ROB-UY 3303 Project 2

Seoyoon Kim / NetID: sk11361

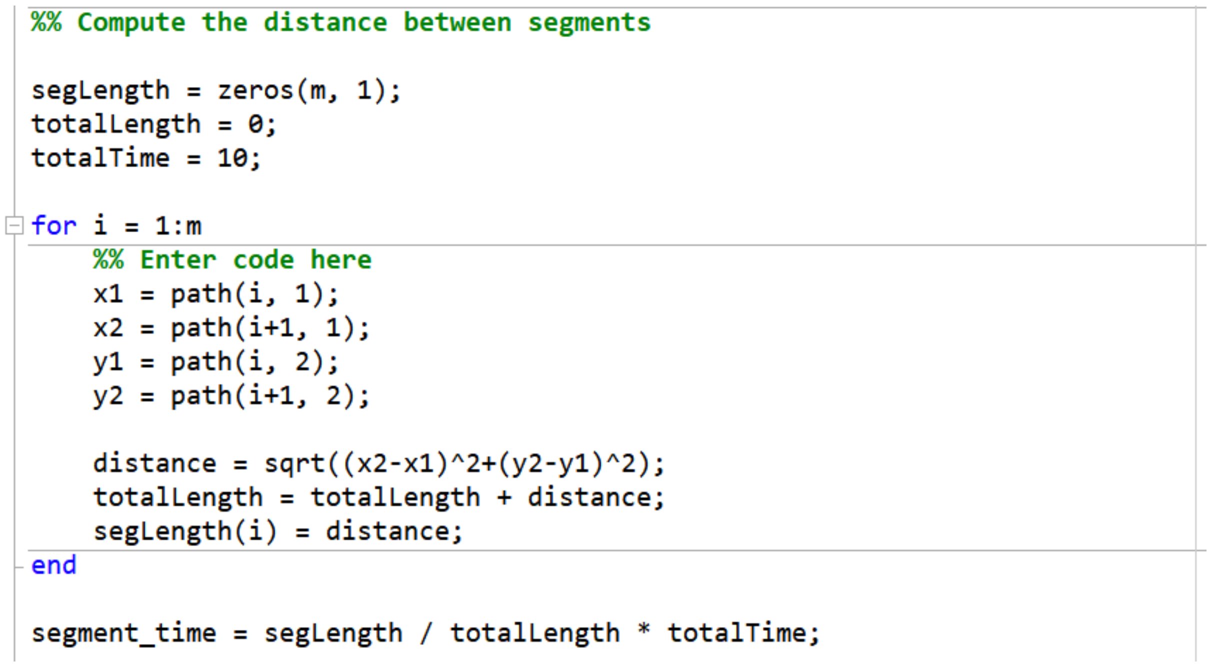
1. Choosing the Order of Polynomial

There are two assignment constraints at the two end points. There are three constraints at each intermediate waypoints, two assignment constraints and one continuity constraints. Therefore, total constraints are $4 + 3(N-1) = 3N + 1$. Since, $N-1$ segments of order M connect N points can impose $(M+1)(N-1)$ constraints, $M = 3$ is moderate for $N \geq 5$.

