Assignment 2A
Fitting Polynomials to Non-Holonomic Trajectories
EC4.403 - Robotics: Planning and Navigation

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

April 1, 2022

Contents

List of Figures

*M.S. by research - CSE, IIIT Hyderabad, Roll No: 2021701032

1 Theory: Non-holonomic Trajectory Planning using Bernstein Basis

This section only contains the theoretical background and procedure explanation for the assignment. It does not answer any specific question.

Proceed to section 2 for beginning of assignment submission.

1.1 Bernstein Polynomials

We generally use Bernstein Polynomials to approximate functions. They can also be used to generate trajectories through waypoints under certain constraints (as we will see here). This subsection explores only the polynomials.

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

$$b_{v,n}(x) = \binom{n}{v} x^v \, (1-x)^{n-v}, \quad v = 0, \ldots, n \tag{1.1}$$

1.2 Approximating functions

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

$$B_n(f)\,(x) = \sum_{v=0}^{n} f\left(\frac{v}{n}\right) b_{v,n}(x) \tag{1.2}$$

The above approximation is more accurate if $n \to \infty$, that is $\lim_{n\to\infty} B_n(f) = f$.

Such a combination can be run through the waypoints (which will be presented as $f(v/n)$ to the polynomial) to approximate the function to which those waypoints belong. An application of this method is to generate smooth splines through a given set of control points (called *knots*) which is very useful in computer vision [2].

The basis function values for $n = 5$ (fifth-degree) is shown in figure

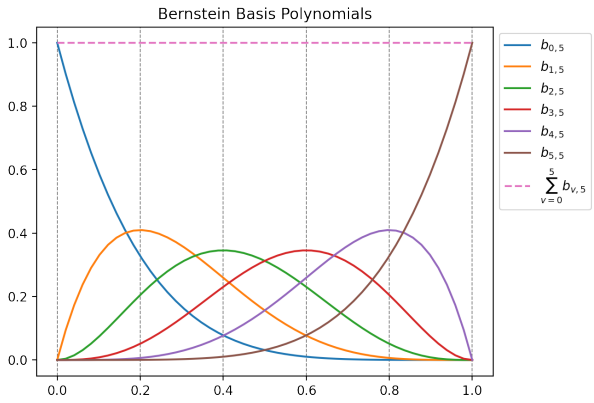


Figure 1: Bernstein Basis Values for $n = 5$

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 $f\,(v/n)$ 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.

1.3 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

$$\dot{x} = v\cos(\theta) \quad \dot{y} = v\sin(\theta) \qquad \Rightarrow \dot{y} = \dot{x}\tan(\theta) \quad \Rightarrow y = \int \dot{x}\tan(\theta)\, dt \tag{1.3}$$

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.

1.3.1 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 1.3.

1.3.2 Modeling x and $\tan(\theta)$

We model x and $\tan(\theta)$ as degree $n = 5$ Bernstein polynomials. Specifically, we do the following for x

$$x(t) \approx B_n(x)(t) = B_n(x)(\mu(t)) = B_x(\mu(t)) = \sum_{i=0}^{5} W_{x_i} B_i(\mu(t)) \tag{1.4}$$

Note that the x in the equations 1.1 and 1.2 mean an *input* to some function (in the context of those equations). The x in the above equation 1.4 is *itself a function* (not an input, the input here is time t).

The function $\mu(t) = \frac{t - t_o}{t_f - t_o} = \mu_t$ normalizes t to be in the range $[0, 1]$. The Bernstein basis term therefore becomes $B_i(\mu(t)) = b_{i,5}(\mu_t)$ (see equation 1.1 for expansion).

Also note that W_{x_i} is a proxy for x which decides the weight for a particular Bernstein basis B_i (see figure 1 for the plots).

