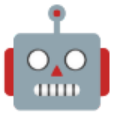


Robotics - Planning and Navigation - Assignment 2

Coding Task

The implementation of the Bernstein-based trajectory planning is in the code attached to this report.

Following is the link to the videos of simulations run as a part of this assignment:

https://drive.google.com/drive/folders/12DSjP_QLWCvoDicUDYmeZDkewGbVQUug?usp=sharing

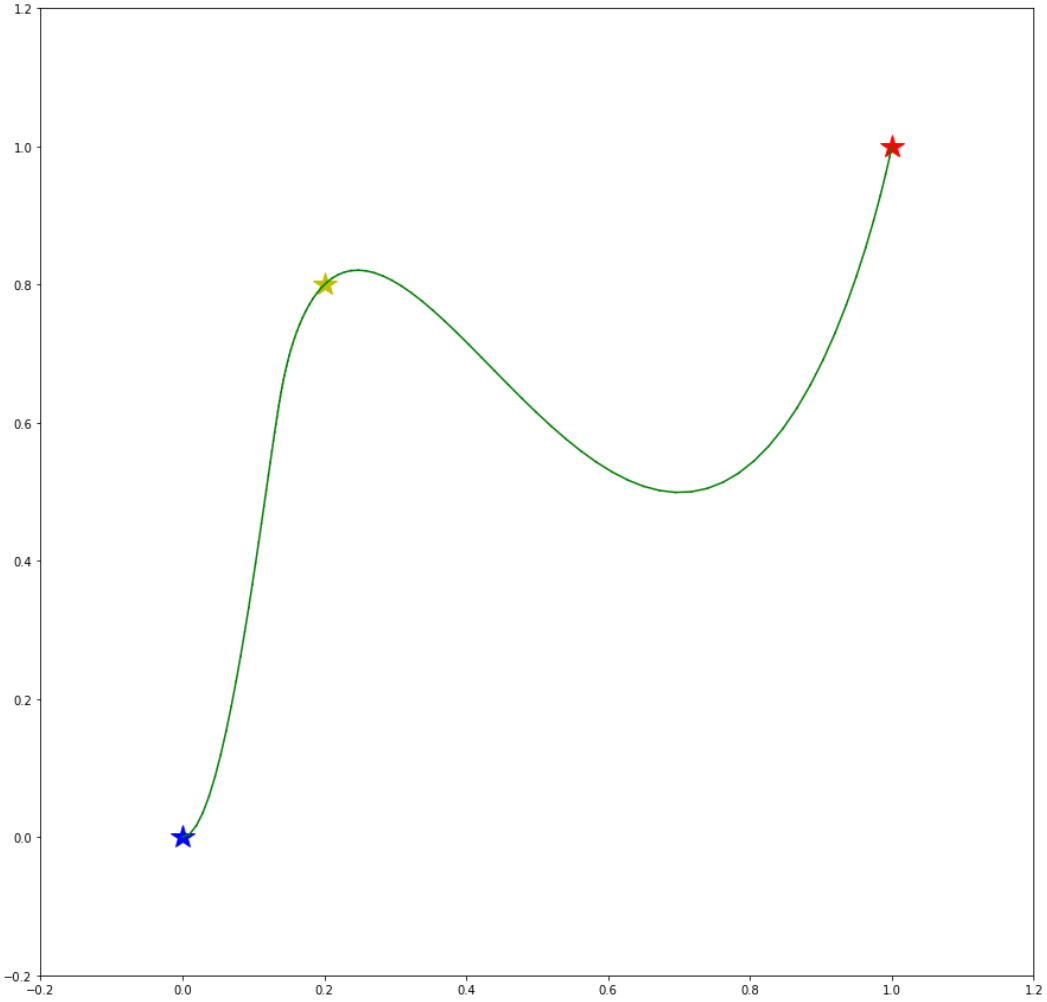
Run 1

Task Details

The robot starts at $x_0 = 0$, $y_0 = 0$ with velocity $\dot{x}_0 = 1.0$, $\dot{y}_0 = 0.0$ and moves towards the goal location $x_f = 1$, $y_f = 1$ with velocity $\dot{x}_0 = 0.2$, $\dot{y}_0 = 1.0$. We pass through the waypoint $x_f = 0.2$, $y_f = 0.8$ with velocity $\dot{x}_0 = 0.7$, $\dot{y}_0 = 0.7$.

