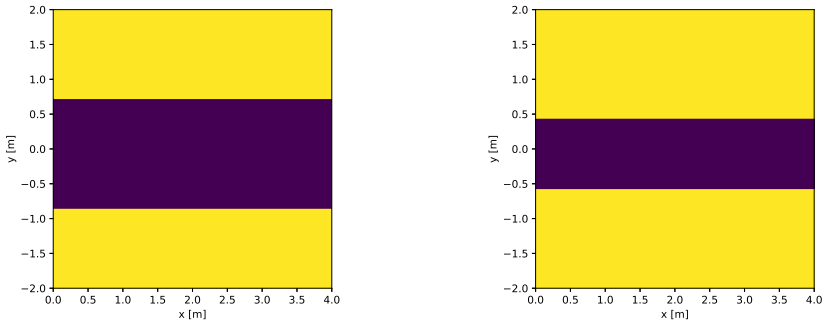
Figure 4.1: Comparison of time needed to create a tree of n nodes.

and the algorithm introduced in this thesis is shown in Figure 4.1. In this comparison, both algorithms were executed for 50 iterations each, generating trees with 100, 200, 300, and up to 2500 nodes. Even though generating polynomials should be computationally cheap, the result in Figure 4.1 speaks against it. It looks like the curvature constraint affects the performance quite heavily. During the creation of the RRT*s with over 2000 nodes, maximum curvature is exceeded in more than 1 million cases. This is happening mostly in the phase of rewiring in densely covered areas. In other areas, such as small unmanned aircraft, the polynomial steering function had much worse results, than the approach using optimization methods [2].

4.2 Testing scenarios

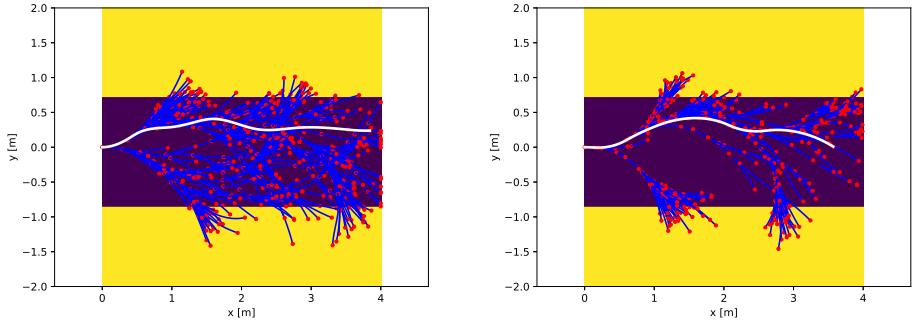
Based on the comparison in Section 4.1, for this section, Marek's algorithm is creating a tree of 2500 nodes and the modified algorithm with polynomial paths is creating a tree with 1500 nodes, as they have roughly the same time requirements. This should put the modified algorithm at a disadvantage because a feature of both these algorithms is that the more nodes an RRT* is able to generate, the bigger the probability that the found solution is optimal.

There are three scenarios in which the growth of the tree and the generated path will be compared. In each of these scenarios, the starting position is at (0, 0) with zero heading. To take collisions into account, the underlying



(a) : Mask without dilation. **(b) :** Mask with dilation.

Figure 4.2: Straight road scenario



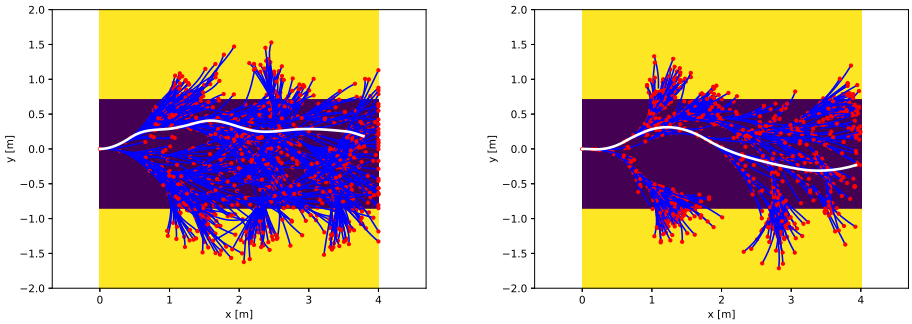
(a) : 500 nodes, polynomial paths. **(b) :** 500 nodes, Marek's algorithm.