The following can be done similarly for $\tan(\theta)$

$$\tan(\theta(t)) = k(t) \approx B_k(\mu(t)) = \sum_{i=0}^{5} W_{k_i} B_i(\mu(t)) \tag{1.5}$$

1.3.3 Finding W_{x_i}

We model the variable x using equation 1.4.

$$x(t) \approx B_n(x)(t) = \sum_{i=0}^{5} W_{x_i} B_i(\mu(t)) \qquad \text{(Eq. 1.4 revisited)}$$

We model $\dot{x}$ as

$$\dot{x}(t) = \sum_{i=0}^{5} W_{x_i} \dot{B}_i(\mu(t)) \tag{1.6}$$

We need to find the values for the W_{x_i} terms (which are constants - they're weights of the Bernstein terms). We're given the values for t_o (starting time), t_w (waypoint time), t_f (final time), $x(t_o) = x_o$, $x(t_w) = x_w$, $x(t_f) = x_f$ (x at start, waypoint, and end time), $\dot{x}(t_o) = \dot{x}_o$, $\dot{x}(t_w) = \dot{x}_w$, and $\dot{x}(t_f) = \dot{x}_f$ ($\dot{x}$ at start, waypoint, and end time).

Upon substituting the above constraints in equations 1.4 and 1.6, we get the following

$$\begin{bmatrix} x_o \\ x_w \\ x_f \\ \dot{x}_o \\ \dot{x}_w \\ \dot{x}_f \end{bmatrix} = \begin{bmatrix} \sum_{i=0}^{5} W_{x_i} B_i(\mu(t_o)) \\ \sum_{i=0}^{5} W_{x_i} B_i(\mu(t_w)) \\ \sum_{i=0}^{5} W_{x_i} B_i(\mu(t_f)) \\ \sum_{i=0}^{5} W_{x_i} \dot{B}_i(\mu(t_o)) \\ \sum_{i=0}^{5} W_{x_i} \dot{B}_i(\mu(t_w)) \\ \sum_{i=0}^{5} W_{x_i} \dot{B}_i(\mu(t_f)) \end{bmatrix} \tag{1.7}$$

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

1.3.4 Finding W_{k_i}

We model the variable $k = \tan(\theta)$ using equation 1.5. Combining equation 1.6 and 1.5 into equation 1.3, we get

$$\begin{aligned} y(t) &= y_o + \int_{t_o}^{t} \left(\sum_{i=0}^{5} W_{x_i} \dot{B}_i(\mu(t)) \right) \left(\sum_{i=0}^{5} W_{k_i} B_i(\mu(t)) \right) dt \\ &= y_o + \int_{t_o}^{t} \sum_{i=0}^{5} W_{k_i} \; f_i(t, t_0, t_f, W_{x_0}, \dots, W_{x_5}) \; dt \\ &= y_o + \sum_{i=0}^{5} W_{k_i} F_i(t, t_0, t_f, W_{x_0}, \dots, W_{x_5}) \end{aligned} \tag{1.8}$$

$$f_i(t, t_0, t_f, W_{x_0}, \dots, W_{x_5}) = B_i(\mu(t)) \sum_{i=0}^{5} W_{x_i} \dot{B}_i(\mu(t))$$

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 1.7 for conditions on y and $k = \tan(\theta)$. The solution will depend on the number and type of constraints.

[1] Wikipedia: Bernstein polynomial

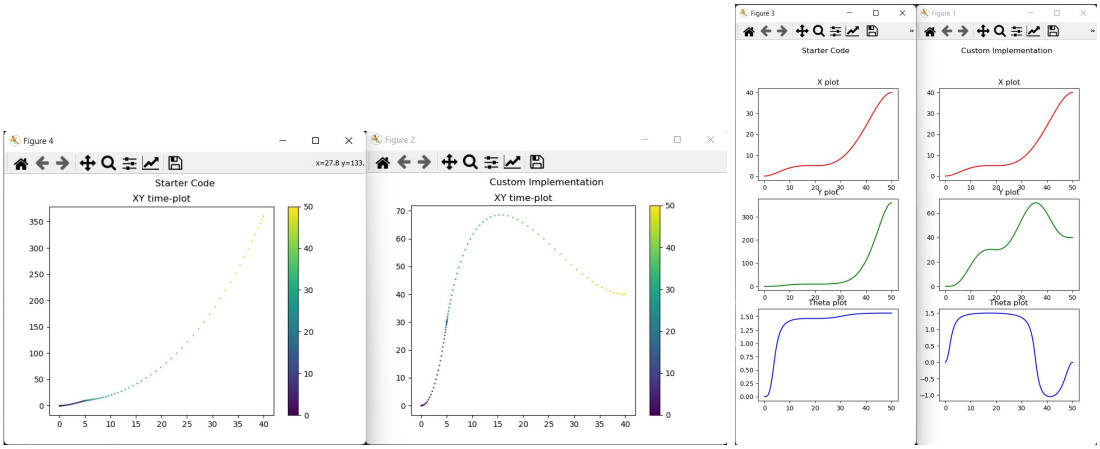
[2] Blog post: `https://www.particleincell.com/2012/bezier-splines/`