$$x = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$

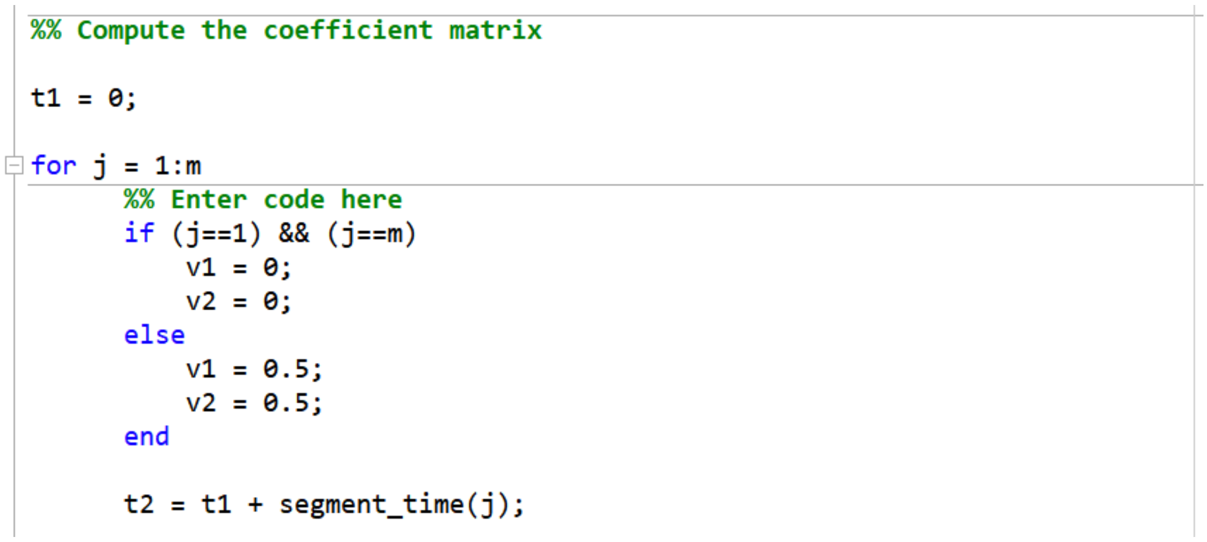
$$\dot{x} = a_1 + 2a_2 t + 3a_3 t^2$$

2. Codes Explanation



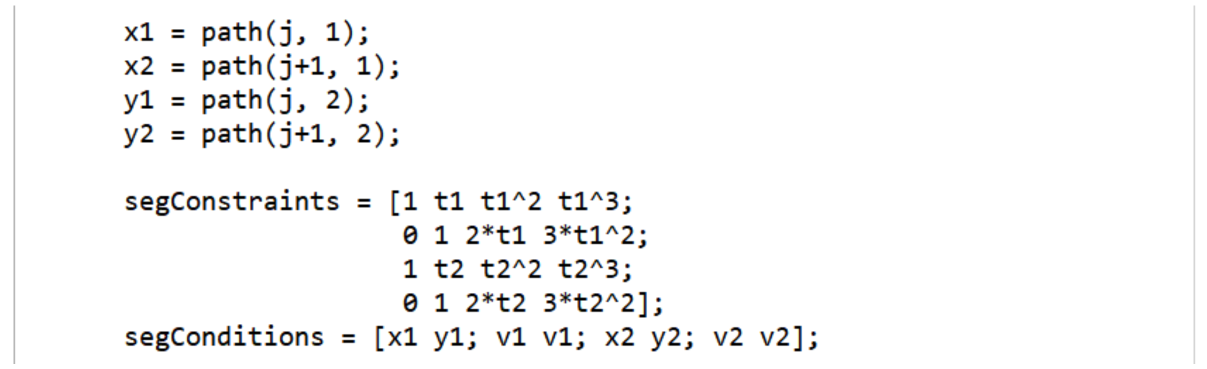
```
%% Compute the distance between segments

segLength = zeros(m, 1);
totalLength = 0;
totalTime = 10;

for i = 1:m
    %% Enter code here
    x1 = path(i, 1);
    x2 = path(i+1, 1);
    y1 = path(i, 2);
    y2 = path(i+1, 2);

    distance = sqrt((x2-x1)^2+(y2-y1)^2);
    totalLength = totalLength + distance;
    segLength(i) = distance;
end

segment_time = segLength / totalLength * totalTime;
```

x1, y1 are the start point of the segment, and x2, y2 are the end point of the segment. A for loop updates the segment every turn, and calculates the distance of the segment. After that, it adds it to the total length and stores the distance to segment length array. Since segment time should be allocated considering the length of the segment with respect to the total length, I divided segment length with total length, and then multiplied total time.



```
%% Compute the coefficient matrix

t1 = 0;

for j = 1:m
        %% Enter code here
        if (j==1) && (j==m)
            v1 = 0;
            v2 = 0;
        else
            v1 = 0.5;
            v2 = 0.5;
        end

        t2 = t1 + segment_time(j);
```

Since the velocity of the start point and the end point are 0, I wrote an if statement to distinguish with other intermediate cases. t1 is the time of the start point, and t2 is the time of the end point of the segment.