Figure 4.3: Straight road scenario, red dots - nodes, blue lines - paths, white line - the best path

masks are dilated. The first scenario is a straight road, as seen in Figure 4.2. Even though it looks easy, the algorithms might struggle there, creating a meandering path. The second scenario can be seen in Figure 4.6 and it is a 90-degree turn. The last one is a zigzag turn shown in Figure 4.10a.

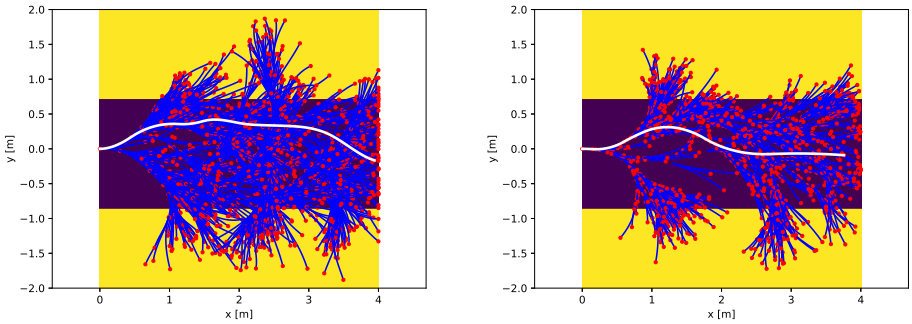
4.2.1 Straight road

This scenario is the simplest one, it is a straight passage with the goal being at (4, 0). However, keeping a straight line is a rather challenging task for these algorithms which is seen right in the first phase of the tree growth in Figure 4.3. Neither of them can maintain a straight line and both oscillate, the second phase in Figure 4.4 is almost the same, but the resulting path in Figure 4.5b keeps a straight line when approaching the goal.



(a) : 1000 nodes, polynomial paths. **(b) :** 1000 nodes, Marek's algorithm.

Figure 4.4: Straight road scenario, red dots - nodes, blue lines - paths, white line - the best path

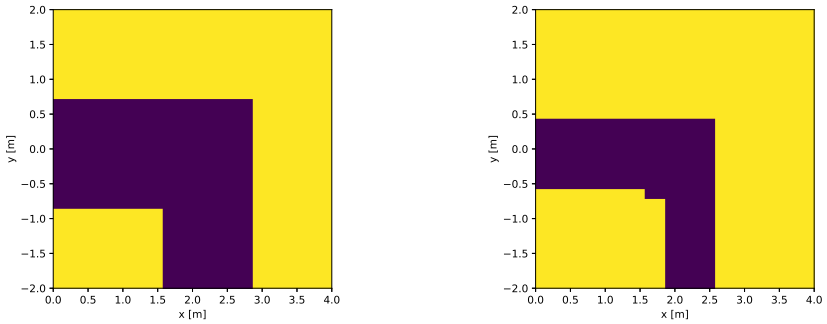


(a) : 1500 nodes, polynomial paths. **(b) :** 2500 nodes, Marek's algorithm.

Figure 4.5: Straight road scenario, red dots - nodes, blue lines - paths, white line - the best path

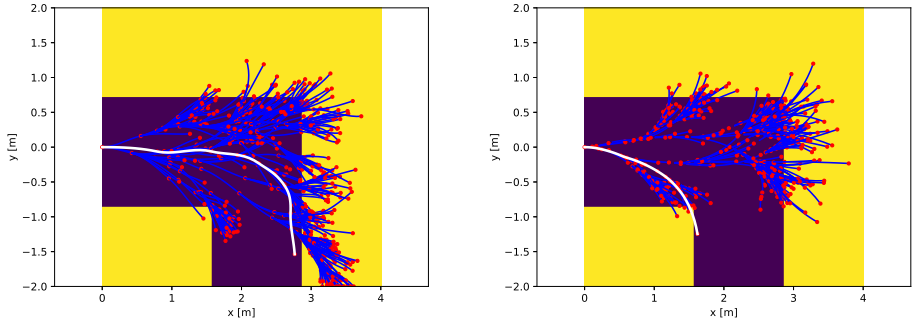
4.2.2 90 degree turn

This scenario tests the algorithms' ability to make a 90-degree turn. The goal is situated at point (2.4, -2) and the vehicle starts at (0, 0) with zero heading. Looking at both trees in Figure 4.7, Figure 4.7a is biasing more towards the goal already in the early stages of the growth. However, both algorithms improve the path in Figure 4.8 and appear quite similar. Both algorithms demonstrated the ability to make sharp turns.