2 Note on Custom Implementation

The given `starter_code.ipynb` (mirrored in `src/starter_code.py`) was not used and a custom implementation was done in the `src` folder. This was done because

- The custom implementation uses sympy under-the-hood to handle integration and solving constraints. This makes using Wolfram Alpha not necessary.
- The custom implementation can scale better for different values of n. Only some changes in the number of constraints in the source code will be required (only two places), rest everything will scale accordingly. The starter code provided will have to be entirely re-written for different n values.

To see that they both give the same results, the script `comp_starter_custom.py` can be used. It shows the matching results and images (visualizing trajectories with the same constraints).



(a) XY plot (b) X, Y, θ plot

Figure 2: Comparison of custom and starter codes
For each of the image above, the left window is starter code and the right one is the custom implementation.

As we can see in figure 2, the starter code fails in some cases, specially when the angle $\theta \approx 90°$ (which is when $\tan(\theta) \to \infty$). The custom implementation, resolving the set of equations (and not being hard-coded), can recover (up to a limited extent).

Additional Notes for practicality of the submission

- The angle constraint at waypoint was dropped (both the k and $\dot{k}$ constraints are not applied). This was done to get a stable Bezier curve.
  Usually these things are deployed over small stretches and linked using terminal constraints. It is impractical to fit one through the entire path. This is *smooth patching* of multiple Bezier curves.

The main submission starts from the next section.

3 Bernstein Polynomials for Multi-rotor UAVs

The theory discussed in section 1 only applied for a differential drive robot (which has different, non-holonomic constraints).

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.

It is understood that the UAV cannot be at any orientation and fixed in space. Imagine the UAV pitching forward and not accelerating forward (not possible, the inclined thrust vector will accelerate it forward). The mathematical model for a UAV quadcopter is described below

3.1 Quadcopter Motion Model

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 Changes proposed

Based on the model above, the following changes will be needed

- The motion constraints mentioned in the end of the last section must be implemented (like in equation 1.3).
- Parameterized representation for UAV model will be needed.
- We can probably have a crude path planner (like RRT in configuration space) and merge bezier splines using a library like PyPI: bezier.

4 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*.

4.1 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.

5 Collision Avoidance

5.1 Static Obstacle

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

5.1.1 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.

5.1.2 Use an optimization algorithm

We can optimize for a waypoint using an optimization formulation (similar to the multiple robot formulation in the next subsection).

5.2 Multiple Non-holonomic Robots

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

$$\mathbf{r}(\lambda) = \sum_{i=0}^{b} B_{i,b}(\lambda)\, \mathbf{p}_i \qquad \text{where } B_{i,b}(\lambda) = {}^{b}\mathrm{C}_i\, \lambda^i\, (1-\lambda)^{b-i}\ ,\ i = 0, 1, \ldots, b \tag{5.1}$$

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

$$\mathbf{v}(\lambda) = \frac{d\mathbf{r}(\lambda)}{d\lambda} = b \sum_{i=0}^{b-1} (\mathbf{p}_{i+1} - \mathbf{p}_i)\, B_{i,b-1}(\lambda) \tag{5.2}$$

