Assignment-2A: Fitting polynomials to non-holonomic trajectories

EC4.403: Robotics: Planning and Navigation

Abhayram A. Nair(2019102017)

Aravind Narayanan(2019102014)

NOTE: Red Star refers to the finish position, green star refers to the waypoints, and initial position is referred to by the start of the dotted blue lines in all the output figures.

Question 1:

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

Answer:

In the multi-rotor UAVs, we have to plan the trajectory using Bernstein polynomials in a similar fashion fundamentally to the nonholonomic car example. We have a 3-D space so we obtain the Bernstein polynomial equations for x,y,z where the **constraints will be in terms of roll, pitch, yaw, velocity at each waypoint** through which we can infer the controls.

The acceleration along with gravity acting will be obtained through the angles and velocity essentially like an acceleration control but with the equation being estimated by the Bernstein polynomials..

Question 2:

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

Answer:

We notice that as we accomodate more way points, there are more constraints added to the system making it even more of an overly over-constrained system as the number of rows are increased in Bx and Bk as waypoints increase.

Modification:

1. For the Bx matrix, we add two rows $Bdot_{w_i}(t)$, $B_{w_i}(t)$ for each increase in waypoint.

2. For the Ax matrix, we add two rows x_{w_i} , $xdot_{w_i}$ for each increase in waypoint.

3. For the Bk matrix, we add two rows $B_{w_I}(t)$, $Coeff_{w_i}$ for each increase in waypoint.

4. For the Ak matrix, we add two rows k_{w_i}, y_{w_i} for each increase in waypoint.

- 1 waypoint:
    - Ax, Bx

```
B = np.array([[Btw[1], Btw[2], Btw[3], Btw[4]],
              [Bdotto[1], Bdotto[2], Bdotto[3], Bdotto[4]],
              [Bdottf[1], Bdottf[2], Bdottf[3], Bdottf[4]],
              [Bdottw[1], Bdottw[2], Bdottw[3], Bdottw[4]]])

A = np.array([[xw - xo*Btw[0] - xf*Btw[5]],
              [xdoto - xo*Bdotto[0] - xf*Bdotto[5]],
              [xdotf - xo*Bdottf[0] - xf*Bdottf[5]],
              [xdotw - xo*Bdottw[0] - xf*Bdottw[5]]])
```

    - Ak, Bk

```
B = np.array([[Btw[1], Btw[2], Btw[3], Btw[4]],
              [Bdotto[1], Bdotto[2], Bdotto[3], Bdotto[4]],
              [Bdottf[1], Bdottf[2], Bdottf[3], Bdottf[4]],
              [coefftf[1], coefftf[2], coefftf[3], coefftf[4]],
              [coefftw[1], coefftw[2], coefftw[3], coefftw[4]],
              [coeffto[1], coeffto[2], coeffto[3], coeffto[4]]])

A = np.array([[kw - ko*Btw[0] - kf*Btw[5]],
              [kdoto - ko*Bdotto[0] - kf*Bdotto[5]],
              [kdotf - ko*Bdottf[0] - kf*Bdottf[5]],
              [yf - ko*coefftf[0] - kf*coefftf[5] - yo],
              [yw - ko*coefftw[0] - kf*coefftw[5] - yo],
              [yo - ko*coeffto[0] - kf*coeffto[5] - yo]])
```

- 2 waypoints:
    - Ax, Bx

```
B = np.array([[Btw1[1], Btw1[2], Btw1[3], Btw1[4]],
              [Bdotto[1], Bdotto[2], Bdotto[3], Bdotto[4]],
              [Bdottf[1], Bdottf[2], Bdottf[3], Bdottf[4]],
              [Btw2[1], Btw2[2], Btw2[3], Btw2[4]],
              [Bdottw1[1], Bdottw1[2], Bdottw1[3], Bdottw1[4]],
              [Bdottw2[1], Bdottw2[2], Bdottw2[3], Bdottw2[4]]])

A = np.array([[xw1 - xo*Btw1[0] - xf*Btw1[5]],
              [xdoto - xo*Bdotto[0] - xf*Bdotto[5]],
              [xdotf - xo*Bdottf[0] - xf*Bdottf[5]],
              [xw2 - xo*Btw2[0] - xf*Btw2[5]],
              [xdotw1 - xo*Bdottw1[0] - xf*Bdottw1[5]],
              [xdotw2 - xo*Bdottw2[0] - xf*Bdottw2[5]]])
```

    - Ak, Bk

```
B = np.array([[Btw2[1], Btw2[2], Btw2[3], Btw2[4]],
              [Btw1[1], Btw1[2], Btw1[3], Btw1[4]],
              [Bdotto[1], Bdotto[2], Bdotto[3], Bdotto[4]],
              [coefftw2[1], coefftw2[2], coefftw2[3], coefftw2[4]],
              [Bdottf[1], Bdottf[2], Bdottf[3], Bdottf[4]],
              [coefftf[1], coefftf[2], coefftf[3], coefftf[4]],
              [coefftw1[1], coefftw1[2], coefftw1[3], coefftw1[4]],
              [coeffto[1], coeffto[2], coeffto[3], coeffto[4]]])

A = np.array([[kw2 - ko*Btw2[0] - kf*Btw2[5]],
              [kw1 - ko*Btw1[0] - kf*Btw1[5]],
              [kdoto - ko*Bdotto[0] - kf*Bdotto[5]],
              [yw2 - ko*coefftw2[0] - kf*coefftw2[5] - yo],
              [kdotf - ko*Bdottf[0] - kf*Bdottf[5]],
              [yf - ko*coefftf[0] - kf*coefftf[5] - yo],
              [yw1 - ko*coefftw1[0] - kf*coefftw1[5] - yo],
              [yo - ko*coeffto[0] - kf*coeffto[5] - yo]])
```

