Planning and Navigation of Mobile Robots

Independent Study Report

Spring 2022 - Under Prof. Madhava Krishna

Avneesh Mishra
avneesh.mishra@research.iiit.ac.in *

May 10, 2022

This report describes the things learned in the Independent Study during the Spring 2022 session. The material is available on GitHub at TheProjectsGuy/IS_RPN22-EC9.404.

Project Description

This project aims to study mobile robots in the context of planning and navigation in known and unknown environments. The aim is to explore aspects of mobile robotics: Motion models, trajectory generation, optimization, planning, and tracking. Overall, the study aims to discover the underlying mechanics of autonomous mobile systems. The study seeks to emulate the course outcomes of Robotics: Planning and Navigation (EC4.403) while exploring as much additional material as possible.

Contents

*M.S. by research - CSE - 2021701032

List of Figures

1 Summary

1.1 Robotic Planning and Navigation - EC4-403

This is a course that the independent study tried to follow. The following was completed from this course

- Assignment 1: Motion planning using RRT. Official submission at Robotics-Planning-Navigation.
- Assignment 2: Bernstein polynomials. Official submission at Robotics-Planning-Navigation.
- Assignment 3: Time scaling, collision cone. Official submission at Robotics-Planning-Navigation.

Additionally, lecture videos were also seen (as long as they were online and recorded). Lectures videos from `Lecture 3` through `Lecture 14` were seen for this study. The following concepts were covered through watching the recorded lectures

> Visibility Graph - Forward and Inverse Kinematics of differential drive robots - UAV motion control - Model Predictive Controller - Collision checking as an optimization problem - Bernstein polynomials - Time scaling - Collision cone and Velocity obstacle - Time scaled collision cone - Inverse Velocity obstacle

The assignments are described in more detail in section 3. A more detailed report for the individual assignments can be found at TheProjectsGuy/IS_RPN22-EC9.404.

1.2 Extra Material

The following was covered by referring to online sources [1]

> Unicycle and bicycle model - wheel encoders - creating a dashboard using Plotly - RRT - **Dijkstra** from scratch - navigating indian cities using **OSMNX** - Trajectory generation using cubic and quintic spirals - tracking trajectories using **pure pursuit and stanley trackers** - Collision checking using circles and convex hulls (swath) - **dynamic window avoidance**

The above topics are also included in the repository and are omitted from this report.

[1] AP102 - Motion planning and Path tracking - Naveen Arulselvan

2 Learning Objectives

The following learning objectives were achieved through this independent study

- Modeling and control of different vehicle models
- Global path planning using sampling and search-based planners
- Local trajectory planning using third and fifth order spirals, Bezier curves (Bernstein polynomials), and other methods
- Local trajectory tracking using optimization techniques and various trackers
- Collision avoidance using time scaling, optimization constraints, and other methods
- Solve the assignments of course `Robotic Planning and Navigation - EC4.403`

3 Robotics Planning and Navigation

3.1 Assignment 1: Motion planning using RRT

A detailed report is present at TheProjectsGuy/IS_RPN22-EC9.404.

RRT Algorithm

The RRT algorithm has the following steps

1. **Define map**, free space, and the problem: This will be a map having the start, stop and the free space information of the environment (including a list of obstacles).
2. Keep **sampling** in free space. Store nodes in a tree (can be modeled as a list with links). Each node in the list has x, y, a parent (index), children (indices), and a v, ω command (to reach it from the parent). We start at the starting point, with zero commands and NULL parent.
3. When a new point in free space is sampled, the closest point from the nearest node (in the tree) and in the allowed distance is obtained. The control commands to this point is obtained. The node is added (as a child and to the list) if the path from the nearest node to this node is free of collisions.
4. We repeat the above step will the most recently added node (at the bottom of list) is within a certain bounds of the goal.

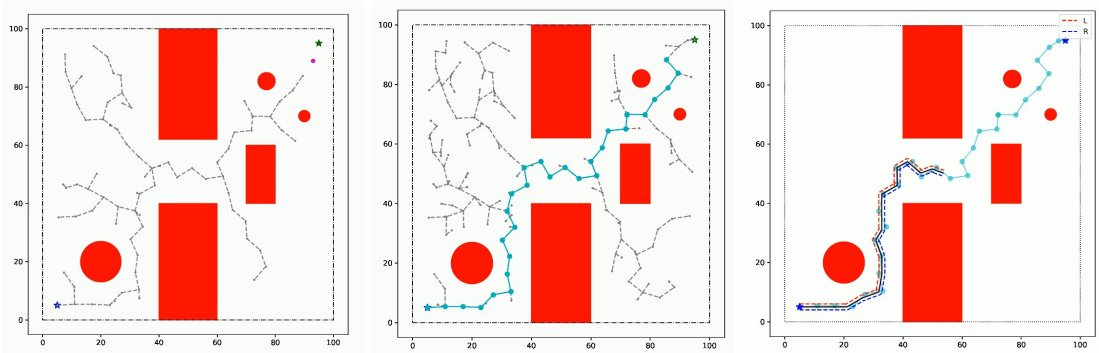
The following implementations were done

- Generating a path assuming that a holonomic robot is going to travel it (do not store v, ω), and then make a non-holonomic robot travel the path. The robot will either go straight or turn at nodes.

  This does not exploit the non-holonomic structure of the vehicle and may not be practical for larger robots. See figure 1 for results using this method.

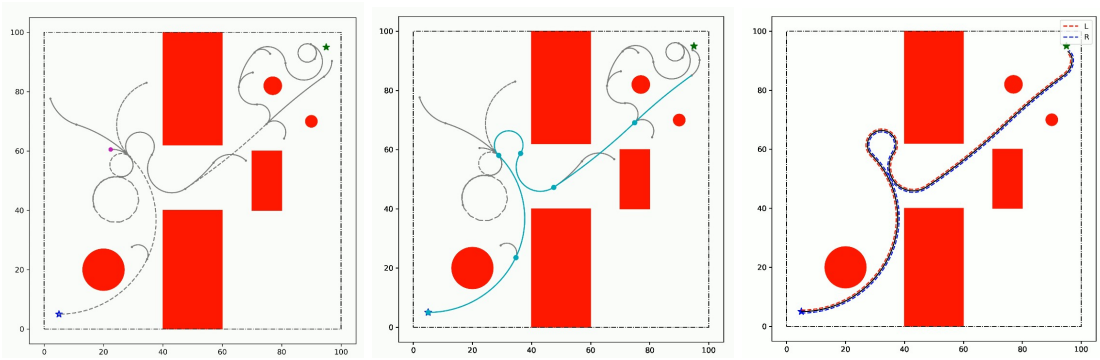
- Exploiting the non-holonomic structure of the vehicle when adding nodes. This has the advantage of creating paths and routes that can inherently be traversed by the vehicle.

