Recitation 5: 2-Link Arm Maze Race

1. Simulation Results

We made it through the maze, **with a score of 18.20**, using our simulated model and trajectory parameters. The following rules were used to find the final scores:

$$Score = 100 * d_{traveled} * (d_{traveled} / (t_{traveled} + 1))$$

Scoring will be applied when any of the follow occur:

- The laser (i.e., end effector coordinate) illegally crosses a maze wall,
- the time since leaving the start line exceeds 10 seconds, or
- The laser successfully crosses the finish line.

The following scores were achieved for our model:

```
Made it through the maze! (Congratulations!)
dist = 50
tval = 4.4935
Normalized distance: 1.00
score: 18.20
```

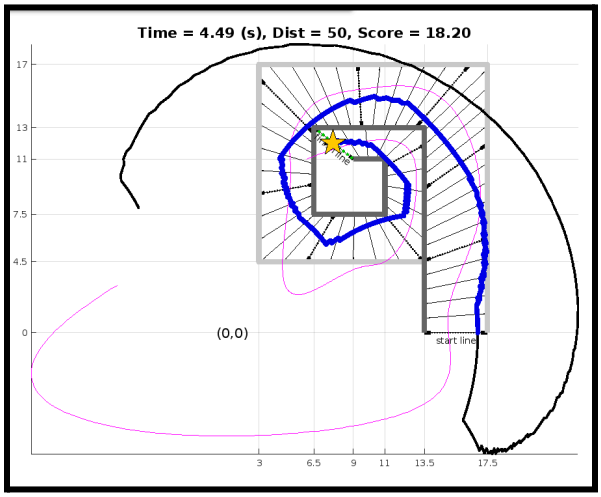

Fig 1 - The ideal and actual trajectory of the end effector. We are finishing the maze successfully. Look at the score values at top.

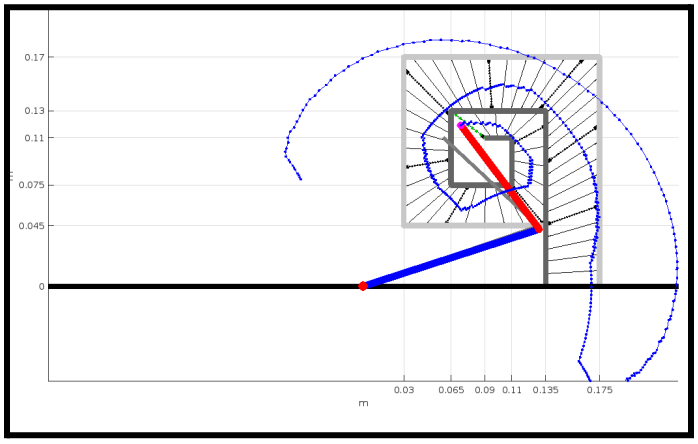

Fig 2 - Animated view of how the robot actually traces the actual trajectory, starting from the initial offset position.

2. Trajectory path

The path was brute forced based on the controller performance and the reaction time constraints. The following path was used, starting from x_offset and y_offset points as given in the problem statement.

```
X = [-6 -6  12.5 15 17  15   10   6.5  5  5    4.5 6   9 11  12.5 13  12 6];
Y = [3  -6  -5   1  7.5 14.5 15.5 14.5 12 9.5  4   2.5 4 4.5 6.5  9.5 12 11];
```

These values were smoothened using the cubic "`spline`" function. The path is clearly visible in the following image.

The green line is the smooth path. Do note that we explicitly want our "ideal" trajectory to cross the boundaries, this is to prevent the controller from bumping into the upper wall while turning.

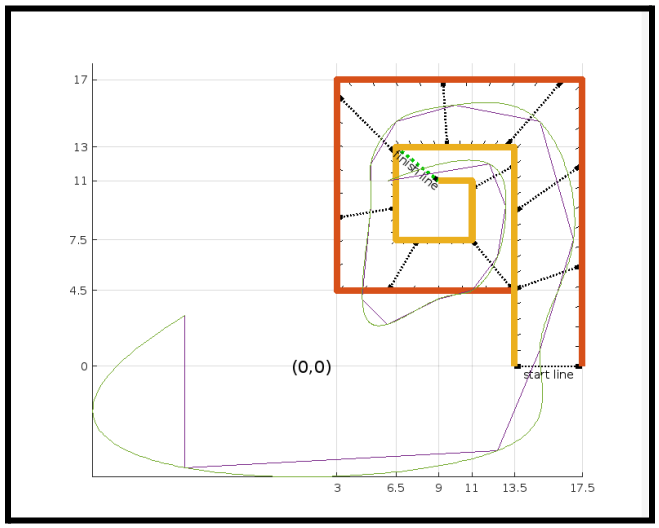

Fig 3 - Sharp and smooth ideal trajectory before starting the simulation.