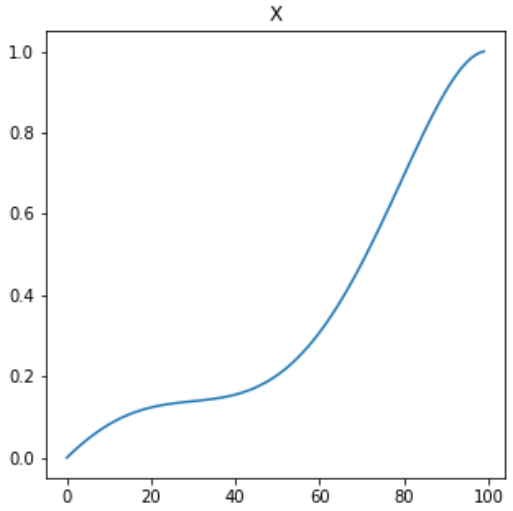
```
b = BernsteinPlanner(
    BernsteinPlanner.Constraint(x=0.0, y=0.0, dx=1.0, dy=0.0, t=0.0),
    BernsteinPlanner.Constraint(x=1.0, y=1.0, dx=0.2, dy=1.0, t=1.0),
    BernsteinPlanner.Constraint(x=0.2, y=0.8, dx=0.7, dy=0.7, t=0.5),
)
```

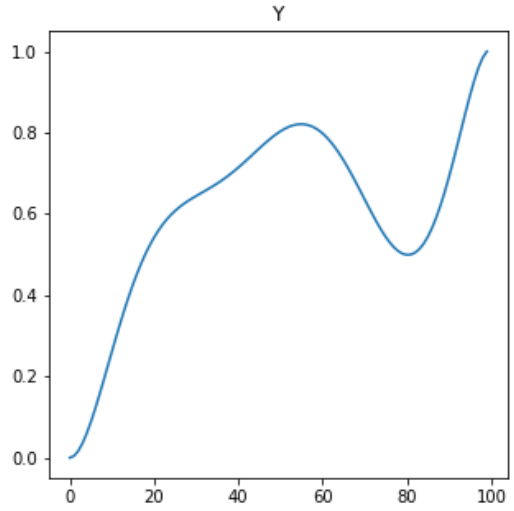
Trajectories Computed

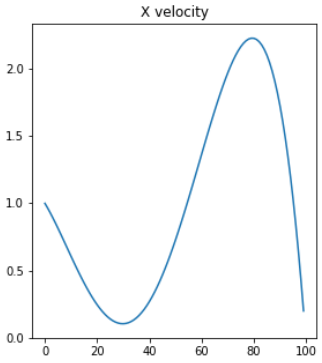


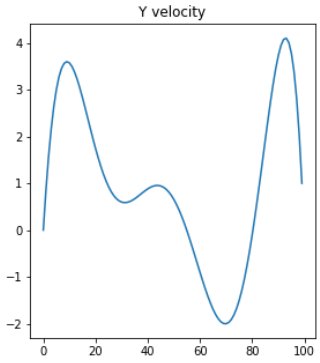
Plots of Individual Parameters

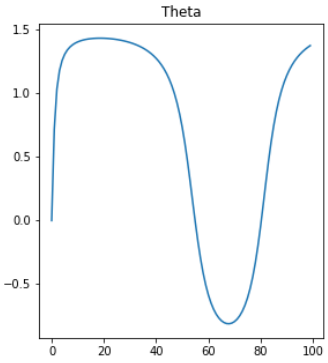
The following are the x, y, and velocity curves for each run.