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

$$\mathbf{p}_1 = \mathbf{p}_0 + \frac{1}{4}\mathbf{v}(0) \qquad\qquad \mathbf{p}_3 = \mathbf{p}_4 - \frac{1}{4}\mathbf{v}(1) \tag{5.3}$$

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

$$s_i = \int_0^1 \left(v_{xi}^2(\lambda_i) + v_{yi}^2(\lambda_i)\right)^{1/2} d\lambda_i \tag{5.4}$$

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

$$\begin{aligned} F = & \sum_i s_i + c_1 \sum_{ij} \max_{ij}(0, \; {}^1\!/_{r_{ij}(t)} - {}^1\!/_{d_s}) + c_2 \sum_i \max_i(0, \; v_i(t) - v_{\max_i}) \\ & + c_3 \sum_i \max_i(0, \; a_i(t) - a_{\min_i}) \\ & \min \qquad F \\ & \text{subjected to} \quad \mathbf{P}_2, \; \mathbf{T}_{\max} \\ & i, j, i \neq j, \;\; 0 \leq t \leq \max_i(T_{\max_i}) \end{aligned} \tag{5.5}$$

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.

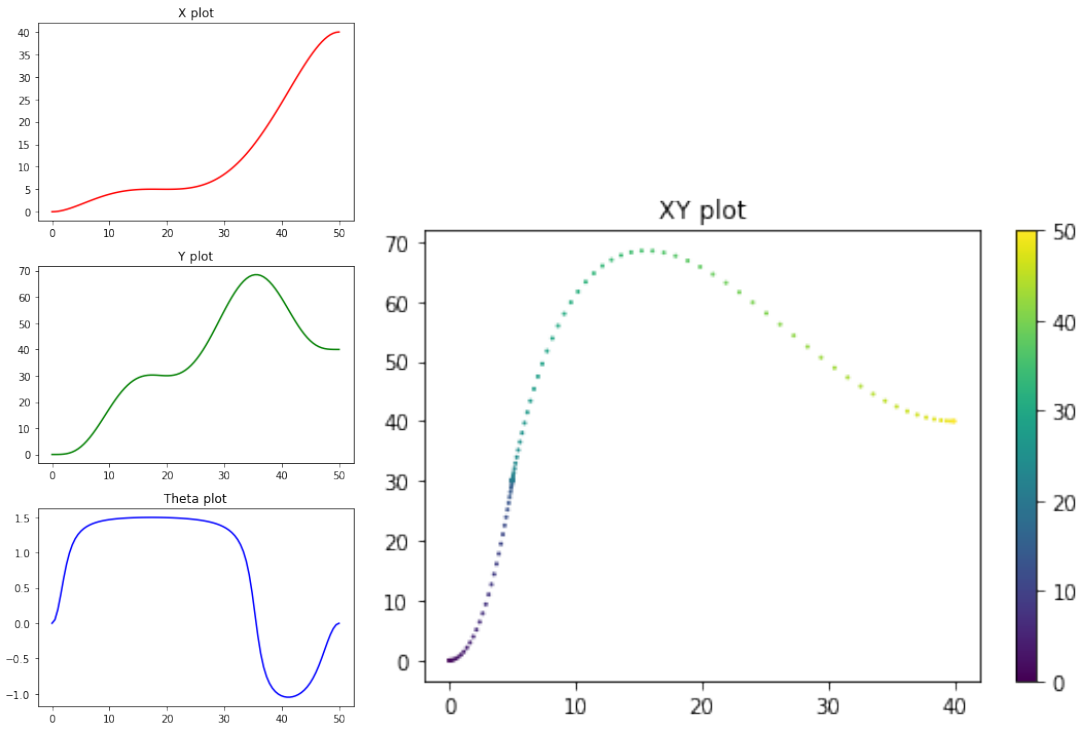
6 Videos Submitted

The videos can be found here (link: `https://iiitaphyd-my.sharepoint.com/:f:/g/personal/avnees h_mishra_research_iiit_ac_in/EogE243Ab4lDvm5VSueCLY8B7tLHOWnVw8i8PQnv5Bndfg?e=k8xJ1n`).