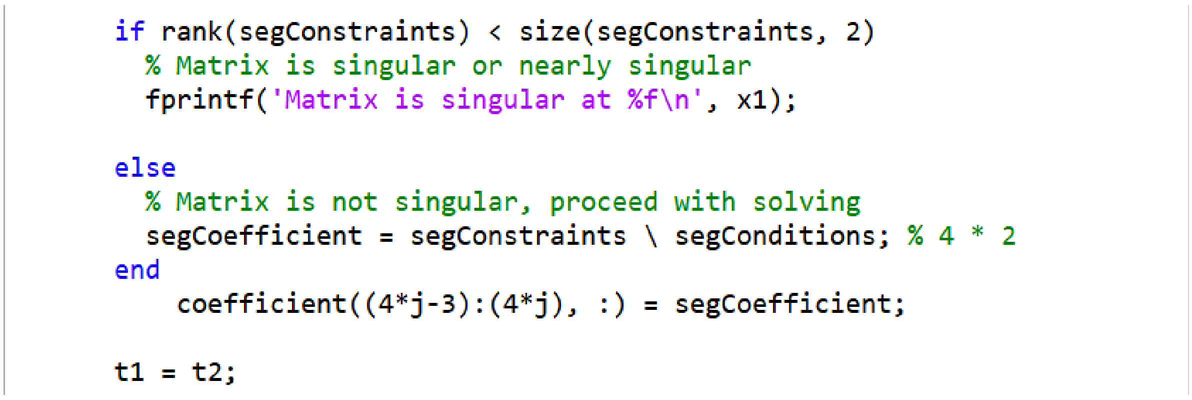


```
        x1 = path(j, 1);
        x2 = path(j+1, 1);
        y1 = path(j, 2);
        y2 = path(j+1, 2);

        segConstraints = [1 t1 t1^2 t1^3;
                          0 1 2*t1 3*t1^2;
                          1 t2 t2^2 t2^3;
                          0 1 2*t2 3*t2^2];
        segConditions = [x1 y1; v1 v1; x2 y2; v2 v2];
```

In order to obtain a trajectory of 2D model, I thought two separate polynomials that fits the given path and describe motion of each x and y coordinates are necessary.

$$\begin{bmatrix} 1 & t_1 & t_1^2 & t_1^3 \\ 0 & 1 & 2t_1 & 3t_1^2 \\ 1 & t_2 & t_2^2 & t_2^3 \\ 0 & 1 & 2t_2 & 3t_2^2 \end{bmatrix} \begin{bmatrix} a_{x,0} & a_{y,0} \\ a_{x,1} & a_{y,1} \\ a_{x,2} & a_{y,2} \\ a_{x,3} & a_{y,3} \end{bmatrix} = \begin{bmatrix} x_1 & y_1 \\ v_1 & v_1 \\ x_2 & y_2 \\ v_2 & v_2 \end{bmatrix}$$

Since I use the 3rd order polynomials, segment constraints become 4×4 matrix, and segment coefficients and conditions become 4×2 matrix.

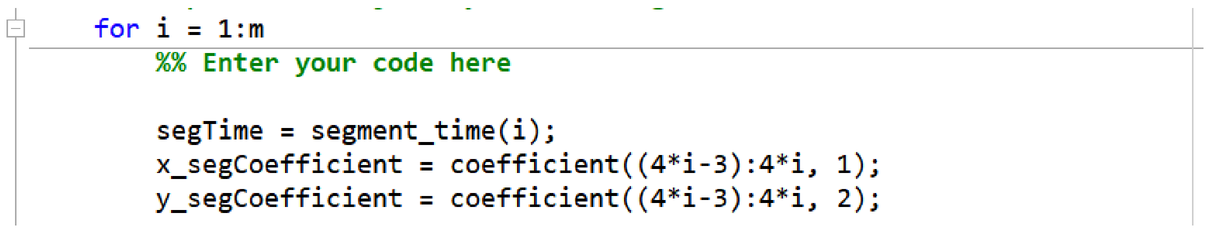


```
        if rank(segConstraints) < size(segConstraints, 2)
          % Matrix is singular or nearly singular
          fprintf('Matrix is singular at %f\n', x1);

        else
          % Matrix is not singular, proceed with solving
          segCoefficient = segConstraints \ segConditions; % 4 * 2
        end
            coefficient((4*j-3):(4*j), :) = segCoefficient;

        t1 = t2;
```

To obtain the segment coefficient from the matrix equation, I used below command.

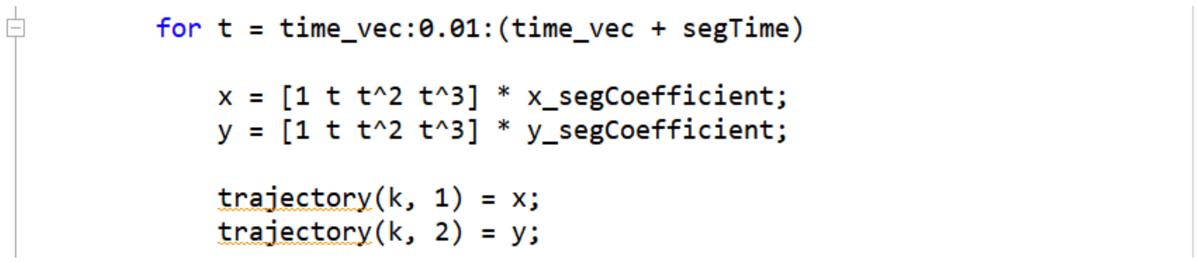
$$segCoefficient = segConstraints\backslash segConditions$$