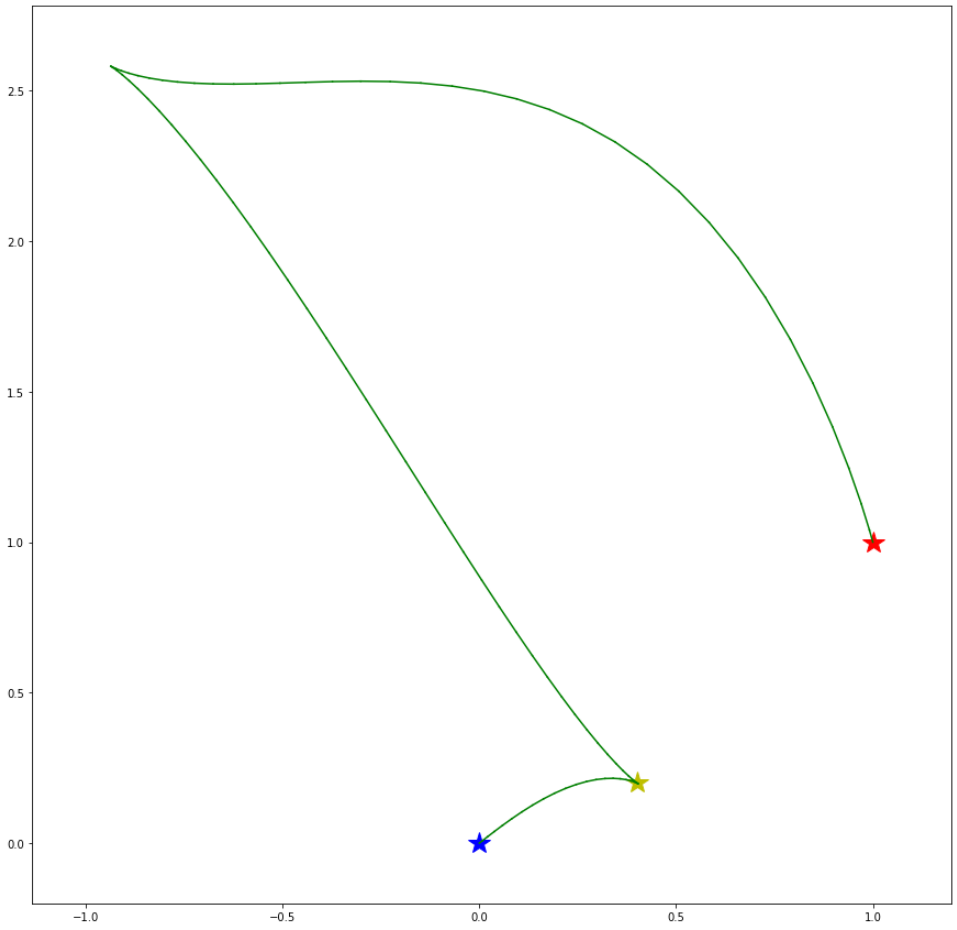
Run 2

Task Details

The robot starts at $x_0 = 0, y_0 = 0$ with velocity $\dot{x}_0 = 0.5, \dot{y}_0 = 0.5$ and moves towards the goal location $x_f = 1, y_f = 1$ with velocity $\dot{x}_0 = 0.2, \dot{y}_0 = -1.0$. We pass through the waypoint $x_f = 0.4, y_f = 0.2$ with velocity $\dot{x}_0 = 0.5, \dot{y}_0 = -0.3$

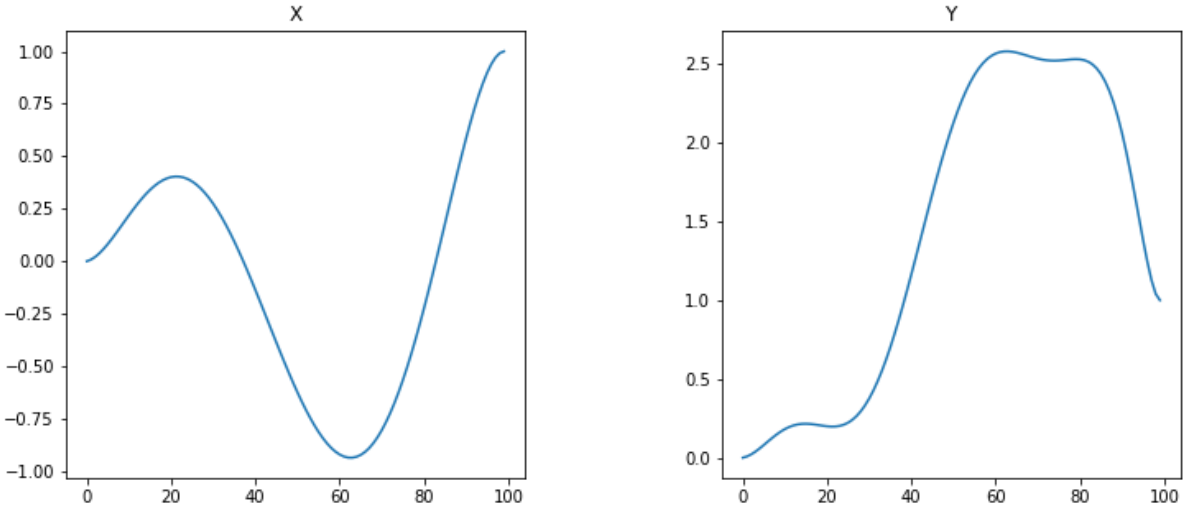
```
b = BernsteinPlanner(
  BernsteinPlanner.Constraint(x=0.0, y=0.0, dx=0.5, dy=0.5, t=0.0),
  BernsteinPlanner.Constraint(x=1.0, y=1.0, dx=0.2, dy=-1.0, t=1.0),
  BernsteinPlanner.Constraint(x=0.4, y=0.2, dx=0.5, dy=-0.3, t=0.2),
)
```

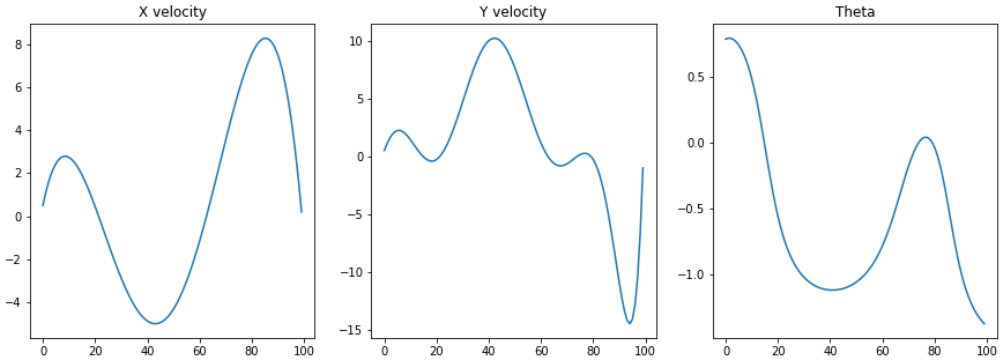
Trajectories Computed



Plots of Individual Parameters

The following are the x, y, and velocity curves for each run.