  This involves either directly sampling in the vehicle's control space or running inverse kinematics on the points sampled in the state space (free map). See figure 2 for results using this method.



(a) Sampling (b) Path (c) Traveling

Figure 1: RRT: Holonomic sampling and non-holonomic travel
The sampling and path are that of a holonomic model. The non-holonomic robot navigates this using a sequence of straight line travels and turning commands.



(a) Sampling (b) Path (c) Traveling

Figure 2: RRT: Non-holonomic sampling and travel
The sampling, path generation and traveling is of a differential drive robot (non-holonomic robot).

3.2 Assignment 2: Bernstein Polynomials

A detailed report is present at TheProjectsGuy/IS_RPN22-EC9.404.

3.2.1 Bernstein Polynomials

A Bernstein polynomial is a linear combination of Bernstein basis polynomials. The $n+1$ Bernstein basis polynomials of degree n are given by the following equation.

$$b_{v,n}(x) = \binom{n}{v} x^v \, (1-x)^{n-v}, \quad v = 0, \ldots, n \tag{3.1}$$

For a continuous function f defined in the interval $[0, 1]$, the following bernstein polynomial can be considered as a viable approximation.

$$B_n(f)\,(x) = \sum_{v=0}^{n} f\left(\frac{v}{n}\right) b_{v,n}(x) \tag{3.2}$$

The above approximation is more accurate if $n \to \infty$, that is $\lim_{n\to\infty} B_n(f) = f$. The basis function values for $n = 5$ (fifth-degree) is shown in figure 3.

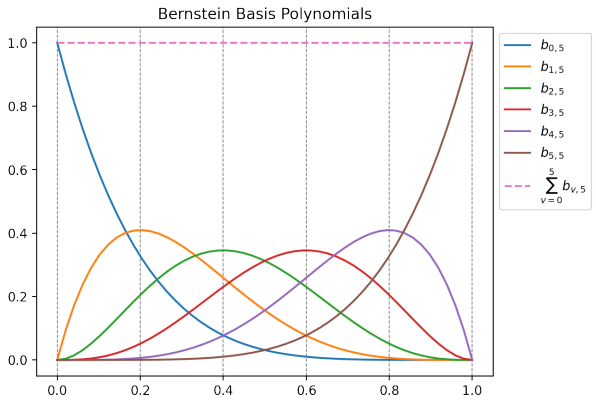


Figure 3: Bernstein Basis Values for $n = 5$
Fifth order bernstein basis values. Note how for different $x = {}^{v}/_{n}$ (where $v = 0, \ldots, n$), the corresponding $b_{v,n}(x)$ peaks, which shows the weight of the control point being considered. Here, the $b_{v,n}(x)$ values are weights and the $f\left({}^{v}/_{n}\right)$ could be theorized as control points (the above graph only has b).
This figure can be generated by running the `src/bernstein_basis.py` script as main.
Note: The sum of all bernstein basis polynomials at any instance x is 1.

3.2.2 Non-Holonomic Path using Bernstein Polynomials

In a non-holonomic system, the system loses its ability to freely move in space due to some physical limitations. A differential drive robot (or a car) cannot move sideways, its non-holonomic constrain stems from the way velocity is resolved along its center, that is

$$\dot{x} = v\cos(\theta) \quad \dot{y} = v\sin(\theta) \qquad \Rightarrow \dot{y} = \dot{x}\tan(\theta) \quad \Rightarrow y = \int \dot{x}\tan(\theta)\, dt \tag{3.3}$$

The above integral cannot be computed directly, only numerical solutions to it exits. The kinematic model of a differential drive robot is given by

$$\dot{q} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

In the above equations, the point x, y, θ is the robot position (in the 2D planar environment) and v, ω are the velocity commands given to the robot body.

Since the system above loses a degree of freedom - it cannot move in all possible directions in the neighborhood, in other words it is *not* holonomic - only two of x, y, θ can be independently modeled.

Problem Statement We are usually given the x, y, θ values at three points in time - the beginning t_o, waypoint t_w or t_c, and end t_f. We are also given the first derivatives of x at the time steps t_o, t_c, and t_f. We are also given $\dot{\theta}$ at time steps t_o and t_f. If this were holonomic (which it isn't), we could independently model the x, y, and θ as Bernstein polynomials and follow the given trajectory. But here, we have to use the constraint given by equation 3.3.

3.2.3 Finding W_{x_i} and W_{k_i}

We model the variable x as

$$x(t) \approx B_n(x)(t) = \sum_{i=0}^{5} W_{x_i} B_i(\mu(t)) \tag{3.4}$$

We model $\dot{x}$ as

$$\dot{x}(t) = \sum_{i=0}^{5} W_{x_i} \dot{B}_i(\mu(t)) \tag{3.5}$$

We get the constraints as

$$\begin{bmatrix} x_o \\ x_w \\ x_f \\ \dot{x}_o \\ \dot{x}_w \\ \dot{x}_f \end{bmatrix} = \begin{bmatrix} \sum_{i=0}^{5} W_{x_i} B_i(\mu(t_o)) \\ \sum_{i=0}^{5} W_{x_i} B_i(\mu(t_w)) \\ \sum_{i=0}^{5} W_{x_i} B_i(\mu(t_f)) \\ \sum_{i=0}^{5} W_{x_i} \dot{B}_i(\mu(t_o)) \\ \sum_{i=0}^{5} W_{x_i} \dot{B}_i(\mu(t_w)) \\ \sum_{i=0}^{5} W_{x_i} \dot{B}_i(\mu(t_f)) \end{bmatrix} \tag{3.6}$$

These are 6 equations and 6 variables, giving the values of W_{x_i} for $i \in [0 \dots 5]$.