6.1 Case 1

File name is `out_1.avi` (trajectories in figure 3). The case uses `single_wpt_path.py` and has the following configurations.

```
# ==== Begin: User configuration area ====
# Points as [x, y]
start_pt = [0, 0]
end_pt = [40, 40]
way_pt = [5, 30]
# Time values
to, tw, tf = [0, 20, 50]    # Start, waypoint, end
# Other parameters
ko, kw, kf = [0, np.tan(np.pi/5), 0]    # k = np.tan(theta)
dko, dkw, dkf = [0, 0, 0]
dxo, dxw, dxf = [0, 0, 0]
# ==== End: User configuration area ====
```



(a) Plots for x, y, and θ

(b) Plot for x, y path

Figure 3: Case 1 trajectories

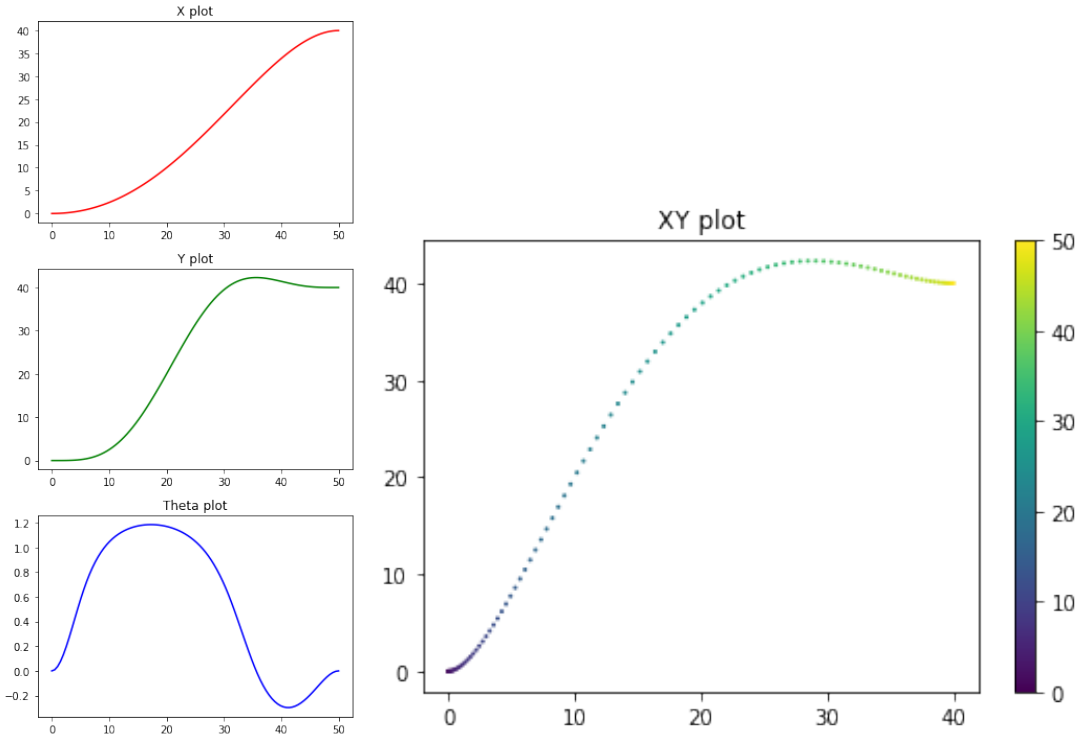
Observations As it can be seen, the constraints for the 'dx' at waypoint is being adhered (the vehicle comes to a stop there). It'll be more beneficial to not stop at waypoints.