Run 3

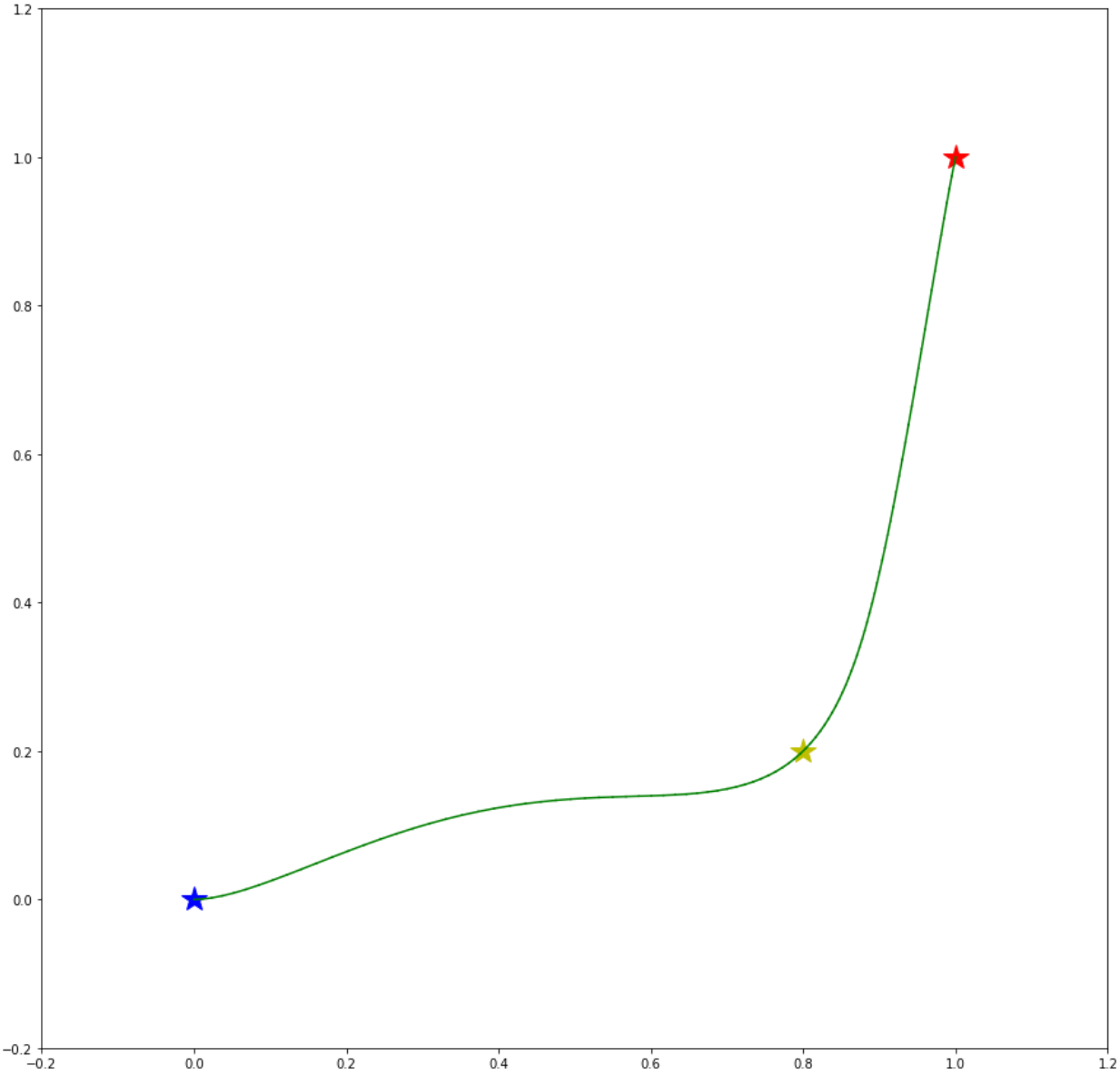
Task Details

The robot starts at $x_0 = 0$, $y_0 = 0$ with velocity $\dot{x}_0 = 1.0$, $\dot{y}_0 = 0.0$ and moves towards the goal location $x_f = 1$, $y_f = 1$ with velocity $\dot{x}_0 = 1.0$, $\dot{y}_0 = 0.2$. We pass through the waypoint $x_f = 0.8$, $y_f = 0.2$ with velocity $\dot{x}_0 = 0.7$, $\dot{y}_0 = 0.7$

```
b = BernsteinPlanner(
  BernsteinPlanner.Constraint(x=0.0, y=0.0, dx=1.0, dy=0.0, t=0.0),
  BernsteinPlanner.Constraint(x=1.0, y=1.0, dx=0.2, dy=1.0, t=1.0),
```