We model the variable $k = \tan(\theta)$

$$\tan(\theta(t)) = k(t) \approx B_k(\mu(t)) = \sum_{i=0}^{5} W_{k_i} B_i(\mu(t)) \tag{3.7}$$

Combining equations for k and $\dot{x}$ models, we get

$$
\begin{aligned}
y(t) &= y_o + \int_{t_o}^{t} \left(\sum_{i=0}^{5} W_{x_i} \dot{B}_i(\mu(t)) \right) \left(\sum_{i=0}^{5} W_{k_i} B_i(\mu(t)) \right) dt \\
&= y_o + \int_{t_o}^{t} \sum_{i=0}^{5} W_{k_i} \; f_i(t, t_0, t_f, W_{x_0}, \ldots, W_{x_5}) \; dt \\
&= y_o + \sum_{i=0}^{5} W_{k_i} F_i(t, t_0, t_f, W_{x_0}, \ldots, W_{x_5}) \qquad (3.8) \\
f_i(t, t_0, t_f, W_{x_0}, \ldots, W_{x_5}) &= B_i(\mu(t)) \sum_{i=0}^{5} W_{x_i} \dot{B}_i(\mu(t))
\end{aligned}
$$

Note that the functions f_i and F_i are polynomials in t (as we have calculated W_{x_i} beforehand), and can be fully estimated beforehand. Solve constraints like in equation 3.6 for conditions on y and $k = \tan(\theta)$. The solution will depend on the number and type of constraints.

3.2.4 Bernstein Polynomials for Multi-robot UAVs

For a multi-rotor robot, additional variables are needed. The variables will now be x, y, z for the UAV's position in 3D Euclidean space, ϕ, θ, ψ for the roll, pitch, and yaw orientation in 3D space. The rotation matrix relating the global and vehicle inertial frame is given by

$$
\mathbf{R} = \begin{bmatrix} c\theta\, c\psi & c\theta\, s\psi & -s\theta \\ s\phi\, s\theta\, c\psi - c\phi\, s\psi & s\phi\, s\theta\, s\psi + c\phi\, c\psi & s\phi c\theta \\ c\phi\, s\theta\, c\psi & c\phi\, s\theta\, s\psi - s\phi\, c\psi & c\phi\, c\theta \end{bmatrix}
$$

The vehicle's position, linear and angular velocity are represented by the following vectors

$$
\xi = [x \;\; y \;\; z]^\top \qquad V = [u \;\; v \;\; w]^\top \qquad \omega = [p \;\; q \;\; r]^\top
$$

The equations of motion are given by the following

$$
\begin{aligned}
\dot{\xi} &= \mathbf{R}^{-1} V \\
F &= m\dot{V} + \omega \times mV \\
\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} &= \mathbf{E}^{-1} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \\
\mathbf{R} \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} &= \begin{bmatrix} F_{XB} \\ F_{YB} \\ F_{ZB} \end{bmatrix} = m \begin{bmatrix} \dot{u} + qw - rv \\ \dot{v} + ru - pw \\ \dot{w} + pv - qu \end{bmatrix}
\end{aligned}
$$

These equations were taken from [VCR15].

3.2.5 Accommodating More Waypoints

We could accommodate more waypoints through the following methods

1. Increase the value of n (n-th degree Bernstein basis). This will allow us to introduce more waypoints while keeping the system *critically constrained*.
2. We could directly add the waypoints, with the same n. This will make the system *over-constrained*.

Piece-wise Bernstein Polynomials However, we usually have a global path planner (like RRT) running at lower frequencies (say 10 Hz). That path planner gives many points (consider them to be waypoints, with the exception of start and goal).

At a different frequency, and within a small neighborhood, we need to travel from one point to another (on this proposed solution by the global path planner) through waypoints. This is done through a piece-wise fitting of Bernstein Polynomials while keeping the end constraints in check (for position and velocity).

Through this method, we can theoretically accommodate all the points proposed by the global path planner into a big piece-wise Bernstein polynomial.

3.2.6 Collision Avoidance

If there is a static obstacle, we can do one of the following things

- **Break the Bernstein path**: We can Break the Bernstein/Bezier path into smaller Bernstein basis polynomials and in the segment containing the obstacle, place a waypoint sufficiently far away from the obstacle. We must keep the end trajectory conditions in check, so that we do not give different velocities for the ends.
- We can optimize for a waypoint using an optimization formulation (similar to the multiple robot formulation in the next subsection).

Multiple Non-holonomic Robots We can selectively optimize coefficients for bernstein polynomials of different robots so that they avoid collisions. The main concepts are derived from [KŠ10] (a summary can be found in [SK07]). Consider a Bernstein polynomial with $b = 4$ degree. The polynomial will be given by

$$\mathbf{r}(\lambda) = \sum_{i=0}^{b} B_{i,b}(\lambda)\, \mathbf{p}_i \qquad \text{where } B_{i,b}(\lambda) = {}^{b}\mathrm{C}_i\, \lambda^i\, (1-\lambda)^{b-i}\ ,\ i = 0, 1, \ldots, b \tag{3.9}$$

Where $\mathbf{r} = [x\ \ y]^\top$ is the path. The points $\mathbf{p}_i = [x_i\ \ y_i]^\top$ are the control points. The normalized time is given by $\lambda(t) = t/T_{\max}$. Considering the first derivative of path (the velocity), we get

$$\mathbf{v}(\lambda) = \frac{d\mathbf{r}(\lambda)}{d\lambda} = b \sum_{i=0}^{b-1} (\mathbf{p}_{i+1} - \mathbf{p}_i)\, B_{i,b-1}(\lambda) \tag{3.10}$$

Here $\mathbf{v}(\lambda) = [v_x,\ v_y]^\top$. We also know that $B_{b-1,i}(0) = 0\,, i = 1, \ldots, b-1$ while $B_{b-1,0}(0) = 1$. We also know that $B_{b-1,i}(1) = 0\,, i = 0, \ldots, b-2$ which $B_{b-1,b-1}(1) = 1$. This means that $\mathbf{v}(0) = 4(\mathbf{p}_1 - \mathbf{p}_0)$ and $\mathbf{v}(1) = 4(\mathbf{p}_4 - \mathbf{p}_3)$. These give us the values for $\mathbf{p}_1$ and $\mathbf{p}_3$ readily as follows

$$\mathbf{p}_1 = \mathbf{p}_0 + \frac{1}{4}\mathbf{v}(0) \qquad\qquad \mathbf{p}_3 = \mathbf{p}_4 - \frac{1}{4}\mathbf{v}(1) \tag{3.11}$$

We already know that $\mathbf{p}_0$ is the starting point and $\mathbf{p}_4$ is the ending point. This means that we have $\mathbf{p}_2$ to optimize for each robot, to avoid a collision.