(a) : Mask without dilation. **(b) :** Mask with dilation.

Figure 4.6: 90 degree turn scenario



(a) : 500 nodes, polynomial paths. **(b) :** 500 nodes, Marek's algorithm.

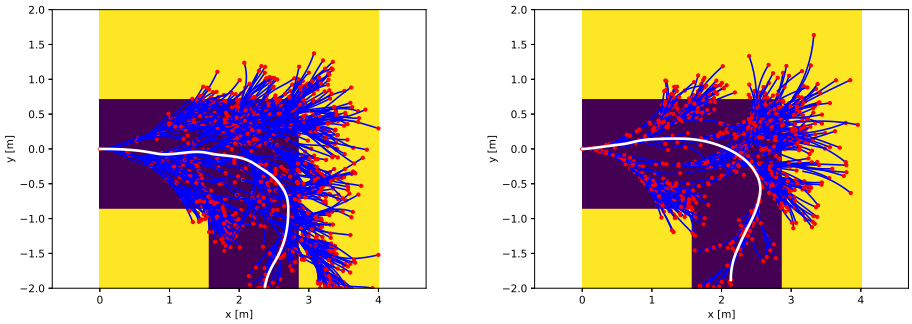
Figure 4.7: 90 degree turn scenario, red dots - nodes, blue lines - paths, white line - the best path

4.2.3 Zigzag turn scenario

In this scenario, the vehicle has to go around two consecutive corners. In Figure 4.11 both of these algorithms go over an undesired surface, yet the direction of growth looks promising. In the end, both manage to find a path to goal and are able to come through and stay on the road.

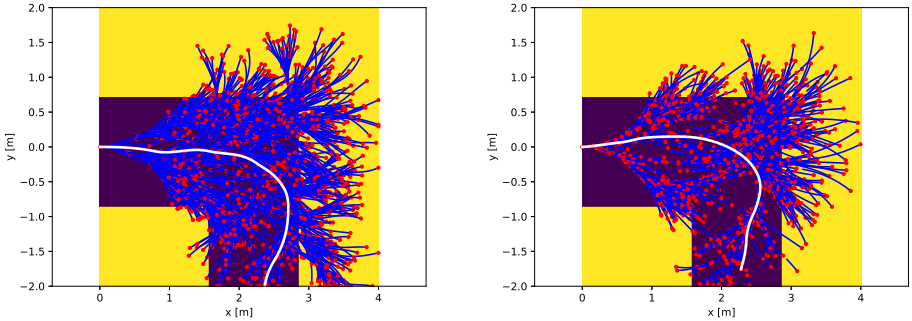
4.3 Planning with adhesion coefficient constraints

This section will explore the effects of changes in friction between a surface and a tire, which limit a vehicle's ability to turn at a specific speed. Consider a scenario where an autonomous vehicle is approaching a turn, but the presence



(a) : 1000 nodes, polynomial paths. **(b) :** 1000 nodes, Marek's algorithm.

Figure 4.8: 90 degree turn scenario, red dots - nodes, blue lines - paths, white line - the best path

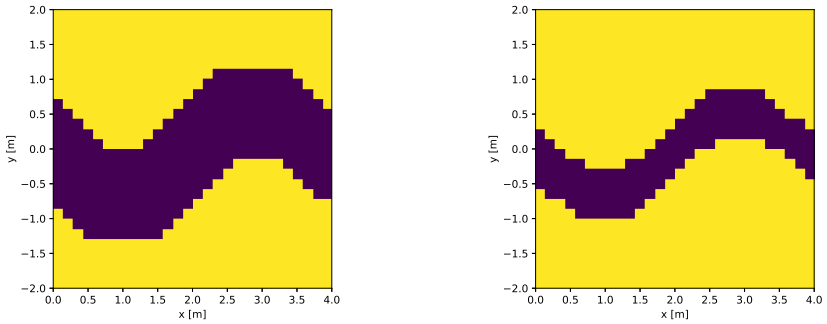


(a) : 1500 nodes, polynomial paths. **(b) :** 2500 nodes, Marek's algorithm.