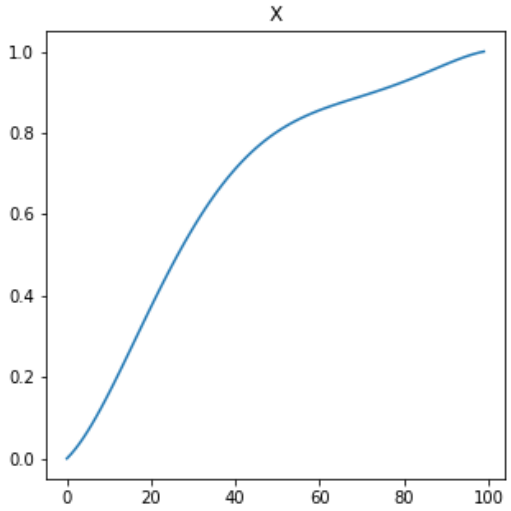
```
  BernsteinPlanner.Constraint(x=0.8, y=0.2, dx=0.7, dy=0.7, t=0.5),
)
```

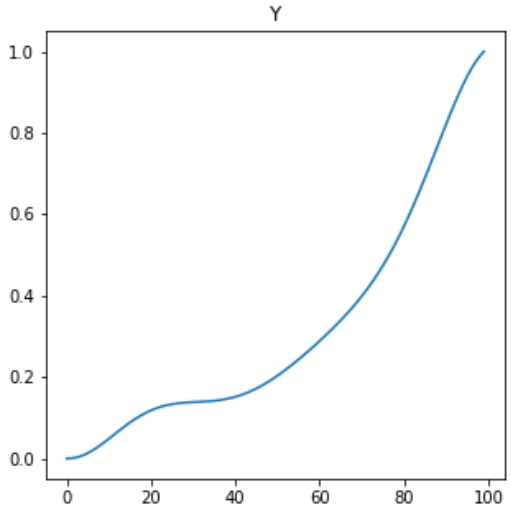
Trajectories Computed

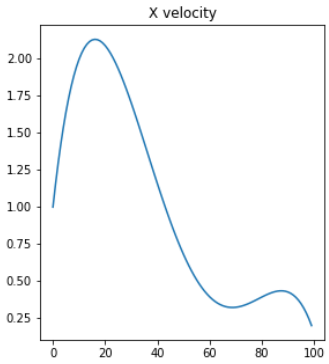


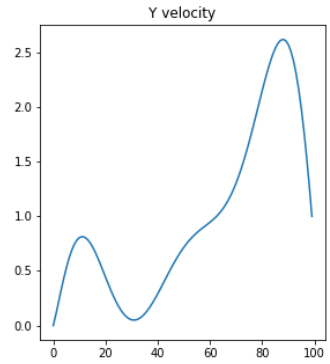
Plots of Individual Parameters

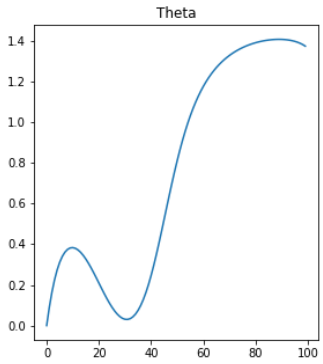
The following are the x, y, and velocity curves for each run.











Question 1

> How would you use Bernstein polynomials in the case of multi-rotor UAVs? (code not required, a brief explanation is sufficient)

A multi-rotor UAV, here considered to be more than 3 rotors, will be a holonomic robot. Non-holonomic constraints like $\dot{y} = \dot{x}\tan(\theta)$ would not apply. Therefore, we can solve for each of the axes independently.

The independent solutions for each axis will be computed purely in terms of the Bernstein polynomials and their derivatives given whether we have position or velocity constraints.

The equations, for all axes, will look like the following:

$$\begin{bmatrix} x_i \\ \dot{x_i} \end{bmatrix} = \begin{bmatrix} B_0(\mu) & B_1(\mu) & \cdots & B_n(\mu) \\ \dot{B}_0(\mu) & \dot{B}_1(\mu) & \cdots & \dot{B}_n(\mu) \end{bmatrix} * \begin{bmatrix} w_0^x \\ w_1^x \\ \cdots \\ w_n^x \end{bmatrix}$$

$$\begin{bmatrix} y_i \\ \dot{y_i} \end{bmatrix} = \begin{bmatrix} B_0(\mu) & B_1(\mu) & \cdots & B_n(\mu) \\ \dot{B}_0(\mu) & \dot{B}_1(\mu) & \cdots & \dot{B}_n(\mu) \end{bmatrix} * \begin{bmatrix} w_0^y \\ w_1^y \\ \cdots \\ w_n^y \end{bmatrix}$$

$$\begin{bmatrix} z_i \\ \dot{z_i} \end{bmatrix} = \begin{bmatrix} B_0(\mu) & B_1(\mu) & \cdots & B_n(\mu) \\ \dot{B}_0(\mu) & \dot{B}_1(\mu) & \cdots & \dot{B}_n(\mu) \end{bmatrix} * \begin{bmatrix} w_0^z \\ w_1^z \\ \cdots \\ w_n^z \end{bmatrix}$$

Using Bernstein Polynomial for representing smooth trajectories along each axis is a choice that we have made as a part of our method, it’s not necessitated by some constraints here.