6.2 Case 2

File name is `out_2.avi` (trajectories in figure 4). The case uses `single_wpt_path.py` and has the following configurations.

```
# ==== Begin: User configuration area ====
# Points as [x, y]
start_pt = [0, 0]
```

```
    end_pt = [40, 40]
    way_pt = [10, 20]
    # Time values
    to, tw, tf = [0, 20, 50]    # Start, waypoint, end
    # Other parameters
    ko, kw, kf = [0, np.tan(np.pi/5), 0]    # k = np.tan(theta)
    dko, dkw, dkf = [0, 0, 0]
    dxo, dxw, dxf = [0, 1, 0]
    # ==== End: User configuration area ====
```



(a) Plots for x, y, and θ

(b) Plot for x, y path

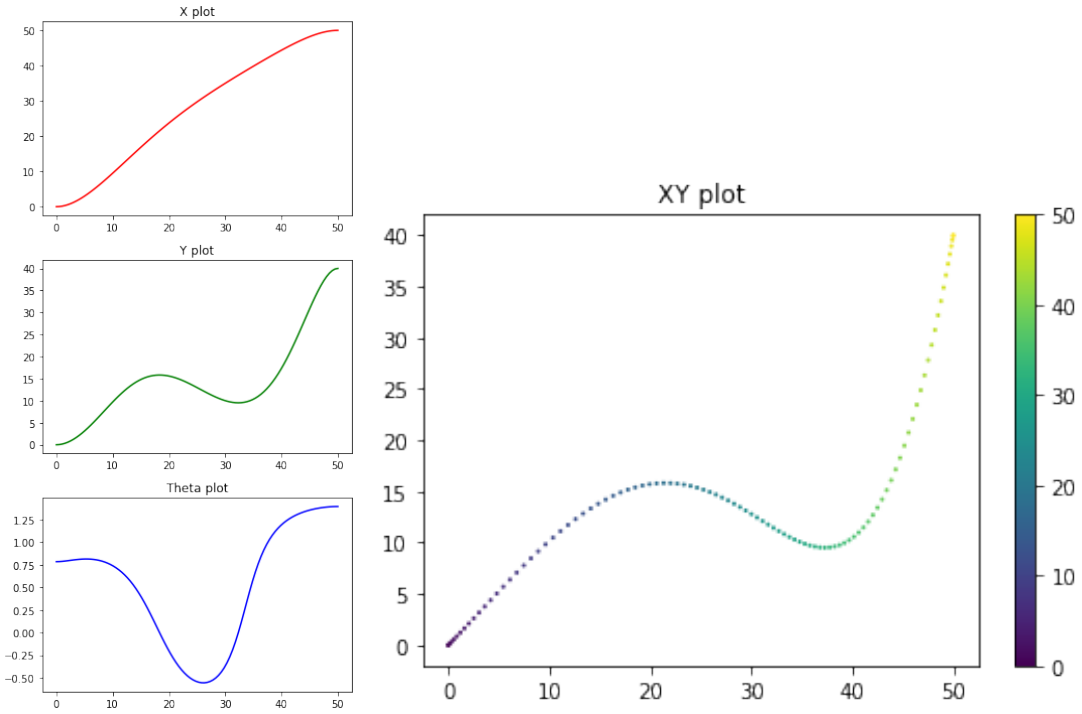
Figure 4: Case 2 trajectories

Observations Here, the vehicle doesn't stop at the waypoints and this makes the overall curve much better. Let's see observations when the θ is not zero at starting and ending.

6.3 Case 3

File name is `out_3.avi` (trajectories in figure 5). The case uses `single_wpt_path.py` and has the following configurations.

```
    # ==== Begin: User configuration area ====
    # Points as [x, y]
    start_pt = [0, 0]
    end_pt = [50, 40]
    way_pt = [35, 10]
    # Time values
    to, tw, tf = [0, 30, 50]    # Start, waypoint, end
    # Other parameters
    ko, kw, kf = [np.tan(np.deg2rad(45)),
        np.tan(np.pi/5),
        np.tan(np.deg2rad(80))]    # k = np.tan(theta)
    dko, dkw, dkf = [0, 0, 0]
    dxo, dxw, dxf = [0, 1, 0]
    # ==== End: User configuration area ====
```



(a) Plots for x, y, and θ

(b) Plot for x, y path

Figure 5: Case 3 trajectories

Observations Here, the vehicle starts and ends with an angle other than zero. This makes tha curve better conditioned and there is very little overshoot. It also has a very nice curve to reach the waypoint and be back to the goal.

7 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).

References

[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.

[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.