Figure 4.9: 90 degree turn scenario, red dots - nodes, blue lines - paths, white line - the best path

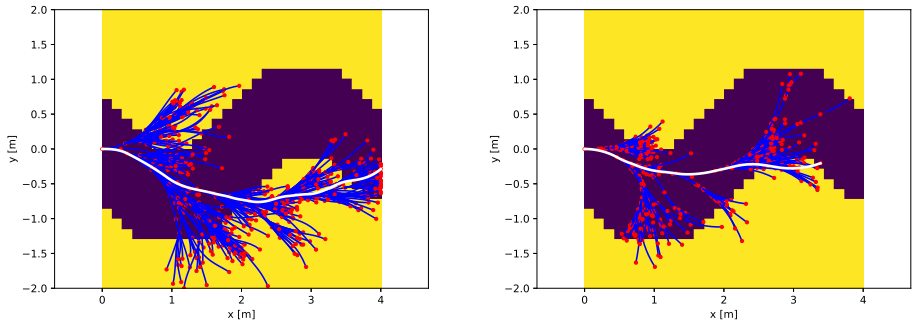
of snow requires the vehicle to respond accordingly and plan its motion. That is where the polynomial steering function can be applied together with the curvature constraint.

The mask with the snow is depicted in Figure 4.14 and the adhesion coefficient of snow is $\mu_s = 0.4$. The road is assumed to be made of concrete and dry concrete has an adhesion coefficient $\mu_c = 0.8$ [21]. Growth of the tree under constraints caused by snow with a lower adhesion coefficient than the rest of the road can be seen in Figure 4.15. In Figure 4.15a it is visible, that the paths crossing the snowy area do not have large curvature, they are rather straight. Then the tree grows more and finds a path with a relatively large turning radius in Figure 4.15b. Now comparing the final path in Figure 4.16 to the path obtained with no snow in the way in Figure 4.9a, it has a larger turning radius resulting in what seems like a much smoother path, than the one generated for dry road. The drawback is, that the maximum curvature restriction imposes even greater demands on the polynomial steering function



(a) : Mask without dilation. **(b) :** Mask with dilation.

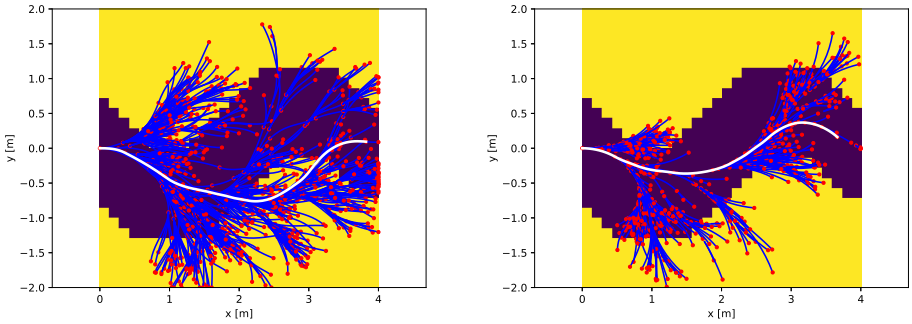
Figure 4.10: Zigzag turn scenario



(a) : 500 nodes, polynomial paths. **(b) :** 500 nodes, Marek's algorithm.

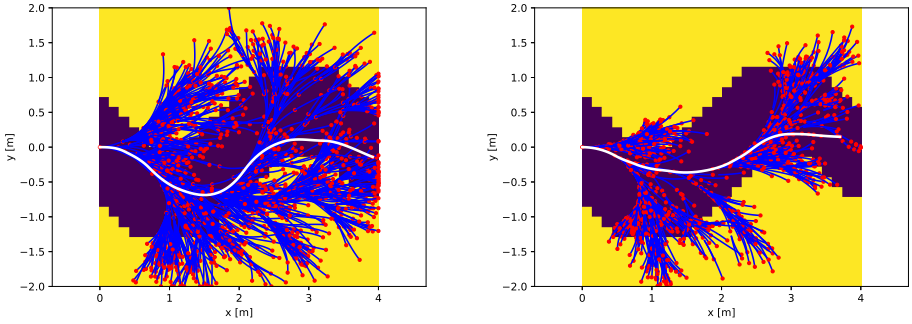
Figure 4.11: Zigzag turn scenario, red dots - nodes, blue lines - paths, white line - the best path

and sampling method respectively. Expressed in numbers, the average time needed to construct a tree with 1500 nodes when the road is dry is 0.3 seconds. However, once the adhesion coefficient decreases, the computational time rises. In the case of snowy conditions with an adhesion coefficient of 0.4, the computational time is about 0.45 seconds on average.



(a) : 1000 nodes, polynomial paths. **(b) :** 1000 nodes, Marek's algorithm.