Multiple robots Let us consider a case with multiple robots, each with a trajectory given by $\mathbf{r}_i(\lambda_i) = [x_i(\lambda_i), y_i(\lambda_i)]^\top$. Note that each robot can have its own normalized time reference given by $\lambda_i = t/T_{\max_i}$. The distance between two robots i and j is given by $r_{ij}(t) = |\mathbf{r}_i(t) - \mathbf{r}_j(t)|$. We must keep this above a certain threshold d_s (minimum distance between two robots). We would also like to minimize the length of the paths of each robot. The length of path of robot i is given by

$$s_i = \int_0^1 \left(v_{xi}^2(\lambda_i) + v_{yi}^2(\lambda_i)\right)^{1/2} d\lambda_i \tag{3.12}$$

Note that the above equation is a variable *only* in $\mathbf{p}_{2i}$ (which is $\mathbf{p}_2$ for robot i). We can theoretically minimize this and get $\mathbf{p}_{2i}$ from this itself (keeping collision avoidance and other things aside). But this doesn't guarantee anything other than the shortest path length.

Optimization problem We will now consider minimizing the collective path lengths, while adhering to the constraints. This gives the following optimization objective

$$\min \sum_{i=1}^{n} s_i$$
$$\text{subject to } \; d_s - r_{ij}(t) \leq 0 \; , \;\; v_i(t) - v_{\max_i} \; , \; a_i(t) - a_{\max_i} \leq 0 \; \forall \, i, j, i \neq j \; , \; 0 \leq t \leq \max_i(T_{\max_i})$$

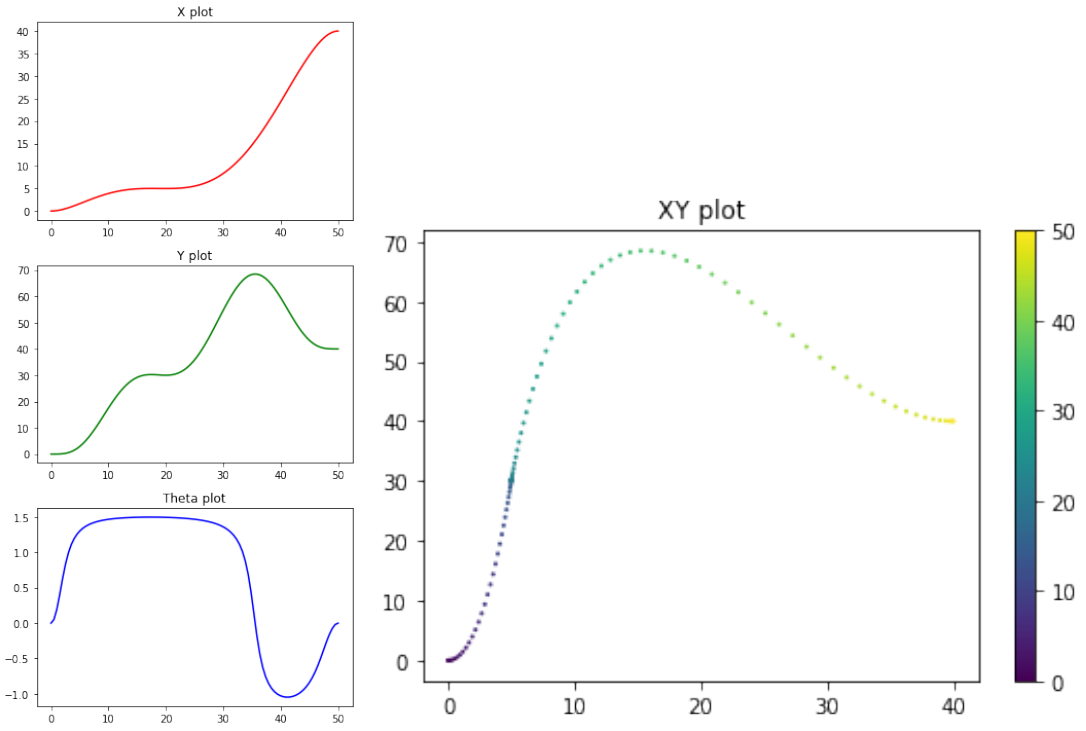
This can be converted to the following optimization problem

$$\begin{aligned} F = & \sum_i s_i + c_1 \sum_{ij} \max_{ij}(0, \; {}^1\!/_{r_{ij}(t)} - {}^1\!/_{d_s}) + c_2 \sum_i \max_i(0, \; v_i(t) - v_{\max_i}) \\ & + c_3 \sum_i \max_i(0, \; a_i(t) - a_{\min_i}) \\ & \min \qquad F \\ & \text{subjected to} \quad \mathbf{P}_2, \; \mathbf{T}_{\max} \\ & i, j, i \neq j, \;\; 0 \leq t \leq \max_i(T_{\max_i}) \end{aligned} \tag{3.13}$$

The values c_1, c_2, and c_3 are scalar constants in the above equation. The c_1 term ensures no collision, the c_2 term ensures maximum possible velocity, and the c_3 term ensures maximum acceleration. Due to the latter two, the penalty function F is also subjected to the time for each robot. The above equations can probably be solved through an optimizer like scipy.optimize.minimize.

3.2.7 Results

The videos can be found here.

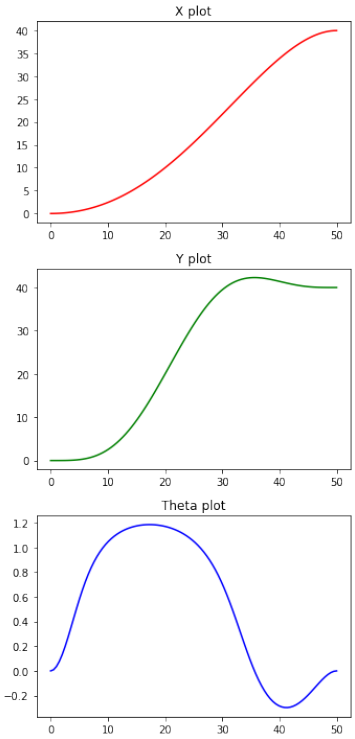


(a) Plots for x, y, and θ

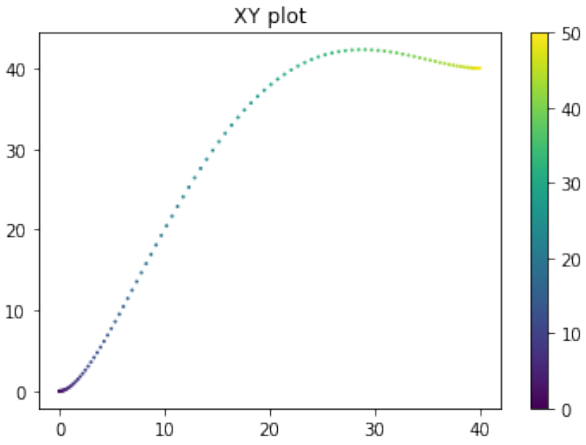
(b) Plot for x, y path

Figure 4: Case 1 trajectories

File name is `out_1.avi`. As it can be seen, the constraints for the 'dx' at waypoint is being adhered (the vehicle comes to a stop there). It'll be more beneficial to not stop at waypoints.

