Mobile Robotics Assignment 5

Team ID 24

Roll No. 2019702002 Nivedita Rufus

Roll No. 2019702008 Sravya Vardhani S

Q) Trajectory planning using polynomial functions

Part (a)

Approach

The general equation for a fourth degree polynomial and its derivative is given by

$$x(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4$$
$$x'(t) = a_1 + 2a_2 t + 3a_3 t^2 + 4a_4 t^4$$

Trajectory of y with respect to time can be defined by a similar equation with different coefficients. Now when we substitute the conditions of the positions at start end and intermediate time steps (we have assumed the robot reaches the cookie in 2 seconds), we get the following matrix,

$$\begin{bmatrix} x_0 \\ x'_0 \\ x_5 \\ x'_5 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 5 & 25 & 125 & 625 \\ 0 & 1 & 10 & 75 & 500 \\ 1 & 2 & 4 & 8 & 16 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}$$

Y has a similar matrix and substituting the following values of x,y we get the coefficients of our fourth degree polynomial.

$$x_0 = 3, x'_0 = 0, x_5 = 9, x'_5 = 0, x_2 = 1$$
$$y_0 = 0, y'_0 = 0, y_5 = 9, y'_5 = 0, y_2 = 2.5$$

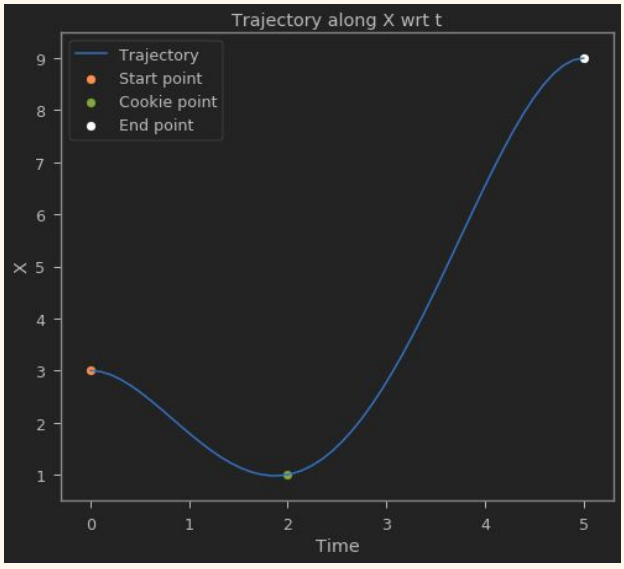
Code Snippet

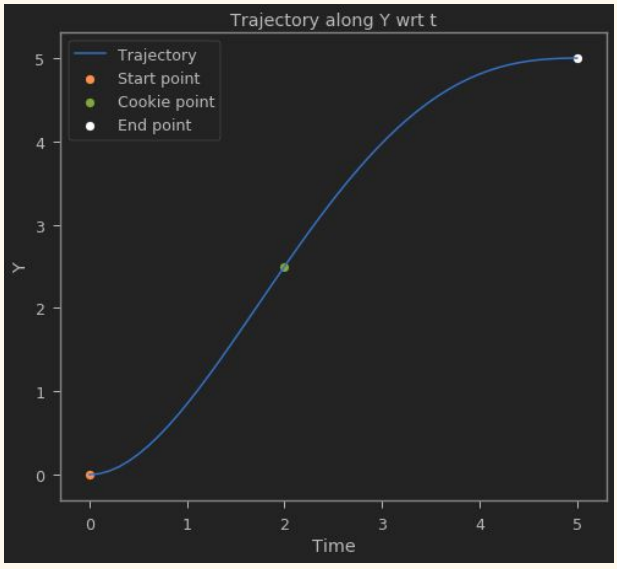
```
a =
np.array([[1,0,0,0,0],[0,1,0,0,0],[1,5,25,125,625],[0,1,10,75,500],[1,2,4,8
,16]])
b = np.array([3,0,9,0,1])
x = np.linalg.solve(a, b)
```

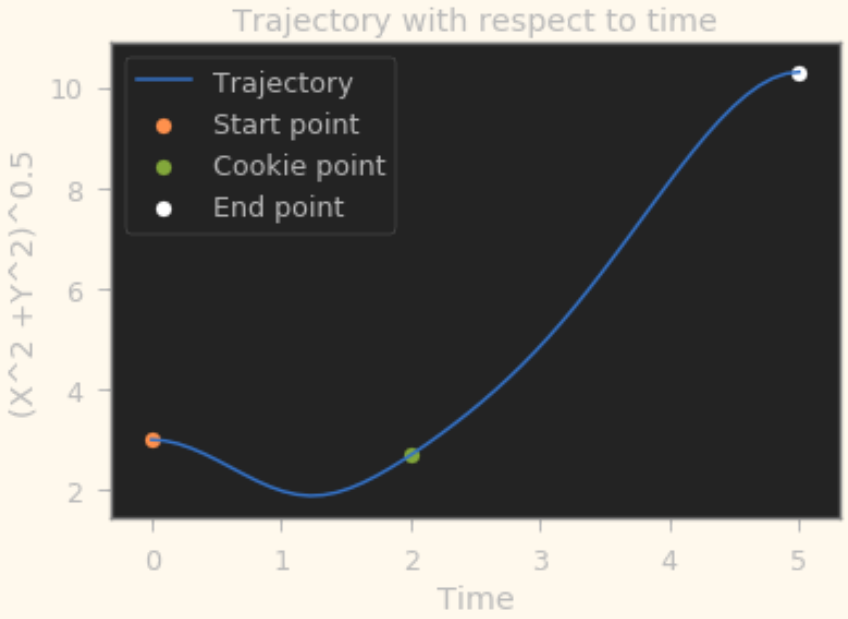
Outputs

The obtained plots are shown below.