- 3 waypoints:

- Ax, Bx

```
B = np.array([[Btw1[1], Btw1[2], Btw1[3], Btw1[4]],
              [Bdotto[1], Bdotto[2], Bdotto[3], Bdotto[4]],
              [Bdottf[1], Bdottf[2], Bdottf[3], Bdottf[4]],
              [Btw2[1], Btw2[2], Btw2[3], Btw2[4]],
              [Btw3[1], Btw3[2], Btw3[3], Btw3[4]],
              [Bdottw1[1], Bdottw1[2], Bdottw1[3], Bdottw1[4]],
              [Bdottw2[1], Bdottw2[2], Bdottw2[3], Bdottw2[4]],
              [Bdottw3[1], Bdottw3[2], Bdottw3[3], Bdottw3[4]]])

A = np.array([[xw1 - xo*Btw1[0] - xf*Btw1[5]],
              [xdoto - xo*Bdotto[0] - xf*Bdotto[5]],
              [xdotf - xo*Bdottf[0] - xf*Bdottf[5]],
              [xw2 - xo*Btw2[0] - xf*Btw2[5]],
              [xw3 - xo*Btw3[0] - xf*Btw3[5]],
              [xdotw1 - xo*Bdottw1[0] - xf*Bdottw1[5]],
              [xdotw2 - xo*Bdottw2[0] - xf*Bdottw2[5]],
              [xdotw3 - xo*Bdottw3[0] - xf*Bdottw3[5]]])
```

- Ak, Bk

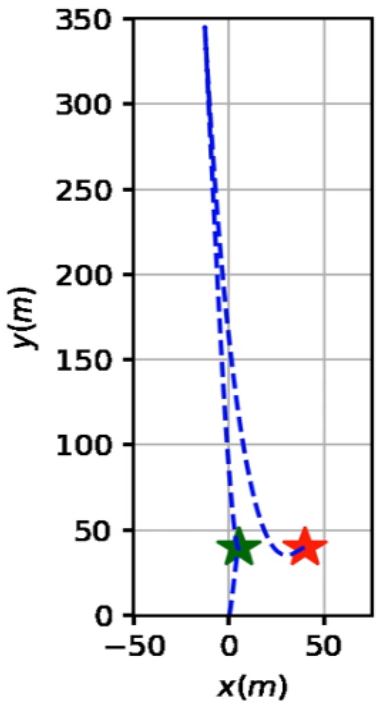
```
B = np.array([[Btw3[1], Btw3[2], Btw3[3], Btw3[4]],
              [Btw2[1], Btw2[2], Btw2[3], Btw2[4]],
              [Btw1[1], Btw1[2], Btw1[3], Btw1[4]],
              [Bdotto[1], Bdotto[2], Bdotto[3], Bdotto[4]],
              [coefftw3[1], coefftw3[2], coefftw3[3], coefftw3[4]],
              [coefftw2[1], coefftw2[2], coefftw2[3], coefftw2[4]],
              [Bdottf[1], Bdottf[2], Bdottf[3], Bdottf[4]],
              [coefftf[1], coefftf[2], coefftf[3], coefftf[4]],
              [coefftw1[1], coefftw1[2], coefftw1[3], coefftw1[4]],
              [coeffto[1], coeffto[2], coeffto[3], coeffto[4]]])

A = np.array([[kw3 - ko*Btw3[0] - kf*Btw3[5]],
              [kw2 - ko*Btw2[0] - kf*Btw2[5]],
              [kw1 - ko*Btw1[0] - kf*Btw1[5]],
              [kdoto - ko*Bdotto[0] - kf*Bdotto[5]],
              [yw3 - ko*coefftw3[0] - kf*coefftw3[5] - yo],
              [yw2 - ko*coefftw2[0] - kf*coefftw2[5] - yo],
              [kdotf - ko*Bdottf[0] - kf*Bdottf[5]],
              [yf - ko*coefftf[0] - kf*coefftf[5] - yo],
              [yw1 - ko*coefftw1[0] - kf*coefftw1[5] - yo],
              [yo - ko*coeffto[0] - kf*coeffto[5] - yo]])
```

Dimensions of Bx, Bk:

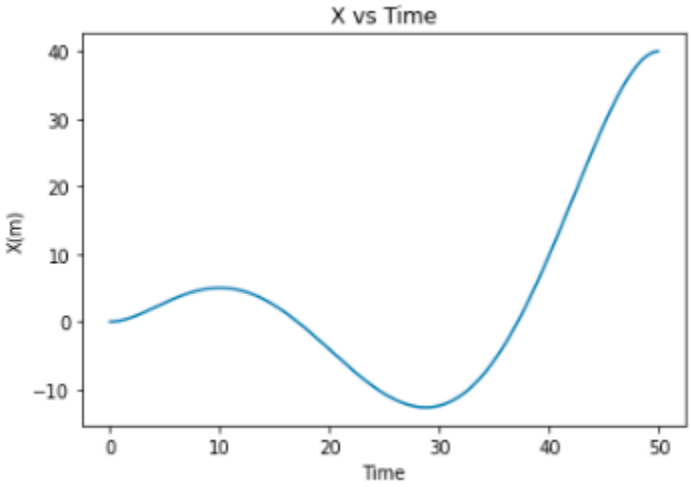
Number of waypoints	Bx (pxq)	Bk (mxn)	System
1	4x4	6x4	Over-constrained system as rows>cols
2	6x4	8x4	Over-constrained system as rows>cols
3	8x4	10x4	Over-constrained system as rows>cols