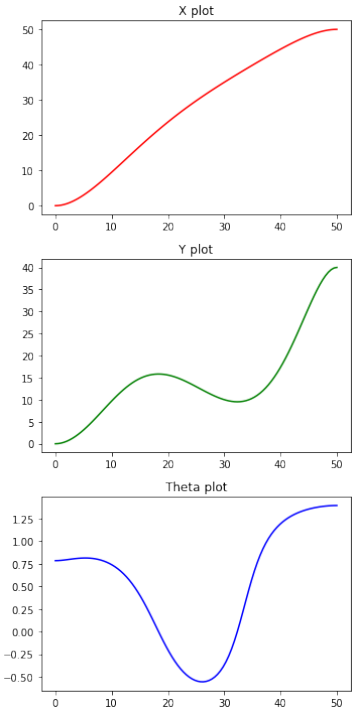


(a) Plots for x, y, and θ

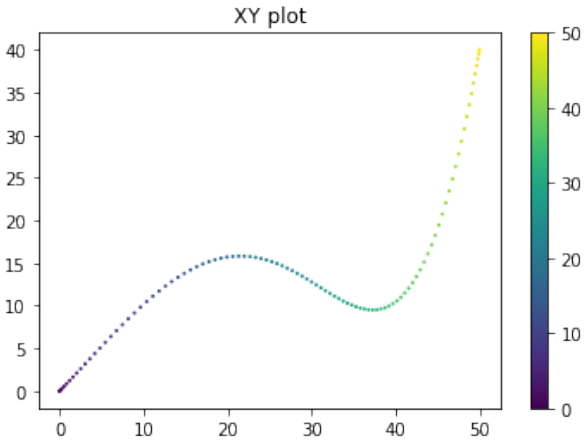


(b) Plot for x, y path

Figure 5: Case 2 trajectories (different waypoint)

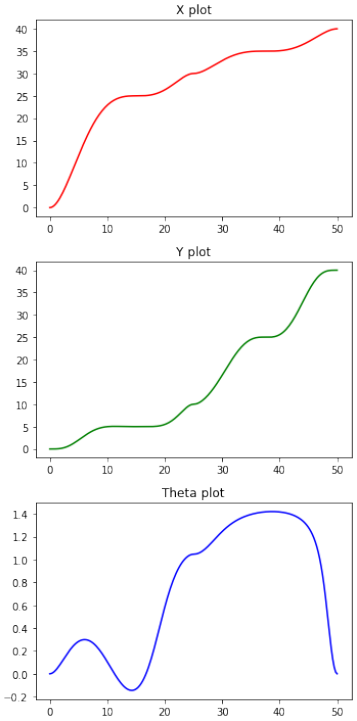


(a) Plots for x, y, and θ

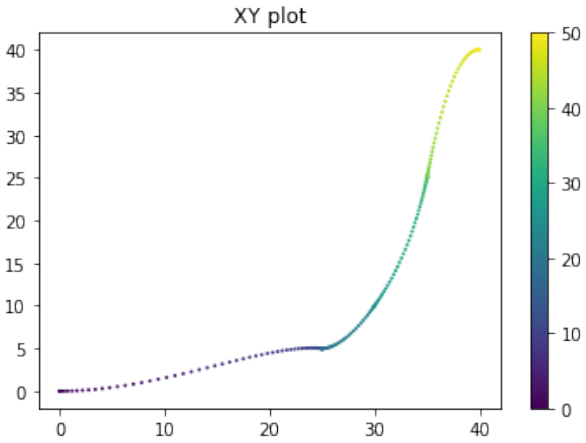


(b) Plot for x, y path

Figure 6: Case 3 trajectories
Avoiding obstacles by moving the waypoints of bernstein polynomials.



(a) Plots for x, y, and θ



(b) Plot for x, y path

Figure 7: Case 4 trajectories
Avoiding obstacles by joining two bernstein polynomials.

Advantages of Bernstein Polynomials Bernstein polynomials have the following advantages over other methods

1. It has smooth blending through the waypoints. This means that not only the function is continuous, and differentiable; but also its derivatives are continuous and differentiable.
2. It can be quickly calculated, given a set of waypoints. If we do not want to solve any equations, we directly set the waypoints as control points.
3. Applying a Euclidean transformation to the entire path is equivalent to applying the transformation to the control points alone. The weighed sum only depends on time (not space).

3.3 Assignment 3: Time scaling

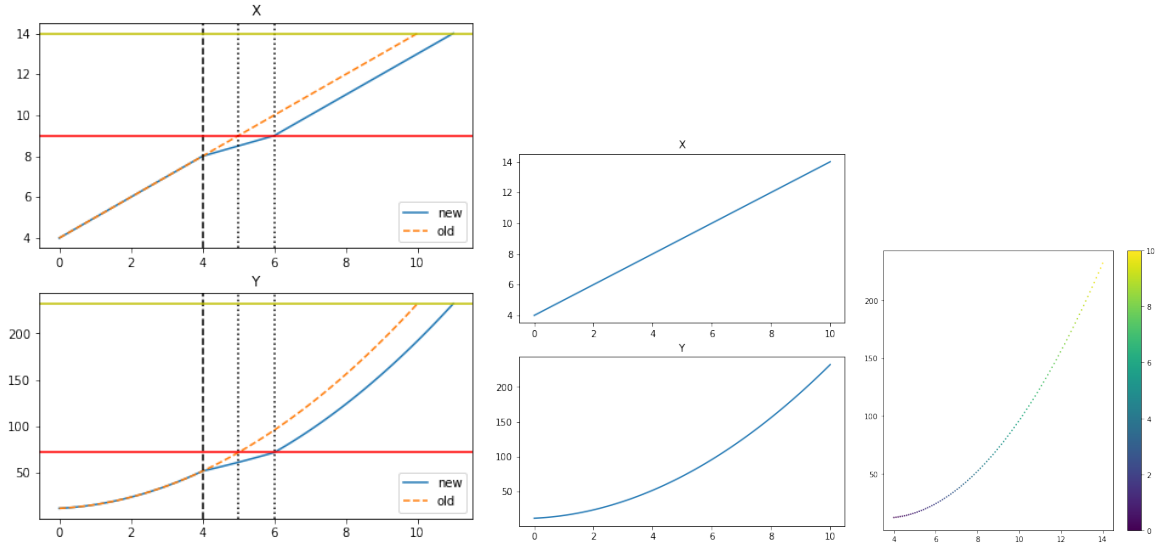
A detailed report is present at TheProjectsGuy/IS_RPN22-EC9.404