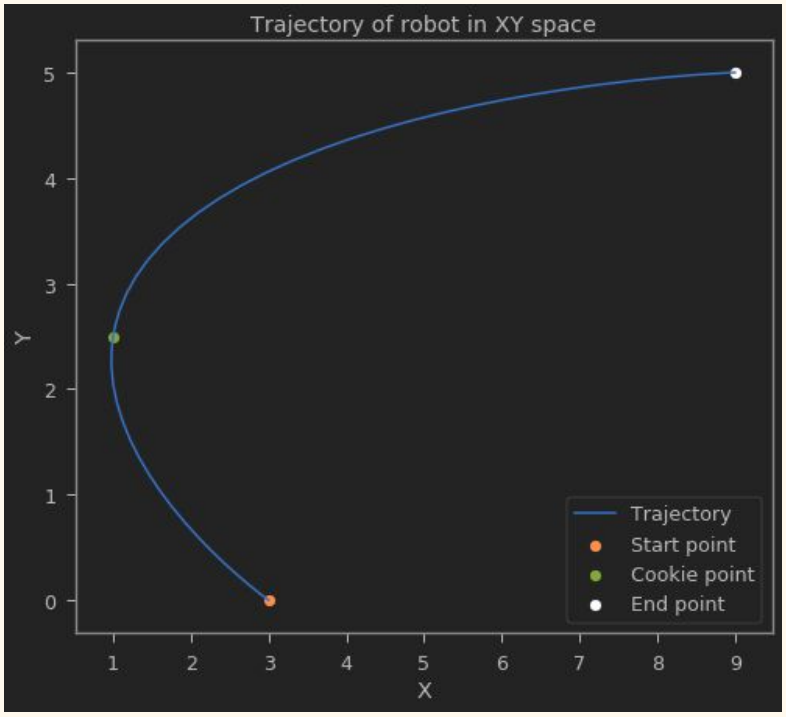
Trajectories along X, Y and XY with respect to time



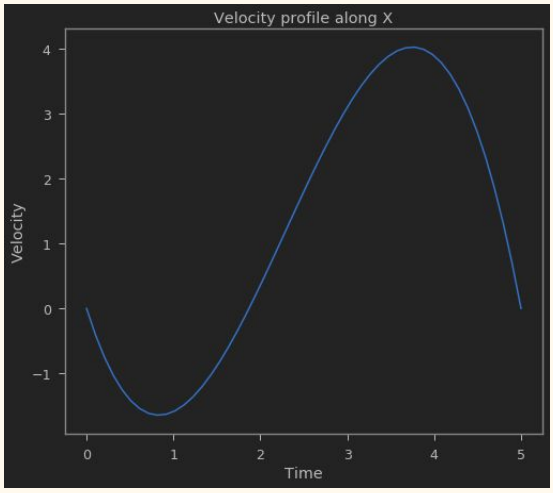


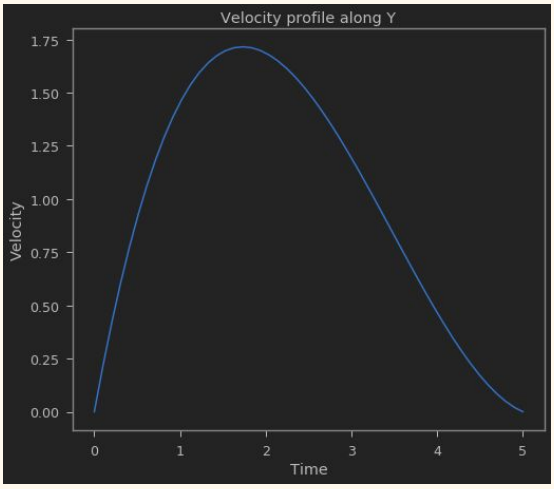


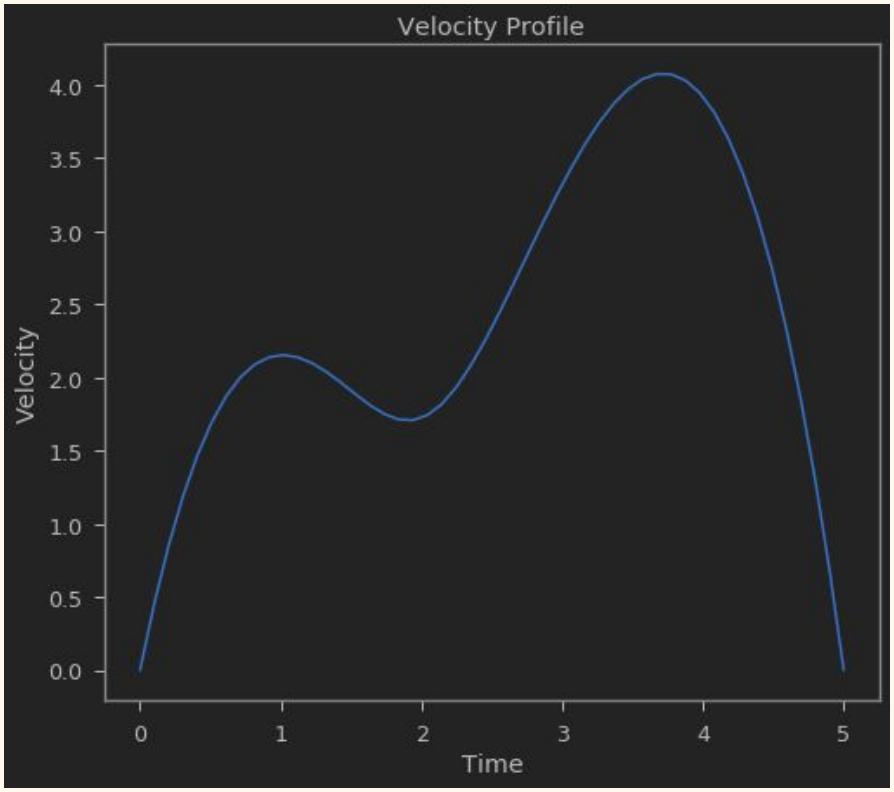
Trajectory of robot



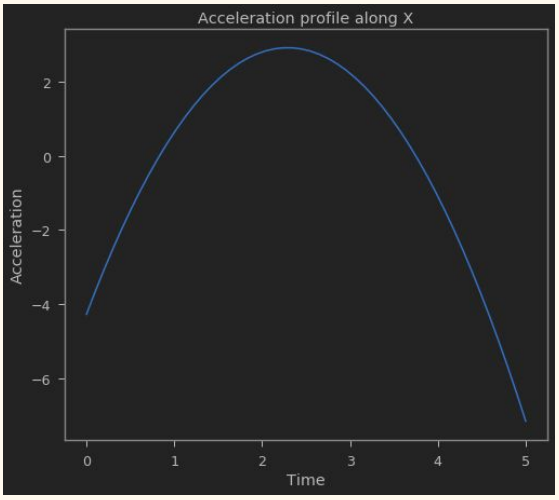
Velocity Profile of X, Y, XY with respect to time