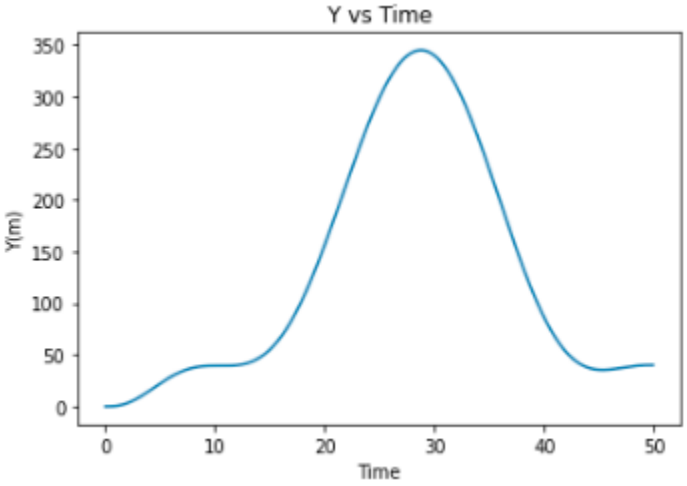
RESULTS

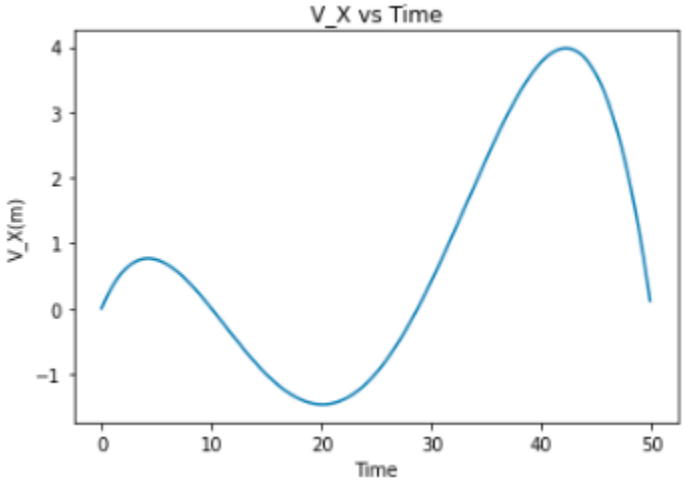
- One Waypoint:

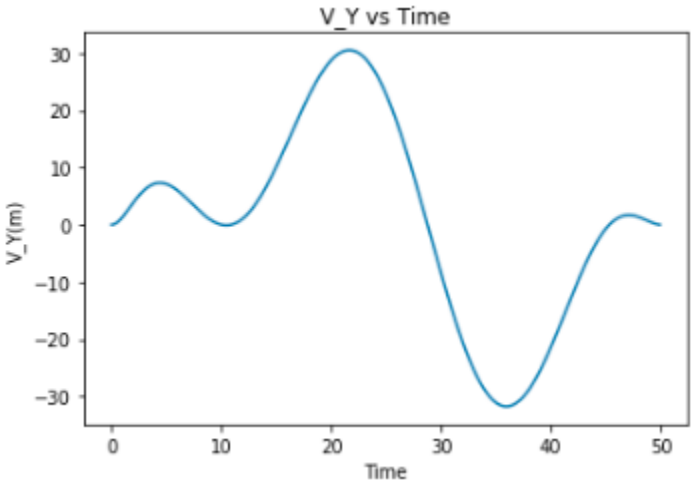


VALUES:


X vs Time
X(m)
Time
40
30
20
10
0
−10
0
10
20
30
40
50

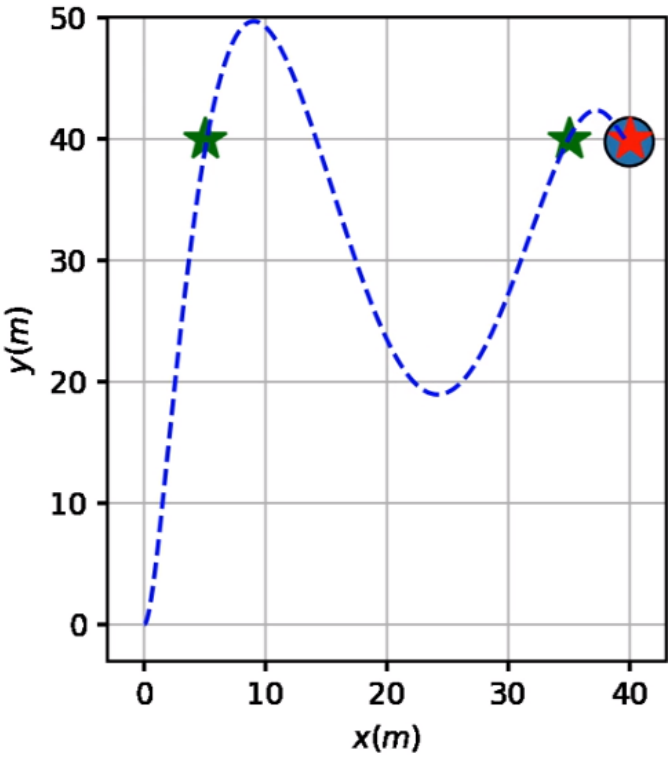

Y vs Time
Y(m)
Time
350
300
250
200
150
100
50
0
0
10
20
30
40
50


V_X vs Time
V_X(m)
Time
4
3
2
1
0
−1
0
10
20
30
40
50

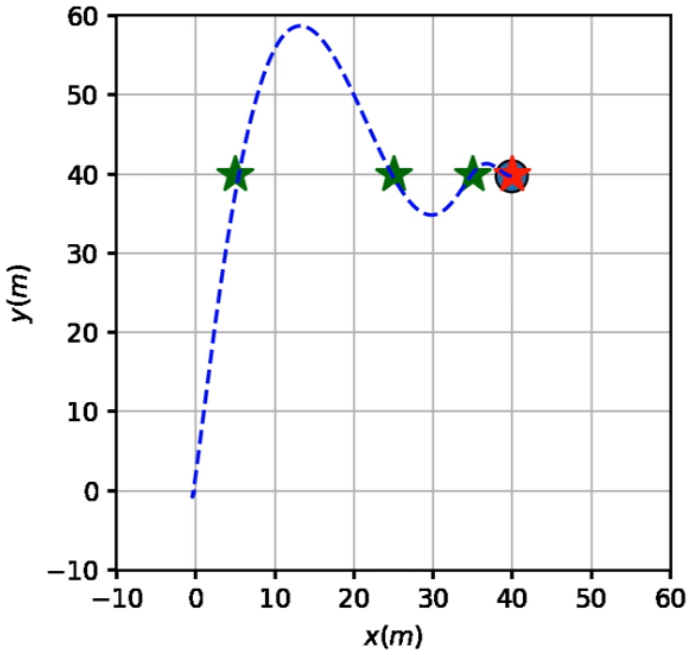


As expected, the initial and final velocities are 0 as per the given requirements.

- Two Waypoints:



- Three Waypoints:



Question 3:

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

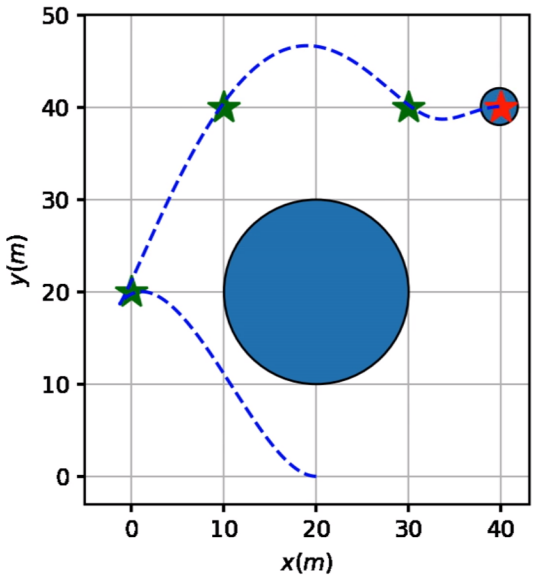
Answer:

To be able to go around the object, we introduce more waypoints around the object and then fit the Bernstein polynomial with respect to those, to be able to avoid a static obstacle.

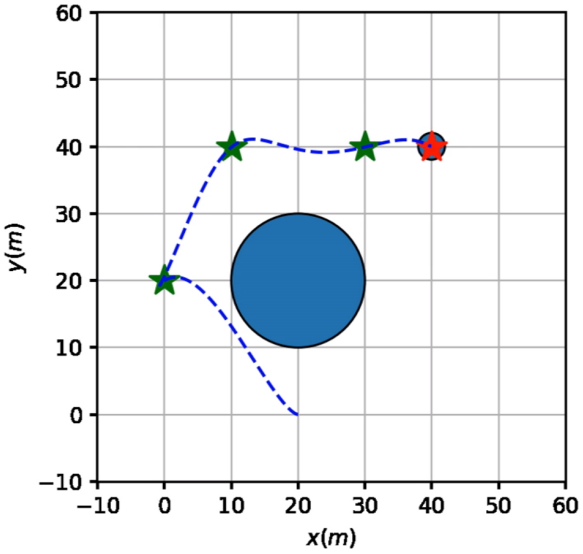
This is evident when we place a circular object and introduce waypoints around the object to make it convenient to reach the destination while avoiding the obstacle.

Example 1:

1) 5th Order

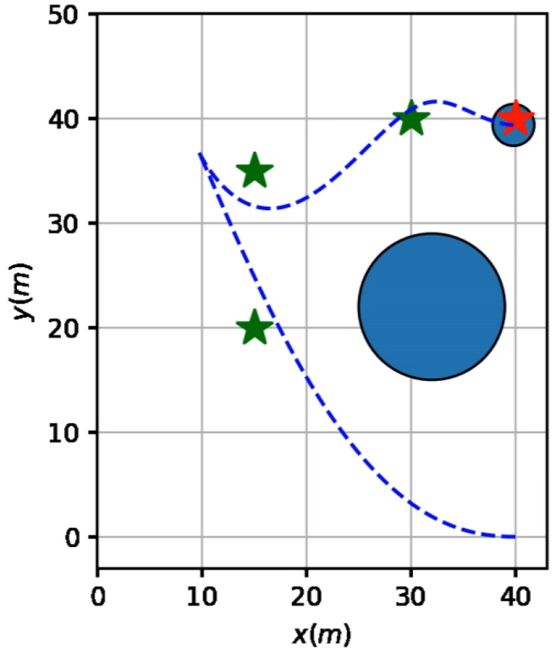


2) 7th Order

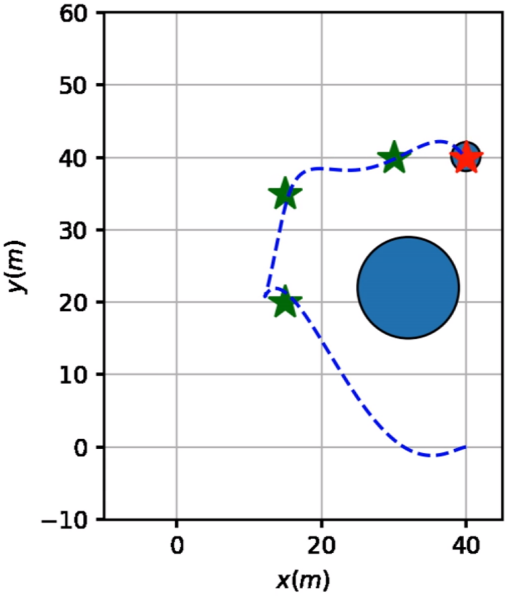


Example 2:

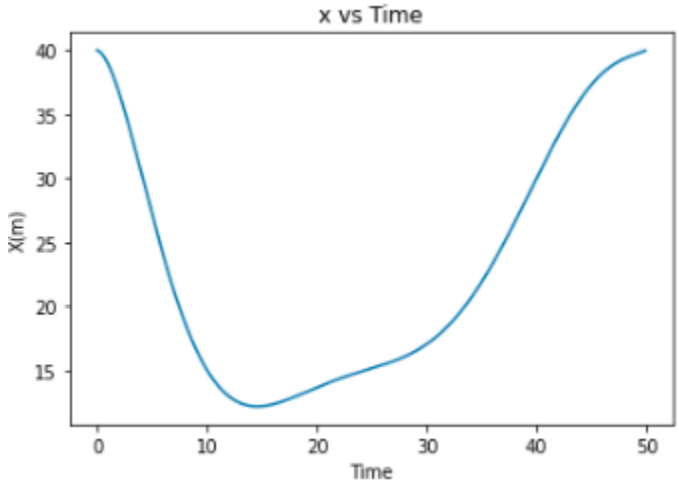
1) 5th Order

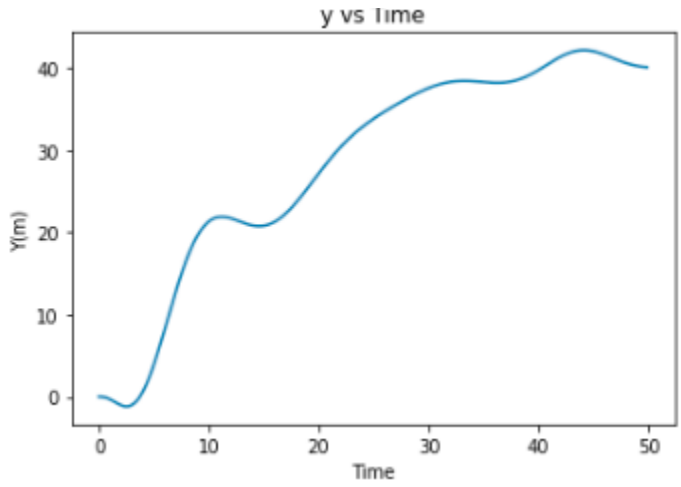


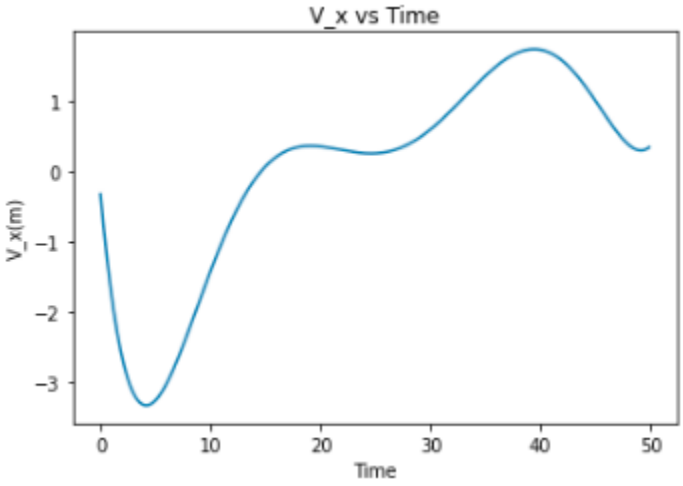
2) 7th Order

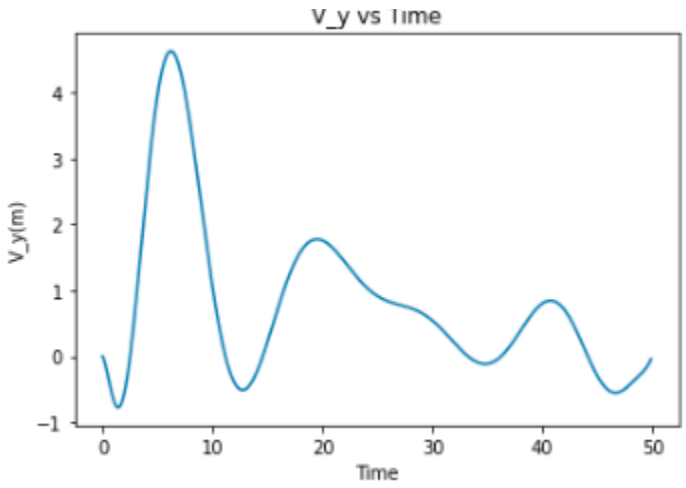


7th Order Example Values:









As expected, the initial and final velocities are 0 as per the given requirements.