3.3.1 Constant Time Scaling

Constant time scaling is implemented using the equation below

$$\dot{x}(\tau) = k\,\dot{x}(t) \tag{3.14}$$

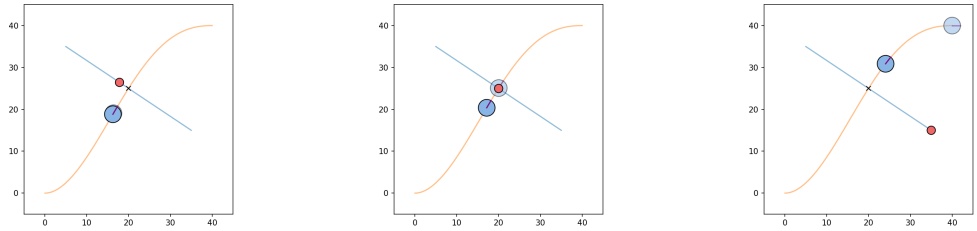
The results for this are shown in figure 9.



(a) Time scaling (b) Original (c) XY

Figure 8: Time scaling

The variables X and Y are functions of time. The time scaling is applied from 4 to 5 seconds, with $k = 0.5$ (the new end of time scaling will therefore happen at 6 seconds).
In left figure, it is apparent that the velocities have decreased to half in the time scaling period, while the duration has doubled.
The center and right figures show the original trajectory (X and Y as function of time in center, and X vs. Y plot in the right).



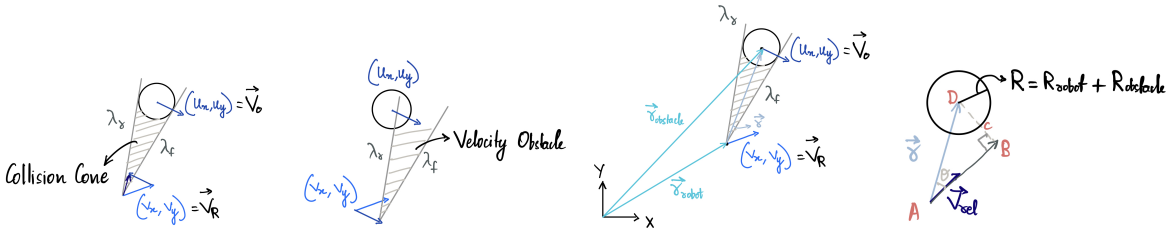
(a) Start (b) Collision avoidance (c) End

Figure 9: Rule-based constant time-scaling

3.3.2 Collision cone based constant time-scaling

A collision cone is comprised of the relative velocities of the robot (with reference to an obstacle) that will lead to a collision in the future. This is demonstrated in the figure 10.

We can formulate the following to avoid a collision



(a) Collision Cone (b) Velocity Obstacle (c) Environment (d) Avoiding collision

Figure 10: Collision Cone

The velocity of obstacle is $\overrightarrow{V}_O = (u_x, u_y)$. The velocity of obstacle is $\overrightarrow{V}_R = (v_x, v_y)$. THe radius of robot is R_{robot} and the radius of obstacle is $R_{obstacle}$. The obstacle is diluted to $R = R_{robot} + R_{obstacle}$ while the robot is reduced to a point. After time scaling, the velocity of the robot becomes scaled by a factor of s, that is, the time scaled robot velocity is $\vec{V}_R = (s\dot{x}_1, s\dot{y}_1)$.

$$\text{DB} \geq \text{DC} \Rightarrow \text{DB}^2 \geq R^2 \Rightarrow \text{AD}^2 - \text{AB}^2 \geq R^2 \qquad \vec{r}_{obstacle} = \vec{r}_{robot} + \vec{r} \Rightarrow \vec{r} = \vec{r}_{obstacle} - \vec{r}_{robot}$$

$$\text{AB} = \|\vec{r}\| \cos(\theta) = \frac{\vec{V}_{rel} \cdot \vec{r}}{\left\|\vec{V}_{rel}\right\|} \Rightarrow \text{AB}^2 = \left[\frac{\vec{V}_{rel} \cdot \vec{r}}{\left\|\vec{V}_{rel}\right\|}\right]^2 \qquad \text{AD} = \|\vec{r}\| \Rightarrow \text{AD}^2 = \|\vec{r}\|^2$$

$$\vec{V}_{rel} = \vec{V}_R - \vec{V}_O = (s\dot{x}_1 - \dot{x}_2, s\dot{y}_1 - \dot{y}_2) \qquad \vec{r} = \vec{r}_{obstacle} - \vec{r}_{robot} = (x_2 - x_1, y_2 - y_1)$$

$$\text{AD}^2 - \text{AB}^2 \geq R^2 \Rightarrow \|\vec{r}\|^2 - \left[\frac{\vec{V}_{rel} \cdot \vec{r}}{\left\|\vec{V}_{rel}\right\|}\right]^2 \geq R^2 \qquad \|\vec{r}\|^2 - \left[\frac{\vec{V}_{rel} \cdot \vec{r}}{\left\|\vec{V}_{rel}\right\|}\right]^2 - R^2 \geq 0$$

$$\vec{V}_{rel} \cdot \vec{r} = (s\dot{x}_1 - \dot{x}_2)(x_2 - x_1) + (s\dot{y}_1 - \dot{y}_2)(y_2 - y_1) \qquad \left\|\vec{V}_{rel}\right\| = \sqrt{(s\dot{x}_1 - \dot{x}_2)^2 + (s\dot{y}_1 - \dot{y}_2)^2}$$

We get the following final inequality for the scaling factor s.

$$(x_1 - x_2)^2 + (y_1 - y_2)^2 - R^2 - \frac{((s\dot{x}_1 - \dot{x}_2)(x_2 - x_1) + (s\dot{y}_1 - \dot{y}_2)(y_2 - y_1))^2}{(s\dot{x}_1 - \dot{x}_2)^2 + (s\dot{y}_1 - \dot{y}_2)^2} \geq 0 \tag{3.15}$$