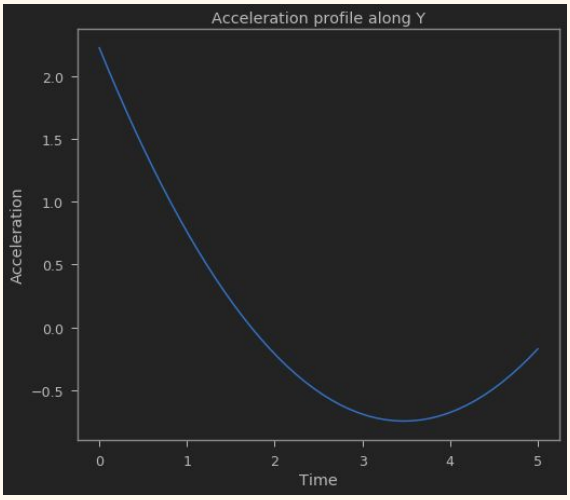


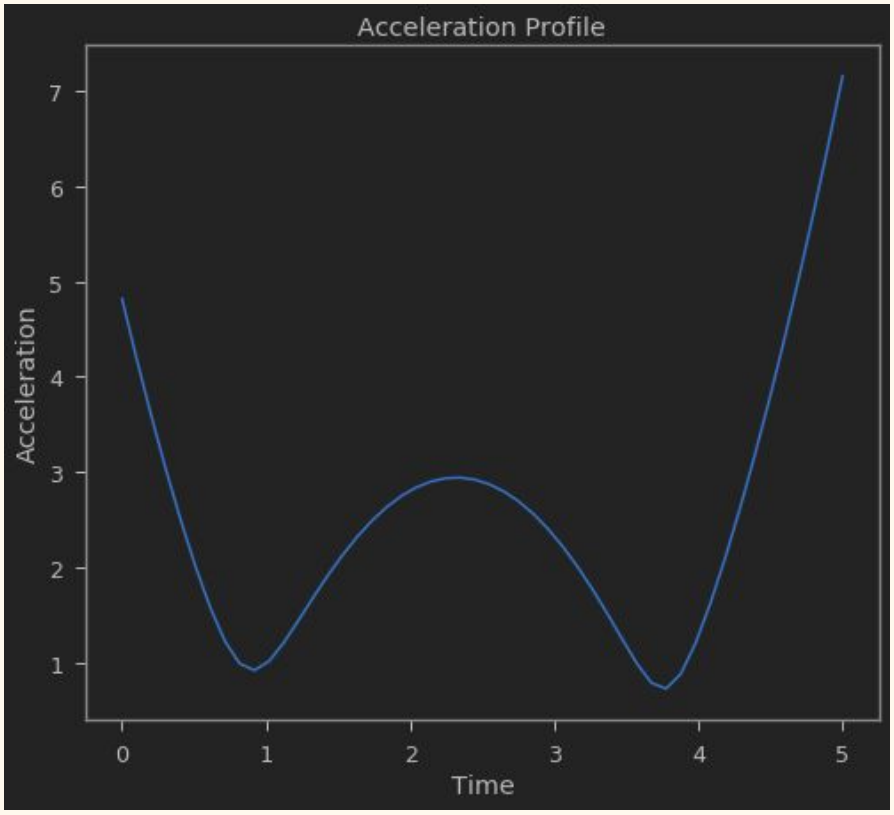




Acceleration Profile of X, Y, XY with respect to time







Part (b)

Approach

A n[th] degree Bernstein equation is given as

$$f(t) = \sum_{i=0}^{n} {}^{n}C_i \tau^i (1-\tau)^{n-i} p_i$$

A fifth degree polynomial and its first derivative is given as

Bernstein coefficients	$t = t_0$	$t = t_f$
$B_0(\mu) = {}^{5}C_0(1-\mu)^5\mu^0$	1	0
$B_1(\mu) = {}^{5}C_1(1-\mu)^4\mu$	0	0
$B_2(\mu) = {}^{5}C_2(1-\mu)^3\mu^2$	0	0
$B_3(\mu) = {}^{5}C_3(1-\mu)^2\mu^3$	0	0
$B_4(\mu) = {}^{5}C_4(1-\mu)^1\mu^4$	0	0
$B_5(\mu) = {}^{5}C_5(1-\mu)^0\mu^5$	0	1

Bernstein coefficients derivatives	$t = t_0, \mu = 0$	$t = t_f, \mu = 1$
$\dot{B}_0(\mu) = {}^{5}C_0 \frac{-5(1-\mu)^4}{(t_f - t_0)}$	$\frac{-5}{(t_f - t_0)}$	0
$\dot{B}_1(\mu) = {}^{5}C_1 \frac{-4\mu(1-\mu)^3 + (1-\mu)^4}{(t_f - t_0)}$	$\frac{5}{(t_f - t_0)}$	0
$\dot{B}_2(\mu) = {}^{5}C_2 \frac{-3\mu^2(1-\mu)^2 + 2(1-\mu)^3\mu}{(t_f - t_0)}$	0	0
$\dot{B}_3(\mu) = {}^{5}C_3 \frac{-2\mu^3(1-\mu) + 3(1-\mu)^2\mu^2}{(t_f - t_0)}$	0	0
$\dot{B}_4(\mu) = {}^{5}C_4 \frac{-\mu^4 + 4(1-\mu)\mu^3}{(t_f - t_0)}$	0	$\frac{-5}{(t_f - t_0)}$
$\dot{B}_5(\mu) = {}^{5}C_5 \frac{5\mu^4}{(t_f - t_0)}$	0	$\frac{5}{(t_f - t_0)}$

Where, $\mu(t) = \frac{t - t_0}{t_f - t_0}$

Substituting all the values in this table we get, for X,

$$\begin{bmatrix} 3.0 \\ 0 \\ 1.0 \\ 0 \\ 9.0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 \\ 0.07776 & 0.2592 & 0.3456 & 0.2304 & 0.0768 & 0.01024 \\ -0.1296 & -0.216 & 0 & 0.192 & 0.128 & 0.0256 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} W_{x0} \\ W_{x1} \\ W_{x2} \\ W_{x3} \\ W_{x4} \\ W_{x5} \end{bmatrix}$$

For Y,

$$\begin{bmatrix} 0 \\ 0 \\ 2.5 \\ 0 \\ 5.0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 \\ 0.07776 & 0.2592 & 0.3456 & 0.2304 & 0.0768 & 0.01024 \\ -0.1296 & -0.216 & 0 & 0.192 & 0.128 & 0.0256 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} W_{y0} \\ W_{y1} \\ W_{y2} \\ W_{y3} \\ W_{y4} \\ W_{y5} \end{bmatrix}$$