I stored the result at the coefficient matrix.



```
    for i = 1:m
        %% Enter your code here

        segTime = segment_time(i);
        x_segCoefficient = coefficient((4*i-3):4*i, 1);
        y_segCoefficient = coefficient((4*i-3):4*i, 2);
```

Segment coefficient of x polynomial is stored in the 1st column of the coefficient matrix, and segment coefficient of y polynomial is stored in the 2nd column of the coefficient matrix. Each polynomial has 4 coefficients, so the range is from $4i - 3$ to $4i$.

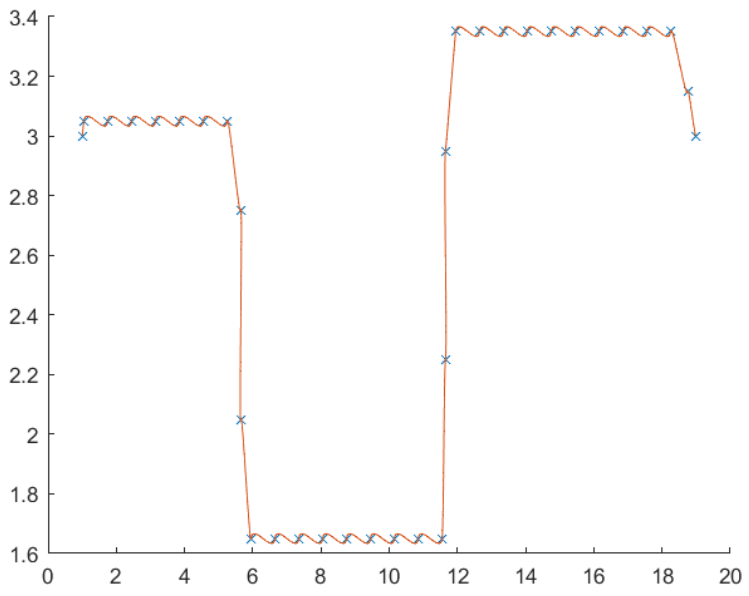


```
        for t = time_vec:0.01:(time_vec + segTime)

            x = [1 t t^2 t^3] * x_segCoefficient;
            y = [1 t t^2 t^3] * y_segCoefficient;

            trajectory(k, 1) = x;
            trajectory(k, 2) = y;
```

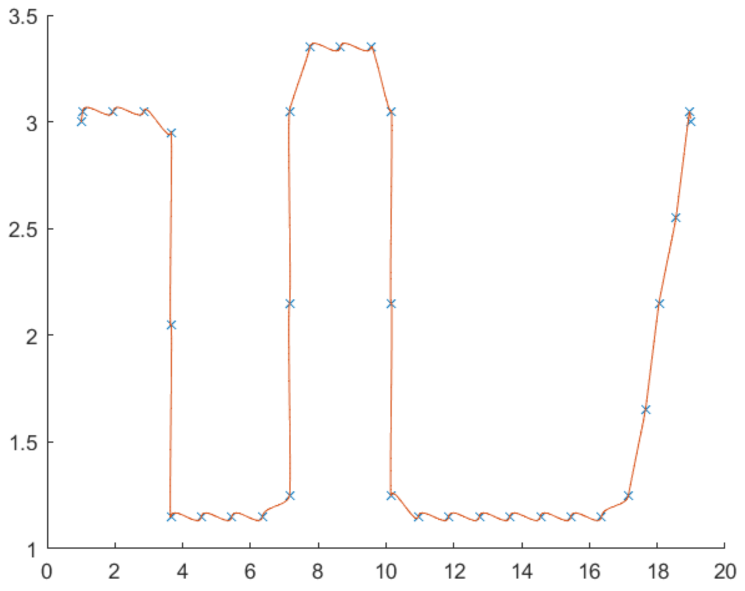
This code combines the two separate polynomial trajectories, and visualize in 2D. Time step is 0.01 second. It first calculates the x and y coordinates for every time step and stores the result in a trajectory matrix.

3. Result

This is the result for path 1. A trajectory passes through all the given points.



This is the result for path 2. A trajectory passes through all the given points.



3.5
3
2.5
2
1.5
1
0
2
4
6
8
10
12
14
16
18
20