Figure 4.12: Zigzag turn scenario, red dots - nodes, blue lines - paths, white line - the best path



(a) : 1500 nodes, polynomial paths. **(b) :** 2500 nodes, Marek's algorithm.

Figure 4.13: Zigzag turn scenario, red dots - nodes, blue lines - paths, white line - the best path

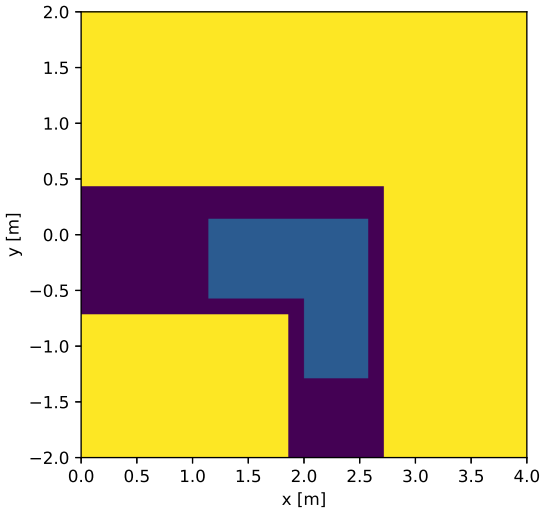
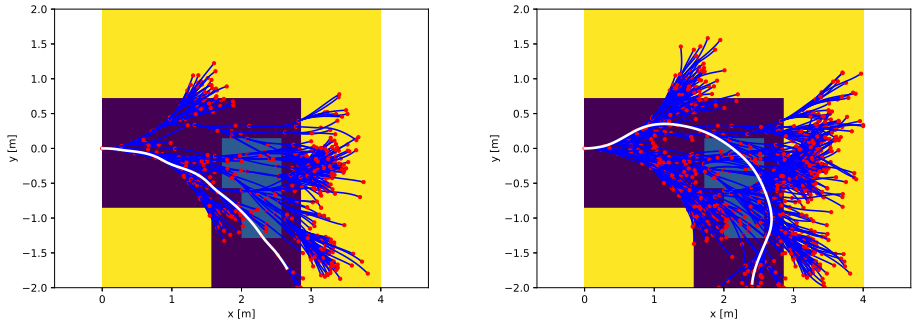


Figure 4.14: 90 degree turn scenario with snow ($\mu_s = 0.4$) in the corner.



(a) : 500 nodes, polynomial paths, snow on the road ($\mu_s = 0.4$).

(b) : 1000 nodes, polynomial paths, snow on the road ($\mu_s = 0.4$).

Figure 4.15: 90 degree turn scenario with snow ($\mu_s = 0.4$) in the corner, red dots - nodes, blue lines - paths, white line - the best path

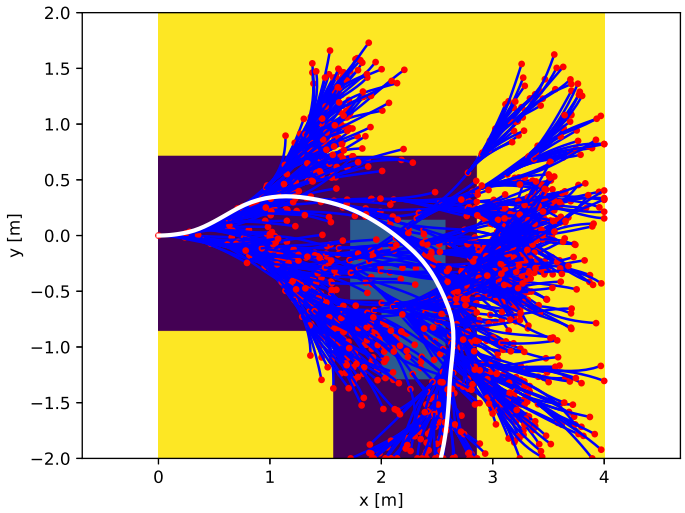


Figure 4.16: 90 degree turn scenario with snow ($\mu_s = 0.4$) in the corner.

Chapter 5

Conclusion

The goal of this thesis was to modify a path-planning algorithm for autonomous vehicles with the use of polynomial paths. The vehicle kinematic model was used to implement constraints to path generation while also allowing to account for changes in the surface driven by the vehicle.