Question 4:

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

Answer:

To be able to avoid collision of multiple nonholonomic trajectories approximated using Bernstein polynomials, we can **check for a potential collision at every single time step**. After that, we **slow down the robot** using the concept of **time-scaling** to be able to avoid collisions when multiple robots are involved.

Question 5:

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

Answer:

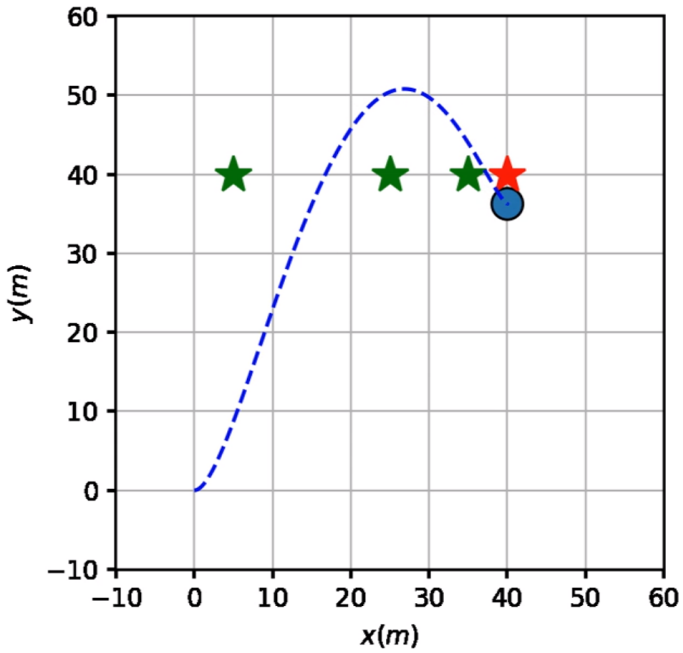
- The Bernstein polynomial and its derivatives can **efficiently approximate both the function (f) and its derivative (f_dot)**. This can also be extended for higher derivatives.
- The basis of the Berstein polynomials have a **numerically stable basis**.

BONUS:

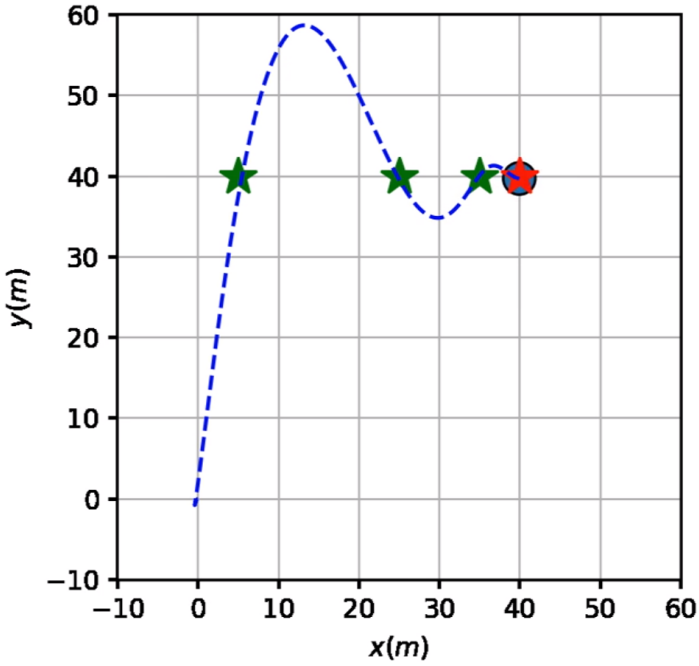
Try with higher/lower order of Bernstein polynomials and report your observations.

Answer:

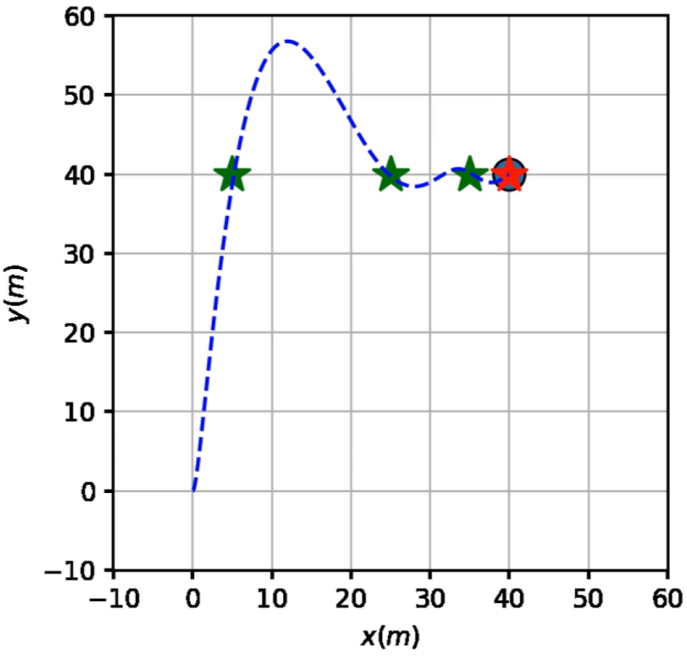
1) **3rd Order Polynomial**



2) 5th Order Polynomial



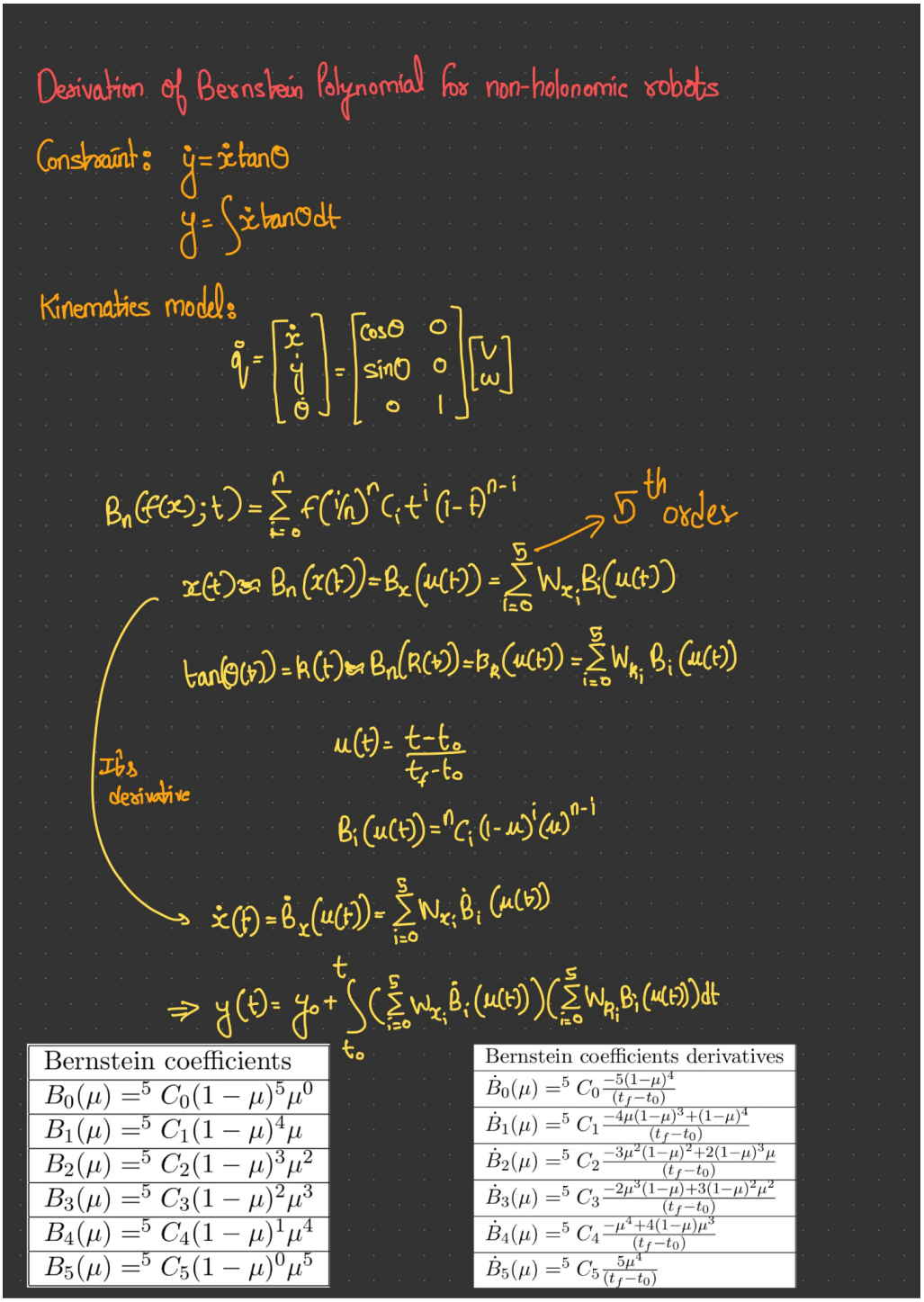
3) 7th Order Polynomial



Observations:

We notice that as the order increases, the trajectory generated is much more stable and converges better. This is expected as with more coefficients in high order, we can approximate the trajectory passing by the waypoints in a better fashion.

Derivation:



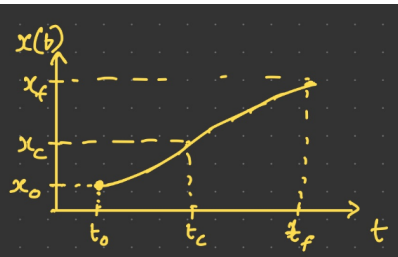
Bernstein coefficients
$B_0(\mu) = {}^5C_0(1-\mu)^5\mu^0$
$B_1(\mu) = {}^5C_1(1-\mu)^4\mu$
$B_2(\mu) = {}^5C_2(1-\mu)^3\mu^2$
$B_3(\mu) = {}^5C_3(1-\mu)^2\mu^3$
$B_4(\mu) = {}^5C_4(1-\mu)^1\mu^4$
$B_5(\mu) = {}^5C_5(1-\mu)^0\mu^5$