We represent this equation in the form $as^2 + bs + c \geq 0$. This gives us the following solution space (set) for s

$$S_{sol} = \begin{cases} [s_{min}, \infty) \cap ((-\infty, \gamma_1] \cup [\gamma_2, \infty)) & a > 0,\ d > 0 \\ [s_{min}, \infty) & a > 0,\ d < 0 \\ [s_{min}, \infty) \cap [\gamma_1, \gamma_2] & a < 0,\ d > 0 \\ \phi & a < 0,\ d < 0 \end{cases} \tag{3.16}$$

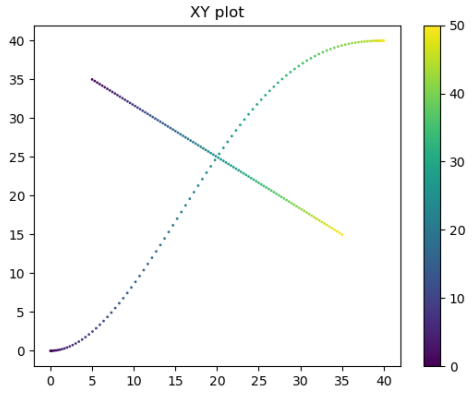
Where

$$d = b^2 - 4ac \qquad \gamma_1 = \frac{-b - \sqrt{d}}{2a} \qquad \gamma_2 = \frac{-b + \sqrt{d}}{2a}$$

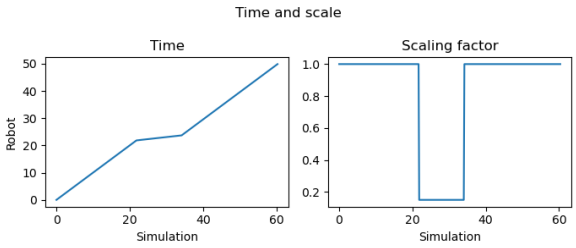
We then apply acceleration constraints and pick $s = \min\{S_{sol}\}$ (minimum of the solution space). A simple result is shown in figure 11 (simulation video stages in figure 12).

3.3.3 Rule-based Linear time scaling

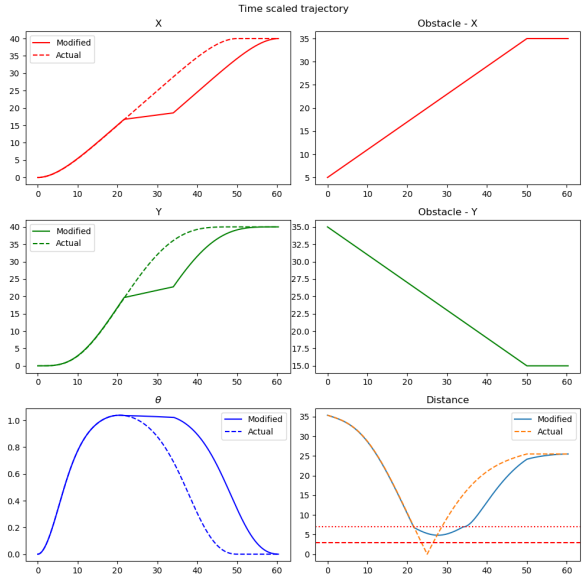
We use the scaling factor as a function of the simulation time, given by $s(t) = a + bt$. The scaling is done as in constant time scaling case. The results are shown in figure 13. The snippets from simulation are shown in figure 14.