3. Parameters used

$$dt = 0.005\ s$$
$$[L1,\ L2] = [13.7,\ 9.65]\ cm$$
$$[offset1,\ offset2] = [169^0,\ -147^0]$$
$$Kp2 = Kp1 = 11.4196;\ \%\ Kp\ for\ PD\ controller$$
$$Kd2 = Kd1 = 1.6169;\ \%\ Kd\ for\ PD\ controller$$
$$f = 0.75;\ \%\ value\ for\ "velocity\ filter"$$
$$A = (1 - f)/0.005;\ \%\ value\ for\ "velocity\ filter"$$

Appendix - Code

rec5_main.m

```C/C++
close all; clear all;
%% Tuned Parameters
L1 = 13.7;
L2 = 9.65;
Kp1 = 11.4196; % Kp for PD controller (for "motor 1")
Kd1 = 1.6169; % Kd for PD controller (for "motor 1")
Kp2 = Kp1; % Kp for PD controller (for "motor 2")
Kd2 = Kd1; % for PD controller (for "motor 2")
f = 0.75; % value for "velocity filter"
A = (1-f)/0.005; % value for "velocity filter"
offset1 = 169*(pi/180);
offset2 = -147*(pi/180);
[x_off, y_off] = FK(offset1,offset2, L1, L2);
x_init = 6;
y_init = 6;
x_sharp = [-6 -6  12.5 15 17  15   10   6.5  5  5    4.5 6   9 11  12.5
13  12 6];
y_sharp = [3  -6  -5   1  7.5 14.5 15.5 14.5 12 9.5  4   2.5 4 4.5 6.5
9.5 12 11];
% Define the number of smooth waypoints
n = 1000;
% Create a finer set of x-values (between 1 and the number of sharp
points)
x_fine = linspace(1, length(x_sharp), n);
% Interpolate the x and y values using cubic spline
sim_x = interp1(1:length(x_sharp), x_sharp, x_fine, 'spline');
sim_y = interp1(1:length(y_sharp), y_sharp, x_fine, 'spline');
%% Final set of values for time and trajectory
t_ref = zeros(1,length(sim_x));
for i = 2:length(t_ref)
 t_ref(i) = t_ref(i-1) + 0.005;
end
theta1_ref = zeros(1,length(t_ref));
theta2_ref = zeros(1,length(t_ref));
for i = 1:length(t_ref)
 [theta1_ref(i), theta2_ref(i)] = IK(sim_x(1,i),sim_y(1,i), L1, L2);
```

```
end
theta1_ref = real(theta1_ref);
theta2_ref = real(theta2_ref);
sim('Rec5_TwoLinkArmWithBacklash');
run('Automated_Scoring.m');
```

IK.m

```
C/C++
function [theta1,theta2] = IK(xee,yee, L1, L2)
% IK = "inverse kinematic", from end effector (x,y) to required
%      joint angles (theta1, theta2)
% Inputs: desired xee and yee of desired end effector (in METERS)
%         L1 and L2 of the two-link arm (in meters)
% Outputs: theta1 and theta2 (motor angles, in RADIANS)
% Lego two-link robot arm uses these link lengths:
% L1 = 0.137 ;               % length of link 1
% L2 = 0.0965 ;              % length of link 2
%% Below, YOU MUST CHANGE THE NEXT TWO LINES, to output the actual IK
r = (xee^2 + yee^2)^0.5;
theta2  = acos( (r^2 - L1^2 - L2^2) / (2*L1*L2) );
% theta1 = atan(yee/xee) - atan((L2*sin(theta2))/(L1+L2*cos(theta2))); %
for positive theta 1
theta1 = atan2(yee,xee) - acos((L2^2 - L1^2 - r^2)/-(2* L1 * r));      %
for any theta 1
```

FK.m

```
C/C++
function [xee,yee] = FK(theta1,theta2,L1, L2)
% FK = "forward kinematics"
% Inputs: theta1, theta2 are motor angles (in RADIANS)
%         L1 and L2 are link lengths (in METERS)
% Outputs: (xee,yee) give coordinates of END EFFECTOR (in METERS)
xee = L1 * cos (theta1) + L2 * cos (theta1+theta2);
yee = L1 * sin (theta1) + L2 * sin (theta1+theta2);
```