Bernstein coefficients derivatives
$\dot{B}_0(\mu) = {}^5C_0 \frac{-5(1-\mu)^4}{(t_f-t_0)}$
$\dot{B}_1(\mu) = {}^5C_1 \frac{-4\mu(1-\mu)^3+(1-\mu)^4}{(t_f-t_0)}$
$\dot{B}_2(\mu) = {}^5C_2 \frac{-3\mu^2(1-\mu)^2+2(1-\mu)^3\mu}{(t_f-t_0)}$
$\dot{B}_3(\mu) = {}^5C_3 \frac{-2\mu^3(1-\mu)+3(1-\mu)^2\mu^2}{(t_f-t_0)}$
$\dot{B}_4(\mu) = {}^5C_4 \frac{-\mu^4+4(1-\mu)\mu^3}{(t_f-t_0)}$
$\dot{B}_5(\mu) = {}^5C_5 \frac{5\mu^4}{(t_f-t_0)}$

To find the weights, (W_x)

$$A_x = B_x W_x$$

$$W_x = \text{pinv}(B_x) A_x$$



where,

$$A_x = \begin{bmatrix} x_{t_c} - W_{x_0} B_0(\mu(t_c)) - W_{x_5} B_5(\mu(t_c)) \\ \dot{x}_{t_0} - W_{x_0} \dot{B}_0(\mu(t_0)) - W_{x_5} \dot{B}_5(\mu(t_0)) \\ \dot{x}_{t_f} - W_{x_0} \dot{B}_0(\mu(t_f)) - W_{x_5} \dot{B}_5(\mu(t_f)) \\ \dot{x}_{t_c} - W_{x_0} \dot{B}_0(\mu(t_c)) - W_{x_5} \dot{B}_5(\mu(t_c)) \end{bmatrix}$$ → 1 waypoint.

$$B_x = \begin{bmatrix} B_1(\mu(t_c)) & B_2(\mu(t_c)) & B_3(\mu(t_c)) & B_4(\mu(t_c)) \\ \dot{B}_1(\mu(t_0)) & \dot{B}_2(\mu(t_0)) & \dot{B}_3(\mu(t_0)) & \dot{B}_4(\mu(t_0)) \\ \dot{B}_1(\mu(t_f)) & \dot{B}_2(\mu(t_f)) & \dot{B}_3(\mu(t_f)) & \dot{B}_4(\mu(t_f)) \\ \dot{B}_1(\mu(t_c)) & \dot{B}_2(\mu(t_c)) & \dot{B}_3(\mu(t_c)) & \dot{B}_4(\mu(t_c)) \end{bmatrix}$$

$$W_x = \begin{bmatrix} W_{x_1} \\ W_{x_2} \\ W_{x_3} \\ W_{x_4} \end{bmatrix} \quad \text{and} \quad W_{x_0} = x(t_0) = x_{t_0}$$

(W_k)

$$y(t) = y_0 + \int_{t_0}^{t} \left(\sum_{i=0}^{5} \left(W_{k_i} f_i(t, t_0, t_f, W_{x_1}, W_{x_2}, W_{x_3}, W_{x_4}, W_{x_5}) \right) \right)$$

$$y(t) = y_0 + \sum_{i=0}^{5} W_{k_i} F_i(t)$$

$$\hookrightarrow \int_{t_0}^{t} f_i(t, t_0, t_f, W_{x_1}, W_{x_2}, W_{x_3}, W_{x_4}, W_{x_5})\, dt$$

To get the weights,

$$k(t_0) = k_0 = \sum_{i=0}^{5} W_{k_i} B_i(u(t_0))$$

So, $W_{k_0} = k_0$,

This can be written as

$$A_k = B_k W_k$$

$$W_k = \text{pinv}(B_k) A_k$$

$$A_k = \begin{bmatrix} \dot{k}_{t_0} - W_{k_0}\dot{B}_0(\mu(t_0)) - W_{k_5}\dot{B}_5(\mu(t_0)) \\ \dot{k}_{t_f} - W_{k_0}\dot{B}_0(\mu(t_f)) - W_{k_5}\dot{B}_5(\mu(t_f)) \\ y_0 - W_{k_0}F_0 - W_{k_5}F_5 \\ y_f - W_{k_5}F_0 - W_{k_5}F_5 \end{bmatrix}$$

$$B_k = \begin{bmatrix} \dot{B}_{k_1}(\mu(t_0)) & \dot{B}_2(\mu(t_0)) & \dot{B}_3(\mu(t_0)) & \dot{B}_4(\mu(t_0)) \\ \dot{B}_1(\mu(t_f)) & \dot{B}_2(\mu(t_f)) & \dot{B}_3(\mu(t_f)) & \dot{B}_4(\mu(t_f)) \\ F_1(t_0) & F_2(t_0) & F_3(t_0) & F_4(t_0) \\ F_1(t_f) & F_2(t_f) & F_3(t_f) & F_4(t_f) \end{bmatrix}$$

$$W_k = \begin{bmatrix} W_{k_1} \\ W_{k_2} \\ W_{k_3} \\ W_{k_4} \end{bmatrix}$$

Now once W_x, W_k are determined, we can obtain $x(t), y(t), \theta(t)$

$$x(t) = \sum_{i=0}^{5} W_{x_i} B_i(u(t))$$

$$y(t) = y_0 + \sum_{i=0}^{5} W_{k_i} F_i(t)$$

$$\theta(t) = \text{atan}\left(\sum_{i=0}^{5} W_{k_i} B_i(u(t))\right)$$

NOTE: This can be extended for any n^{th} order Bernstein polynomial. Coefficients can be obtained by pascal's triangle