Question 2

> What changes/constraints are to be introduced to accommodate more waypoints?

To add more waypoints in the problem, we will have to add constraints, i.e. rows to the matrix when computing both the W_x and the W_k for the corresponding Bernstein polynomials.

These rows would be of the following form for the x-matrix:

$$\begin{bmatrix} x_{w_i} \end{bmatrix} = \begin{bmatrix} B_0(\mu) & B_1(\mu) & \cdots & B_n(\mu) \end{bmatrix} * \begin{bmatrix} w_0^x \\ w_1^x \\ \cdots \\ w_n^x \end{bmatrix}$$

And correspondingly for the y coordinates in the k-matrix:

$$\begin{bmatrix} y_{w_i} \end{bmatrix} = \begin{bmatrix} F_0(\mu) & F_1(\mu) & \cdots & F_n(\mu) \end{bmatrix} * \begin{bmatrix} w_0^k \\ w_1^k \\ \cdots \\ w_n^k \end{bmatrix}$$

Adding these rows will bring the trajectory close to these waypoints. We can increase the value of n, which is the degree of the Bernstein polynomials, to be able to go even closer to the waypoints.

Solving the problem with these new constraints and higher degrees (let n be equal to the number of constraints for passing through exactly) will make the trajectory pass through the waypoints.

Question 3

> How would you avoid collisions of a non-holonomic robot with a trajectory fitted using Bernstein polynomials, with a static obstacle? (no time for time-scaling, simulate a video showing avoidance of a single static obstacle)

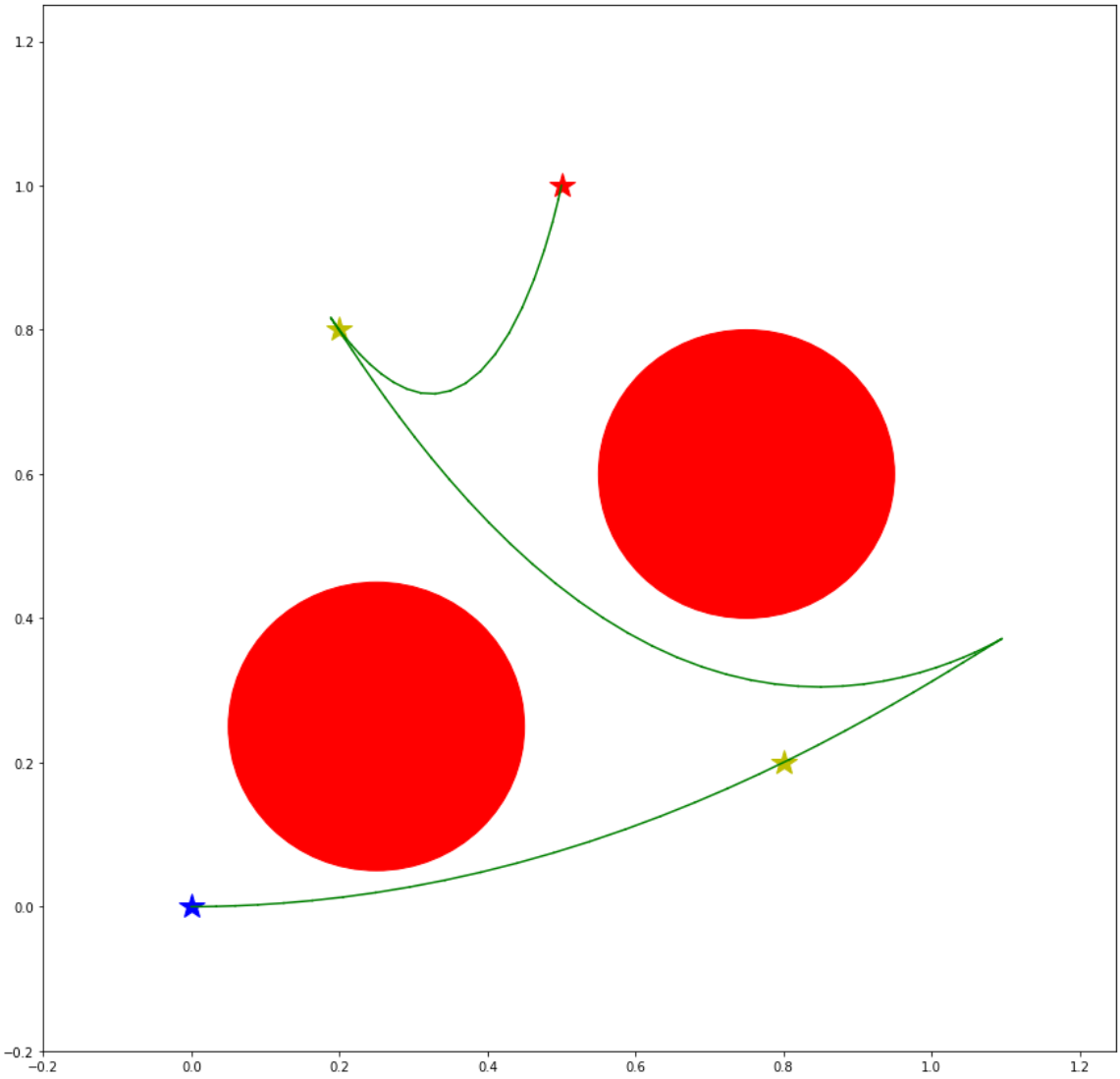
The core idea is to add waypoints in a way to push the trajectory to a path where there are no obstacles. Finding these points can be done in the following ways:

- Simulate the Bernstein path without waypoints. Detect all obstacles the path collides with. Find points on the exterior of this obstacle near the collision point, add those as waypoints in the Bernstein constraints, and re-simulate the paths, iteratively reducing the collision region.
- Run a simple RRT or other Tree Search algorithm to find a path that stays away from obstacles, and then sample some points along that path to fit a Bernstein

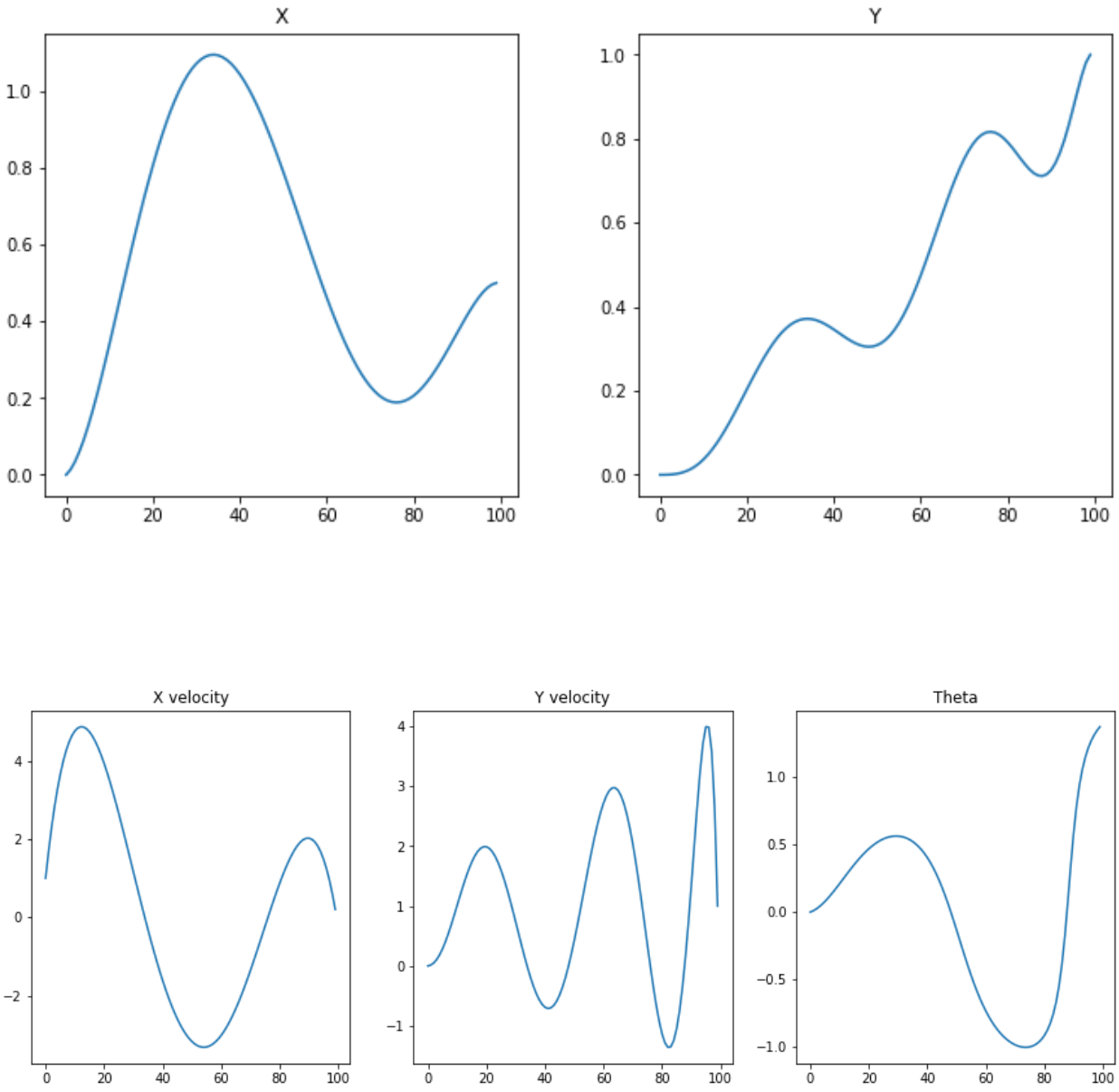
trajectory through it.