(a) XY plots

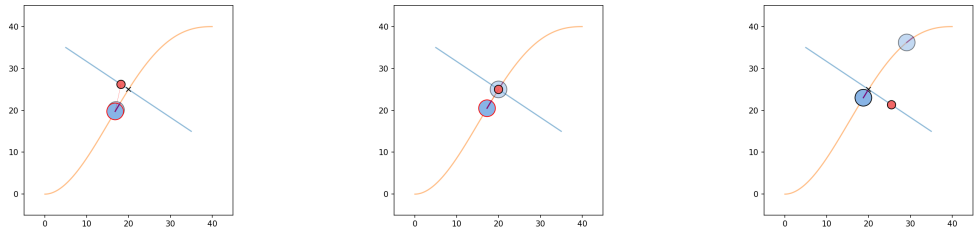


(b) Time scaling



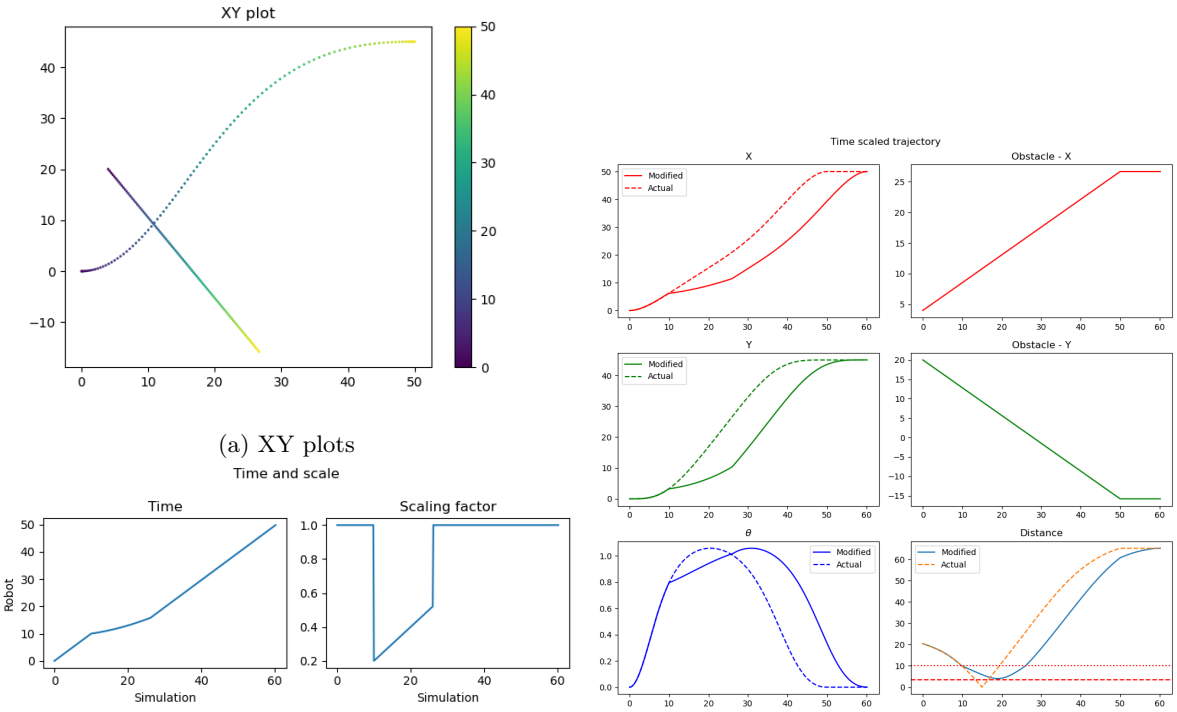
(c) Time scaled results

Figure 11: Collision cone based constant time scaling
The distance plot is shown in the bottom right of c. As seen, the time scaling gets activated at the thin red horizontal line (detection distance) and a collision is avoided (the distance plot does not cross the thick horizontal red line). See the slope and the scaling factor change in b. The original (colliding) trajectories are shown in a.



(a) Start (b) Collision avoidance (c) End

Figure 12: Collision cone - Constant time scaling - Video snaps

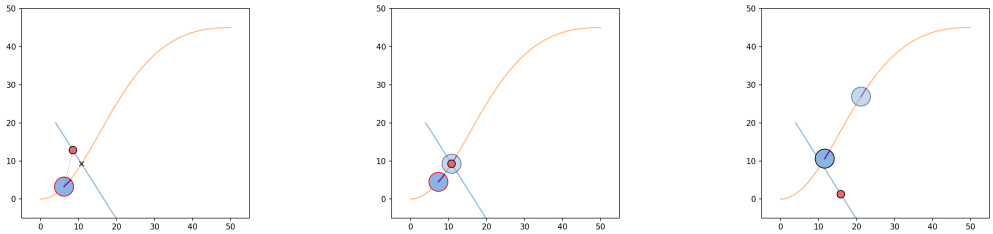


(a) XY plots

(b) Time scaling

(c) Time scaled results

Figure 13: Rule-based Linear time scaling
The distance plot is shown in the bottom right of c. As seen, the time scaling gets activated at the thin red horizontal line (detection distance) and a collision is avoided (the distance plot does not cross the thick horizontal red line). See the slope and the scaling factor change in b (notice the linear slope, with time as parabolic). The original (colliding) trajectories are shown in a.



(a) Start

(b) Collision avoidance

(c) End

Figure 14: Rule-based - Linear time scaling - Video snaps
The simulation is available as `rb_lts_exp1.avi` in the shared OneDrive folder.

3.3.4 Smooth trajectory

As we have seen, using iterative collision cone (in case of constant) and linear time scaling techniques did not completely fix the discontinuity problem. The following could be tried

- Incorporate the acceleration and velocity constraints on the actuators. This way, even if the controller gives a time scaled velocity, the actuator will clip it towards the limits.

  However, this could be dangerous and could lead to collisions if the frequency of control loop isn't high enough or the constraints are too restrictive.

- Trajectories can be smoothened by incorporating information of higher derivatives in the time scaling problem.

- The entire problem could be converted into a constrained optimization problem, enforcing dynamic constraints.

- Some proposals include interleaving time-scaling with MPC [The+19], or using non-linear time scaling [SK13a] can also be used.

3.3.5 MPC Parallel

Trajectories given by simple time-scaling (like what's implemented here) may not be very smooth, whereas trajectories from the MPC will be smooth (because it's from an optimizer using many more constraints on motion).

For the MPC to avoid collisions *specifically* using time-scaling, you could add the time-scaling equations (like equation 3.15) as additional solver constraints and incorporate the scaling factor (or scaled velocities) in the system state (unknown variables). Otherwise, the MPC will deviate from the planned trajectory (to avoid the obstacle) whereas time-scaling will stay on the trajectory.

3.3.6 Multi-robot

Time scaling can be extended to multiple robots using an intersection space of multiple inequalities (as presented in [SK13b]). The solution space can be decomposed into multiple conditions (as done in 3.16, but using different a_i - one for each robot). Linear programming approaches can also solve such problems.

Using velocity obstacle [FS98] is also a feasible solution (single robot and multiple obstacle case). The problem will more or less remain the same.

For any two robot case, the velocity vectors have to have sufficient deviation. If they're parallel or antiparallel, then time-scaling will not give a viable solution (it'll lead to collision or both remaining stationary). In such cases, path will have to be altered.

References

[FS98] Paolo Fiorini and Zvi Shiller. "Motion Planning in Dynamic Environments Using Velocity Obstacles". In: *The International Journal of Robotics Research* 17.7 (1998), pp. 760–772. DOI: 10.1177/027836499801700706. eprint: https://doi.org/10.1177/027836499801700706. URL: https://doi.org/10.1177/027836499801700706.

[SK07] Igor Skrjanc and Gregor Klancar. "Cooperative collision avoidance between multiple robots based on bezier curves". In: *2007 29th International Conference on Information Technology Interfaces*. IEEE. 2007, pp. 451–456.

[KŠ10] Gregor Klančar and Igor Škrjanc. "A case study of the collision-avoidance problem based on Bernstein–Bézier path tracking for multiple robots with known constraints". In: *Journal of Intelligent & Robotic Systems* 60.2 (2010), pp. 317–337.

[SK13a] Arun Kumar Singh and K. Madhava Krishna. "Reactive collision avoidance for multiple robots by non linear time scaling". In: *52nd IEEE Conference on Decision and Control*. 2013, pp. 952–958. DOI: 10.1109/CDC.2013.6760005.

[SK13b] Arun Kumar Singh and K. Madhava Krishna. "Reactive collision avoidance for multiple robots by non linear time scaling". In: *52nd IEEE Conference on Decision and Control*. 2013, pp. 952–958. DOI: 10.1109/CDC.2013.6760005.

[VCR15] Pedro Vílez, Novel Certad, and Elvis Ruiz. "Trajectory generation and tracking using the AR. Drone 2.0 quadcopter UAV". In: *2015 12th Latin American Robotics Symposium and 2015 3rd Brazilian Symposium on Robotics (LARS-SBR)*. IEEE. 2015, pp. 73–78.

[The+19] Raghu Ram Theerthala et al. "Motion Planning Framework for Autonomous Vehicles: A Time Scaled Collision Cone Interleaved Model Predictive Control Approach". In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. 2019, pp. 1075–1080. DOI: 10.1109/IVS.2019.8813823.