Code Snippet

```
def bernstein_poly(t, to = 0, tf = 5):
    B0= (1 - (t - to)/(-to + tf))**5
    B1=(5*(t - to)*(1 - (t - to)/(-to + tf))**4)/(-to + tf)
    B2= (10*(t - to)**2*(1 - (t - to)/(-to + tf))**3)/(-to + tf)**2
    B3= (10*(t - to)**3*(1 - (t - to)/(-to + tf))**2)/(-to + tf)**3
    B4=(5*(t - to)**4*(1 - (t - to)/(-to + tf)))/(-to + tf)**4
    B5= (t - to)**5/(-to + tf)**5
    return np.array([B0,B1,B2,B3,B4,B5])
```

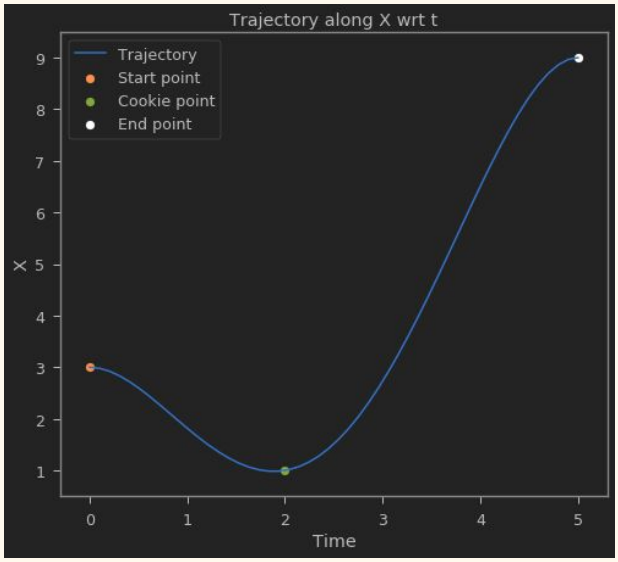
```
//Velocity profiles
def bernstein_der(t, to = 0, tf = 5):
    Bodot=-((5*(1 - (t - t0)/(-t0 + tf))**4)/(-t0 + tf))
```

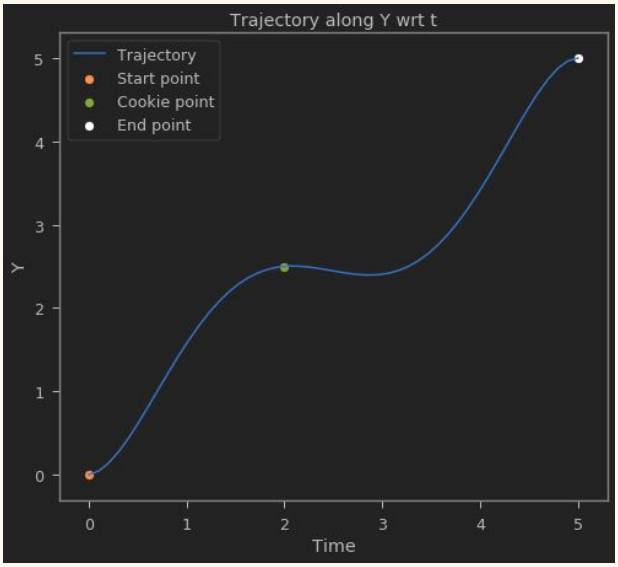
```
    B1dot=-((20*(t - t0)*(1 - (t - t0)/(-t0 + tf))**3)/(-t0 + tf)**2) +
(5*(1 - (t - t0)/(-t0 + tf))**4)/(-t0 + tf)
    B2dot=-((30*(t - t0)**2*(1 - (t - t0)/(-t0 + tf))**2)/(-t0 + tf)**3) +
(20*(t - t0)*(1 - (t - t0)/(-t0 + tf))**3)/(-t0 + tf)**2
    B3dot=-((20*(t - t0)**3*(1 - (t - t0)/(-t0 + tf)))/(-t0 + tf)**4) +
(30*(t - t0)**2*(1 - (t - t0)/(-t0 + tf))**2)/(-t0 + tf)**3
    B4dot=-((5*(t - t0)**4)/(-t0 + tf)**5) + (20*(t - t0)**3*(1 - (t -
t0)/(-t0 + tf)))/(-t0 + tf)**4
    B5dot=(5*(t - t0)**4)/(-t0 + tf)**5
    return np.array([Bodot,B1dot,B2dot,B3dot,B4dot,B5dot])
```

```
//Acceleration profiles
def bernstein_d_der(t, to = 0, tf = 5):
    Boddot=(20*(1 - (t - t0)/(-t0 + tf))**3)/(-t0 + tf)**2
    B1ddot=(60*(t - t0)*(1 - (t - t0)/(-t0 + tf))**2)/(-t0 + tf)**3 -
(40*(1 - (t - t0)/(-t0 + tf))**3)/(-t0 + tf)**2
    B2ddot=(60*(t - t0)**2*(1 - (t - t0)/(-t0 + tf)))/(-t0 + tf)**4 -
(120*(t - t0)*(1 - (t - t0)/(-t0 + tf))**2)/(-t0 + tf)**3 + (20*(1 - (t -
t0)/(-t0 + tf))**3)/(-t0 + tf)**2
    B3ddot=(20*(t - t0)**3)/(-t0 + tf)**5 - (120*(t - t0)**2*(1 - (t -
t0)/(-t0 + tf)))/(-t0 + tf)**4 + (60*(t - t0)*(1 - (t - t0)/(-t0 +
tf))**2)/(-t0 + tf)**3
    B4ddot=-((40*(t - t0)**3)/(-t0 + tf)**5) + (60*(t - t0)**2*(1 - (t -
t0)/(-t0 + tf)))/(-t0 + tf)**4
    B5ddot=(20*(t - t0)**3)/(-t0 + tf)**5
    return np.array([Boddot,B1ddot,B2ddot,B3ddot,B4ddot,B5ddot])
```