Despite the computational efficiency of generating polynomials connecting two points, constraints imposed by both the vehicle itself and the driven surface caused a massive drop-off in the proposed algorithm performance. As a result of this, the developed algorithm was almost two times slower than its benchmark during the creation of a tree with 1000 nodes. With the growing number of nodes, the difference in performance of the compared algorithms increased even more as with 2500 nodes the time needed was two and a half times larger.

Because computational time was the limiting factor, trees generated in the testing scenarios had roughly the same runtime of 0.3 seconds. That resulted in a various number of nodes in each tree. Tree with polynomial paths had only 1500 whereas the benchmark algorithm was able to generate a tree with 2500 nodes. Despite the substantial difference in the tree size, paths generated by both algorithms were of the same quality in the sense of smoothness and the ability to reach the goal.

The proposed algorithm's ability to consider a surface friction coefficient was tested and led to promising results. A noticeable change in path generation occurred whenever a segment of the road was covered by a surface with a lower friction coefficient than the rest of the road. However, the curvature limitations

inflicted by the change in surface meant another rise in computational time.

For future work a sampling method that determines a region where it samples by the friction coefficient might be introduced. This would possibly help with generating fewer samples that are outside of a feasible region given the curvature limitation.

Appendix A

Bibliography

[1] M. Boháč, "Mission planning system for autonomous vehicle," Master's thesis, Czech Technical University in Prague, 2022.

[2] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Robotics Research: The 16th International Symposium ISRR*, pp. 649–666, Springer, 2016.

[3] D. Vosahlik, J. Cech, T. Hanis, A. Konopisky, T. Rurtle, J. Svancar, and T. Twardzik, "Self-supervised learning of camera-based drivable surface friction," in *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pp. 2773–2780, IEEE, 2021.

[4] T. T. Mac, C. Copot, D. T. Tran, and R. De Keyser, "Heuristic approaches in robot path planning: A survey," *Robotics and Autonomous Systems*, vol. 86, pp. 13–28, 2016.

[5] K. Karur, N. Sharma, C. Dharmatti, and J. E. Siegel, "A survey of path planning algorithms for mobile robots," *Vehicles*, vol. 3, no. 3, pp. 448–468, 2021.

[6] C. Liu, S. Lee, S. Varnhagen, and H. E. Tseng, "Path planning for autonomous vehicles using model predictive control," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 174–179, IEEE, 2017.

[7] M. Reda, A. Onsy, A. Y. Haikal, and A. Ghanbari, "Path planning algorithms in the autonomous driving system: A comprehensive review," *Robotics and Autonomous Systems*, vol. 174, p. 104630, 2024.

[8] E. DIJKSTRA, "A note on two problems in connexion with graphs.," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[9] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[10] A. K. Guruji, H. Agarwal, and D. Parsediya, "Time-efficient a* algorithm for robot path planning," *Procedia Technology*, vol. 23, pp. 144–149, 2016.

[11] P. Typaldos, M. Papageorgiou, and I. Papamichail, "Optimization-based path-planning for connected and non-connected automated vehicles," *Transportation Research Part C: Emerging Technologies*, vol. 134, p. 103487, 2022.

[12] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[13] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[14] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[15] L. I. R. Castro, P. Chaudhari, J. Tumova, S. Karaman, E. Frazzoli, and D. Rus, "Incremental sampling-based algorithm for minimum-violation motion planning," in *52nd IEEE Conference on Decision and Control*, pp. 3217–3224, IEEE, 2013.

[16] T. Wongpiromsarn, K. Slutsky, E. Frazzoli, and U. Topcu, "Minimum-violation planning for autonomous systems: Theoretical and practical considerations," in *2021 American Control Conference (ACC)*, pp. 4866–4872, IEEE, 2021.

[17] R. Rajamani, *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

[18] D. Schramm, M. Hiller, and R. Bardini, *Vehicle Dynamics: Modeling and Simulation*. Springer Berlin Heidelberg, 2014.

[19] A. Ravankar, A. A. Ravankar, Y. Kobayashi, Y. Hoshino, and C.-C. Peng, "Path smoothing techniques in robot navigation: State-of-the-art, current and future challenges," *Sensors*, vol. 18, no. 9, p. 3170, 2018.

[20] E. Kreyszig, *Differential geometry*, vol. 11. Courier Corporation, 1991.

[21] S. Evtukov and E. Golov, "Adhesion of car tires to the road surface during reconstruction of road accidents," in *E3S Web of Conferences*, vol. 164, p. 03022, EDP Sciences, 2020.