We have manually added waypoints on the paths the avoid the obstacles, the simulation videos for the same are attached with the assignment. Here are the images of the final trajectory.

Resultant Trajectory



Parameter Plots



https://drive.google.com/file/d/1Sa9HAK5RQmGLT4nXUIGAqD7eoiXF1tHj/view?usp=sharing

Run Configuration

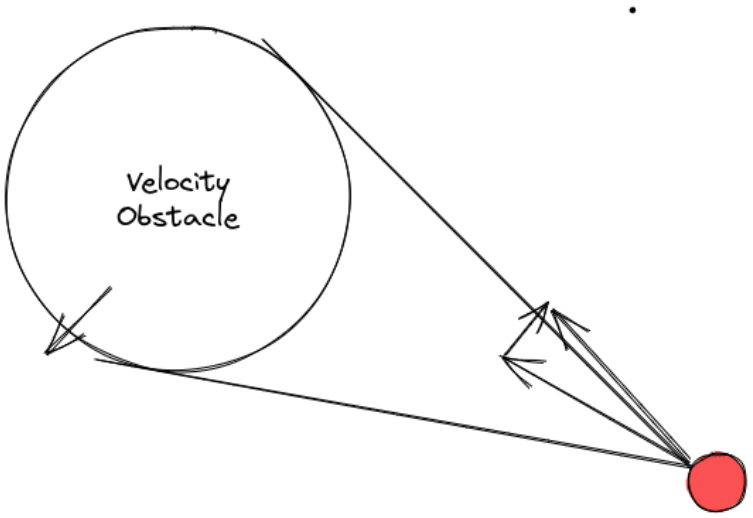
```
b = BernsteinPlanner(
  pos_0=BernsteinPlanner.Constraint(x=0.0, y=0.0, dx=1.0, dy=0.0, t=0.0),
  pos_f=BernsteinPlanner.Constraint(x=0.5, y=1.0, dx=0.2, dy=1.0, t=1.0),
  pos_c1=BernsteinPlanner.Constraint(x=0.2, y=0.8, dx=None, dy=None, t=0.8),
  pos_c2=BernsteinPlanner.Constraint(x=0.8, y=0.2, dx=None, dy=None, t=0.2),
)
```

Question 4

> How can we ensure that multiple non-holonomic robots with trajectories approximated using Bernstein polynomials do not collide? (intuitive understanding sufficient, no need for code)

There are many ways to do multi-robot obstacle avoidance, here is a brief outline of the possible solutions:

- Each robot can be treated as a velocity obstacle, and we can add the minimum deviation velocity required to escape the collision cone of every other robot estimated from vision and other sensors at each timestep to attempt to keep the robots on the original course while also not colliding.
- Time Scaling can be used to slow down or speed up such that any velocity obstacle can be avoided by escaping the collision cone. Optimal Reciprocal Collision Avoidance (ORCA) is one such method used popularly for multi-robot systems.
- We can estimate current positions and velocities for all robots, and then put the required separation into the constraints of a Quadratic Optimization Problem. This problem can be either linearized or be converted to one in polar coordinates. The Alternating Direction Method of Multipliers (ADMM) is used to do an outer non-linear optimization and an inner linear constraint optimization.



A Schematic of how adding transverse velocity or decreasing the in-line velocity can help escape collision cones.

Question 5

> What are the advantages of using Bernstein polynomials over other approximation techniques?

Bernstein polynomials for representing trajectories can be compared to the following other representation methods:

- A sequence of Points: This is hard to work with given a non-holonomic robot since we don't have the controls required to go from one point to the next, and constraints cannot be incorporated into it.
- Arbitrary Polynomials: They are pretty similar in use to Bernstein, but their derivative computations involve more product operations and hence they are less feasible.
- Arbitrary Functions: Actually representing the trajectory in functional form, like $y = x\tan(\theta)$ can make these functions hard to manipulate. Computing Derivatives of these functions is hard, same with their products, sums, and integrals.

The benefit of using Bernstein polynomials is that its **derivative is another Bernstein polynomials, and the same applies to its integral, products, sums, and any composition thereof**.

Compared to polynomials which have all the same features, the derivative is easier to compute since it does not require multiplication with different constants through the vector, it just has subtractions together with multiplication by an overall constant, which makes it computationally efficient over the other method.

$$\frac{\partial B_n^{N_{max}}(t)}{\partial t} = n(B_{n-1}^{N_{max}-1} - B_{n-1}^{N_{max}})$$

Therefore, Bernstein Polynomials and Bezier Splines are used as standards for trajectory representation.

Question 6

> BONUS: Try with higher/lower order of Bernstein polynomials and report your observations.

Higher-order Bernstein polynomials can better fit any trajectory satisfying a higher number of constraints.

However, the downside is that if we use too high a degree for the Bernstein polynomials, we can get a very wavy trajectory, and compromising a lot on smoothness may also increase the distance traveled.