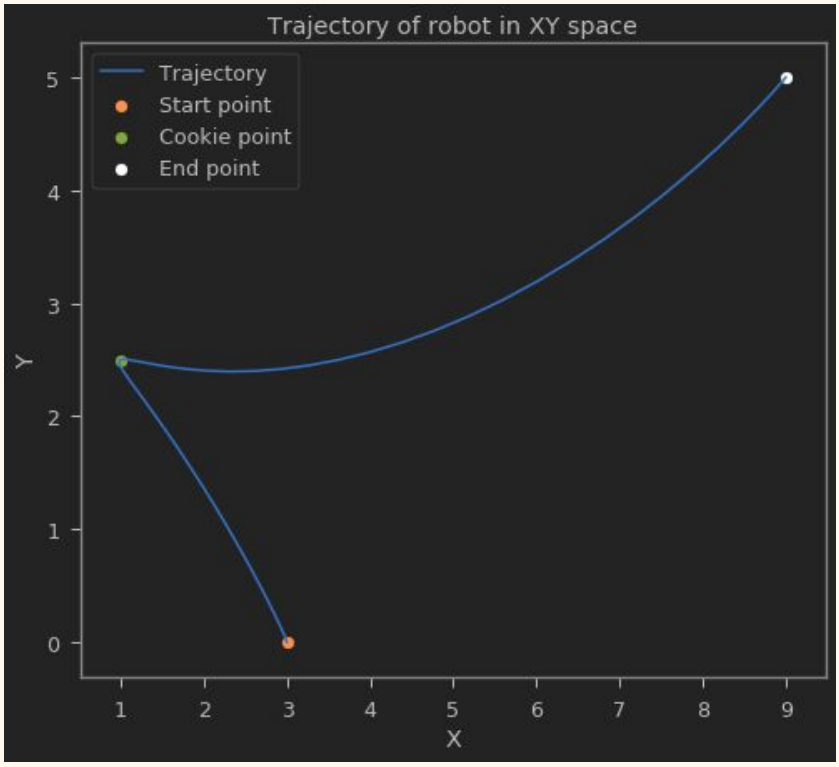
Outputs

Trajectories along X, Y and XY with respect to time

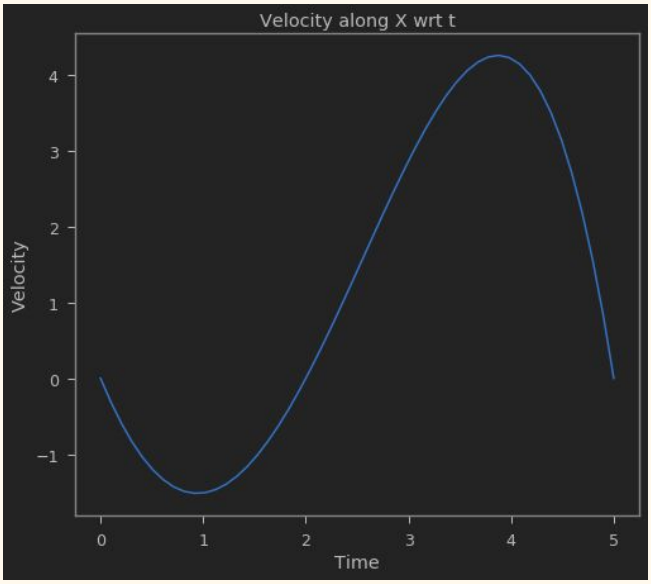


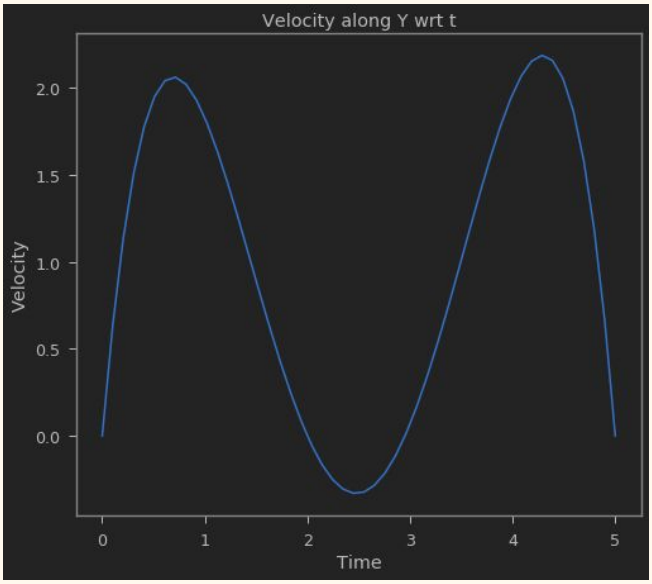


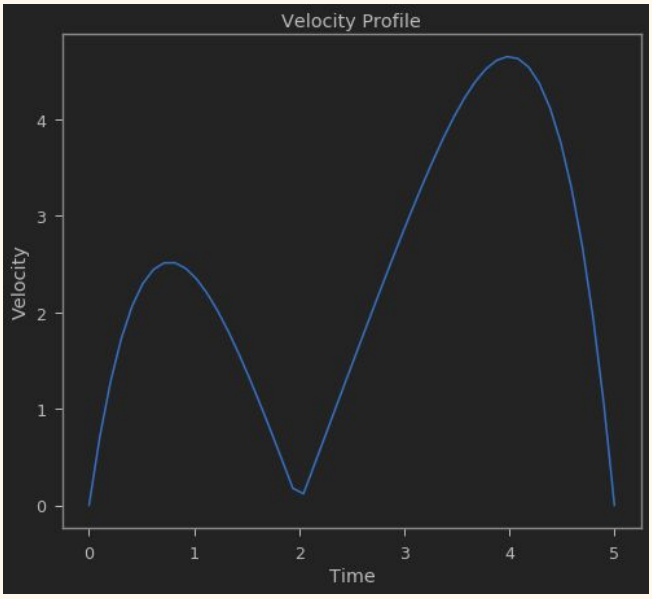
Trajectory of robot



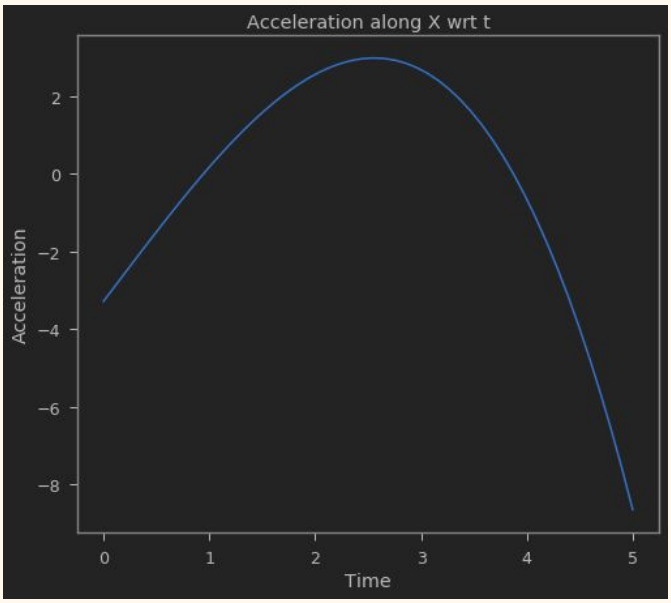
Velocity Profile of X, Y, XY with respect to time

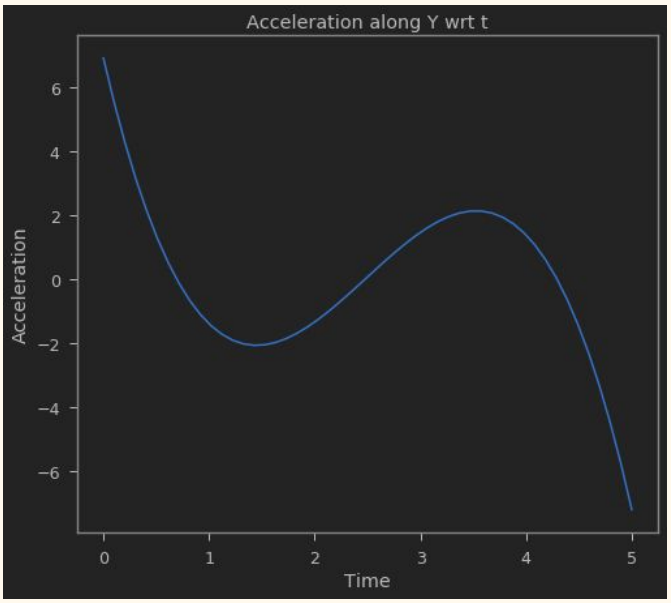


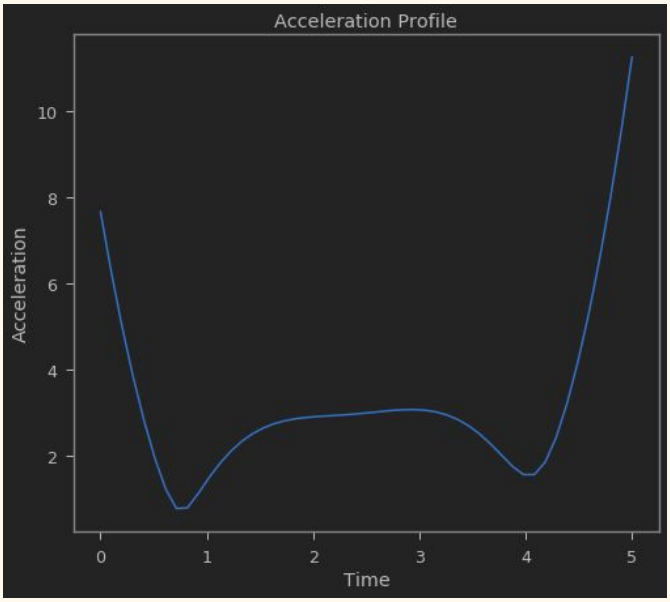




Acceleration Profile of X, Y, XY with respect to time







Part (c)

Given below is a description and implementation of the approach we used to arrive at a different trajectory to avoid the enemy robot. For this we aim to choose a path that would avoid collision with the enemy robot and also have minimum jerk.

- To achieve a trajectory that would avoid collision with the enemy robot we choose a metric (max.radius) as follows:

Code snippet:

```
enemy_radius = 2
robot_radius = 1
offset = 0.2
max_radius = enemy_radius + robot_radius + offset
# offset is margin by which we avoid the enemy robot
```

- We assume a quartic polynomial trajectory as done in part (a) with the same constraints and generate a trajectory which will be referred to as the reference trajectory hereafter.

Code snippet:

```
#generation of the reference trajectory
a =
np.array([[1,0,0,0,0],[0,1,0,0,0],[1,5,25,125,625],[0,1,10,75,50
0],[1,2,4,8,16]])
b = np.array([3,0,9,0,1])
x = np.linalg.solve(a, b)

a =
np.array([[1,0,0,0,0],[0,1,0,0,0],[1,5,25,125,625],[0,1,10,75,50
0],[1,2,4,8,16]])
b = np.array([0,0,5,0,2.5])
y = np.linalg.solve(a, b)
```

```
t = np.linspace(0, 5, num=50)
yt = y[0] + y[1]*(t) + y[2]*(t**2) + y[3]*(t**3) + y[4]*(t**4)
xt = x[0] + x[1]*(t) + x[2]*(t**2) + x[3]*(t**3) + x[4]*(t**4)
```

- Now the point on the calculated reference trajectory with the shortest perpendicular distance from the obstacle point (5, 4) is found from which the slope of the line joining the two points is calculated.

Code snippet:

```
def slopeFromPoints(P,Q):
    a = Q[1] - P[1]
    b = P[0] - Q[0]
    c = a*(P[0]) + b*(P[1])
    return -a/b
```

- Now to generate alternate trajectories so as to avoid the enemy robot, points are sampled along the line giving rise to all possible trajectories. The limits of the sampled points can be chosen in accordance with the value of max. radius(we had chosen a large margin of twice the max.radius generating 50 alternate trajectories).

Code snippet:

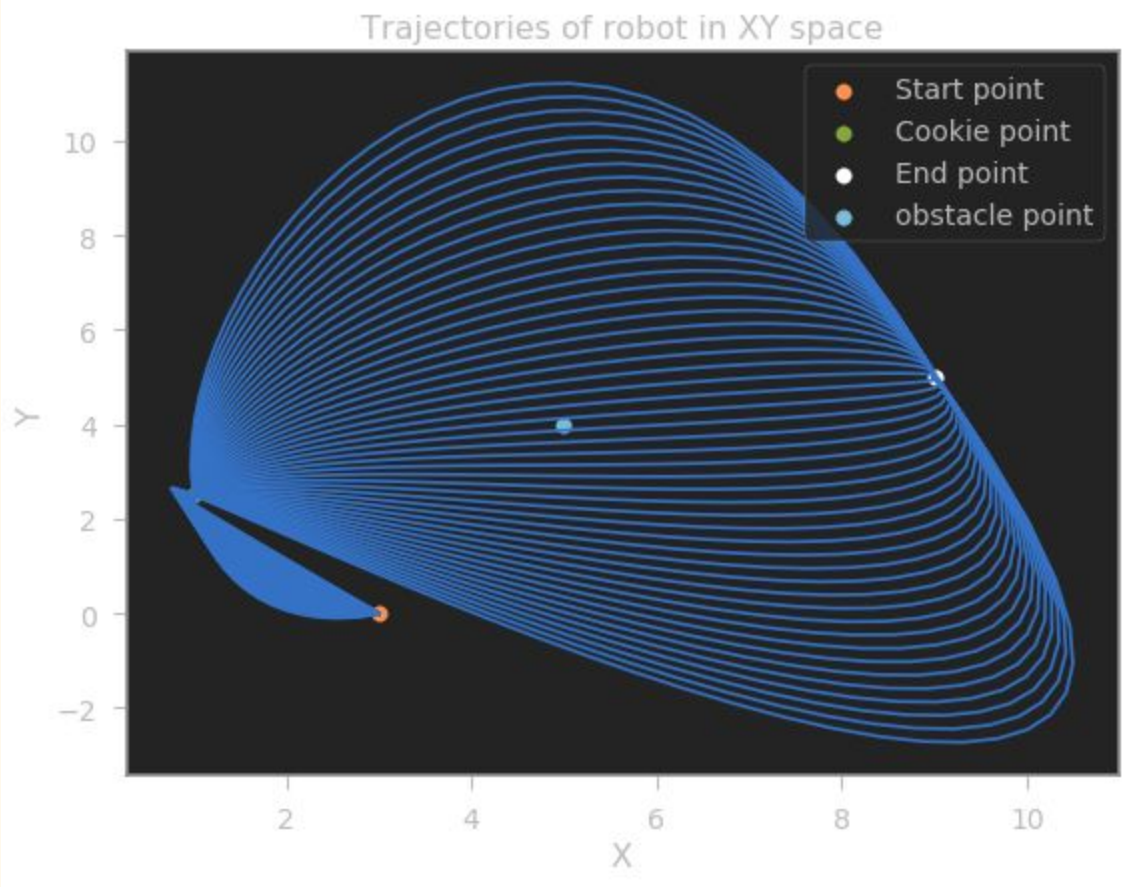
```
def get_points(slope, x,y, d):
    a = arctan(slope)
    x1 = x - d*cos(a)
    x2 = x + d*cos(a)
    y1 = y - d*sin(a)
    y2 = y + d*sin(a)
    return x1,y1,x2,y2
```

- We generate all possible trajectories using the quartic polynomial constraints.(Assumption: The robot crosses the obstacle point at time t = 3.5s).

Code snippet:

```
xt=[]
yt =[]
for x,y in zip(x_margin,y_margin):
    a =
np.array([[1,0,0,0,0],[0,1,0,0,0],[1,5,25,125,625],[1,2,4,8,16],
[1,3.5,12.25,42.875,150.0625]])
    b = np.array([3,0,9,1,x])
    xt.append(np.linalg.solve(a, b))
    a =
np.array([[1,0,0,0,0],[0,1,0,0,0],[1,5,25,125,625],[1,2,4,8,16],
[1,3.5,12.25,42.875,150.0625]])
    b = np.array([0,0,5,2.5,y])
    yt.append(np.linalg.solve(a, b))
xt = np.asarray(xt)
yt = np.asarray(yt)
```

Plot of all generated trajectories:



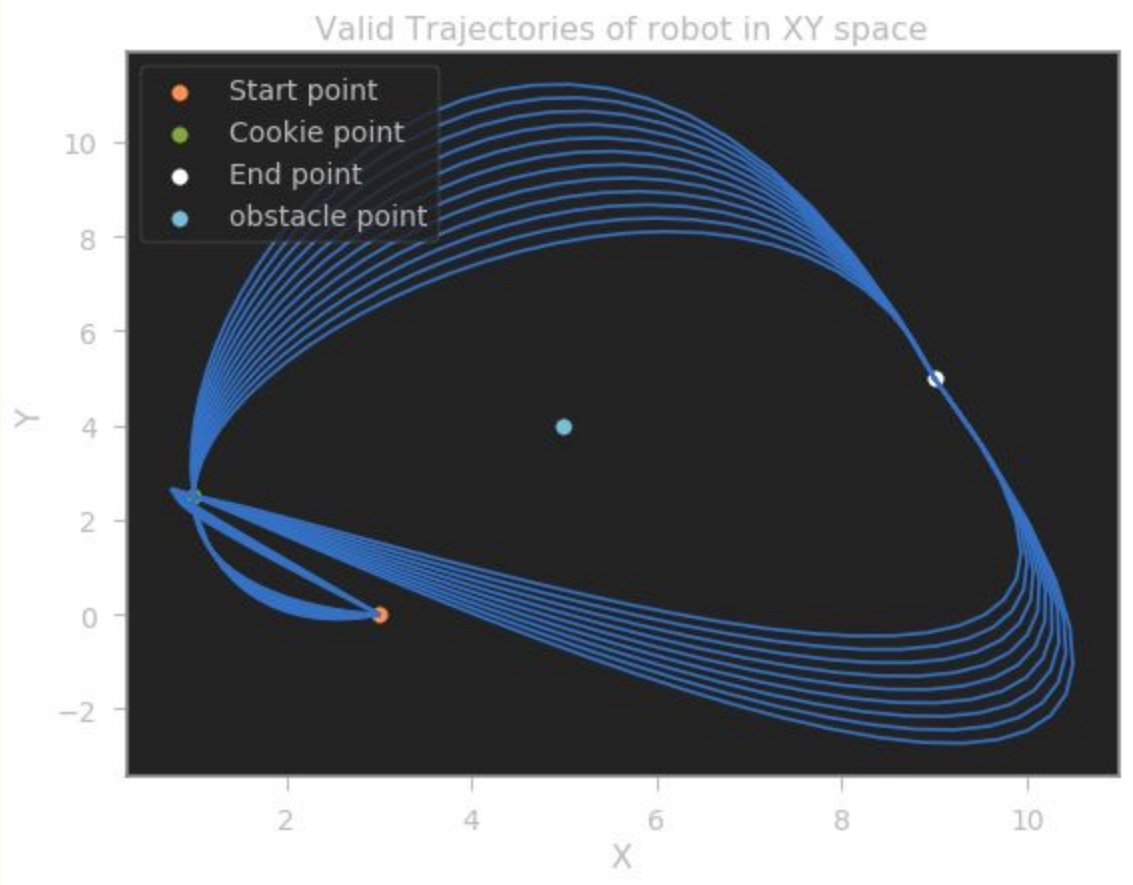
- Now, the task is to choose valid trajectories. For this we eliminate all those trajectories which occur in the collision zone (less than max.radius = 3.2 units).

Code snippet:

```
#Obtaining the valid trajectories i.e. the trajectories which
avoid collision
valid_traj = []
coeffx = []
coeffy = []
i = 0
for x,y in zip(X,Y):
```

```
    temp = list(zip(x,y))
    index = pairwise_distances_argmin([[5,4]], temp)
    if(np.hypot((temp[index[0]][0] - 5),(temp[index[0]][1] - 4))
> max_radius):
        valid_traj.append(temp)
        coeffx.append(xt[i])
        coeffy.append(yt[i])
        i += 1
valid_traj = np.asarray(valid_traj)
coeffx = np.asarray(coeffx)
coeffy = np.asarray(coeffy)
```

Plot of Valid trajectories among all generated trajectories:



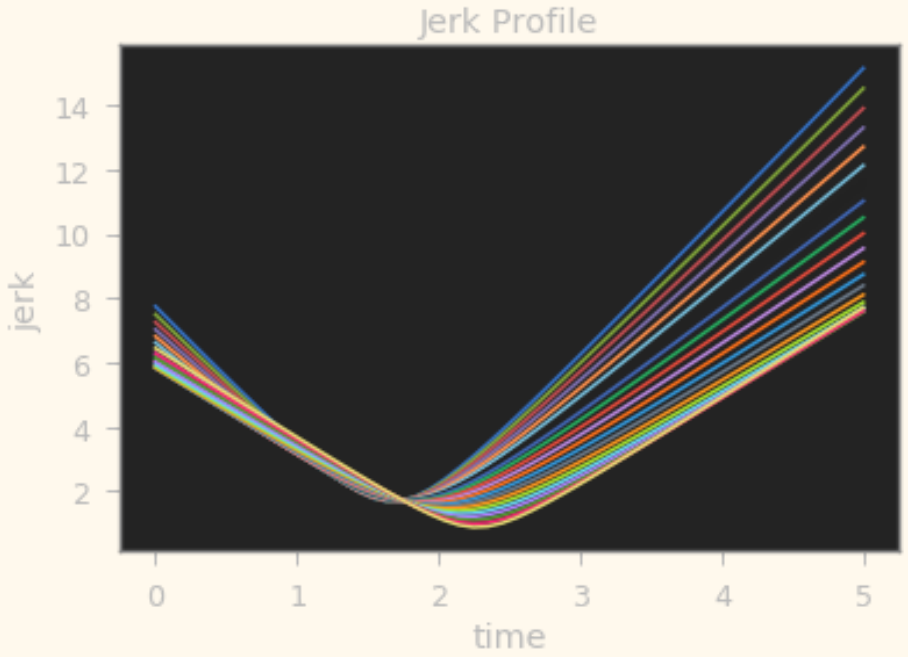
- From the obtained set of valid trajectories, the least jerkiest trajectory is chosen(minimum rate of change of acceleration).

Code snippet:

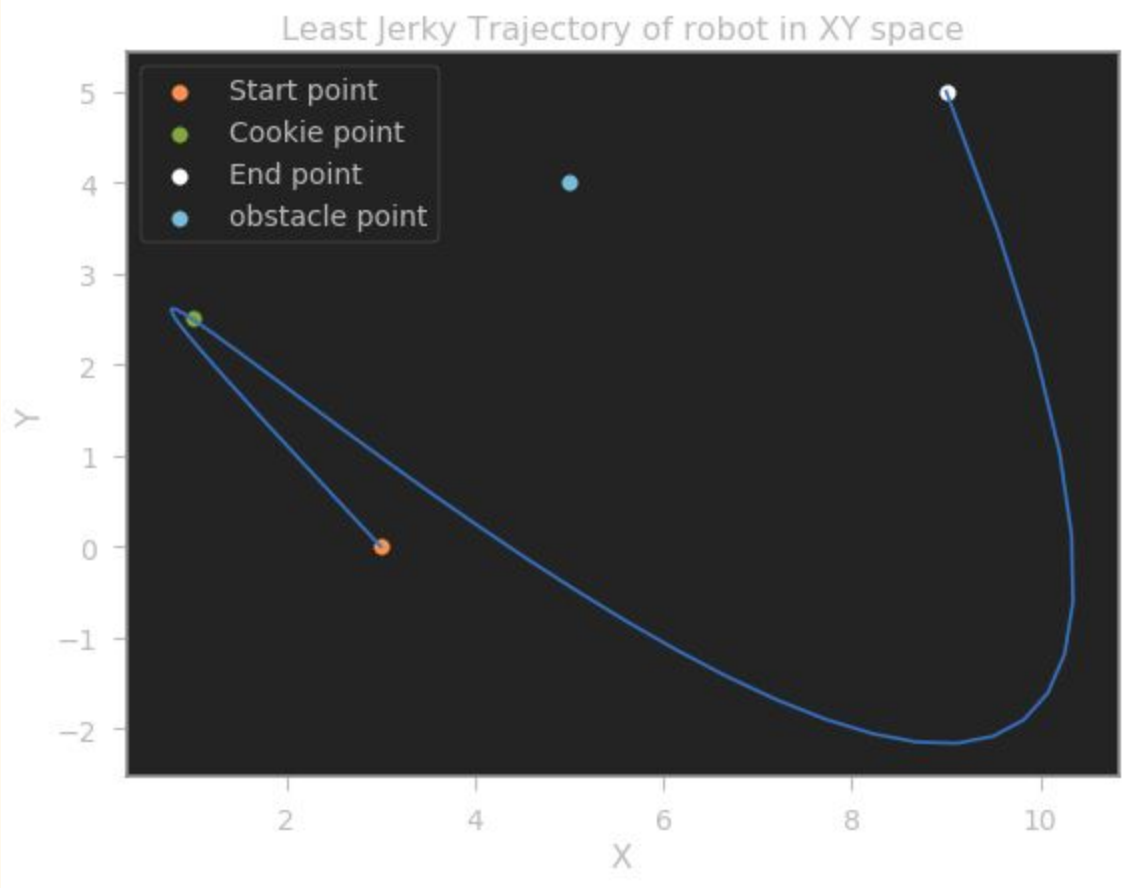
```
#choosing the least jerkiest trajectory
mean_jerk =[]

for x,y in zip(coeffx,coeffy):
    jx = 6*x[3] + 24*x[4]*(t**1)
    jy = 6*y[3] + 24*y[4]*(t**1)
    jerk = np.hypot(jx,jy)
    mean_jerk.append(np.mean(abs(jerk)))
plt.legend()
plt.xlabel("time")
plt.ylabel("jerk")
plt.title("Jerk Profile")
mean_jerk = np.asarray(mean_jerk)
index = np.where(mean_jerk == np.amin(mean_jerk))
```

Plot of Jerk Profile among all valid trajectories:



Plot of the optimal trajectory that avoids collision and has the least jerk:



References

[1] Prof. K. Madhava Krishna, Gourav Kumar, Enna Sachdeva, Nonholonomic Trajectory Planning (Bernstein Basis method)

Roles:

We have discussed together and implemented the functions there is no distinct role division.