ASSIGNMENT 3

NOVEMBER 12, 2021

Submitted to: Prof. Dr. Kemalettin Erbatur

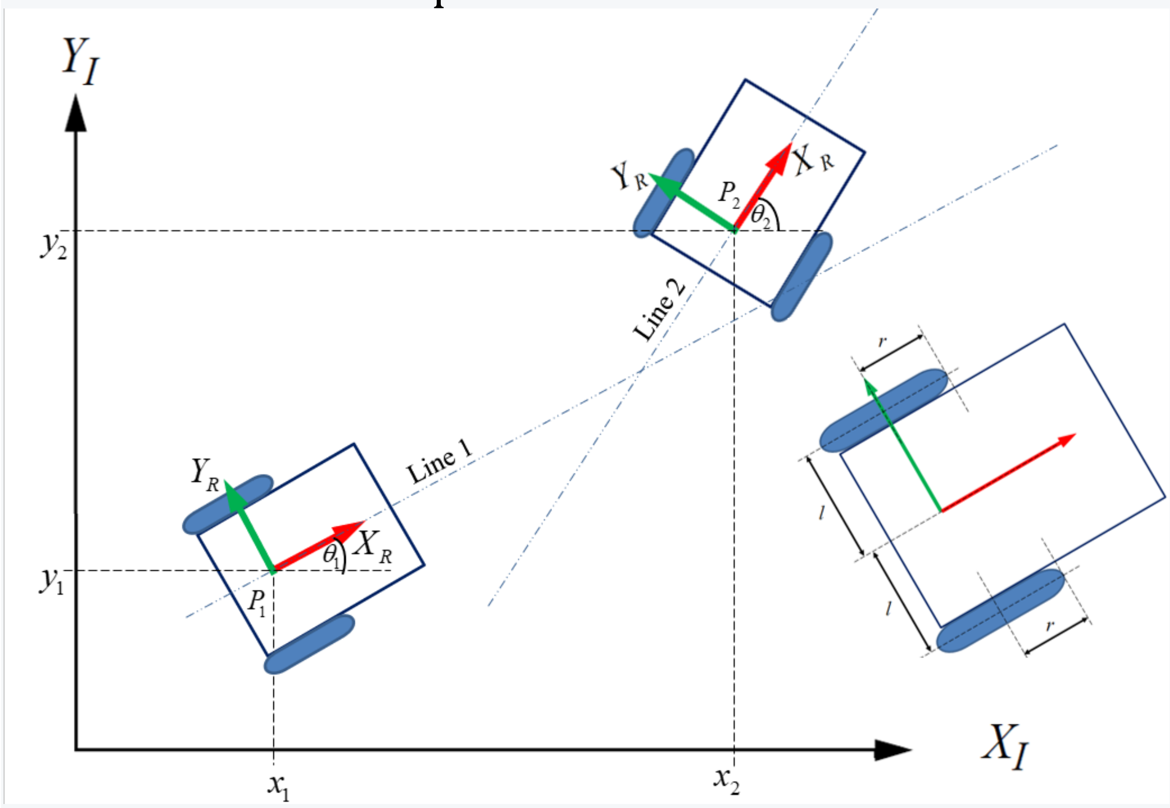
Name of Student: Moses Chuka Ebere
Student Number: 31491
Course: Autonomous Mobile Robotics (ME 525)

TABLE OF CONTENTS

INTRODUCTION

This is a computational assignment to determine the trajectory of a wheel robot.

According to the question, the robot is expected to move from P1 along line 1 up till the intersection point and turn at this point to align with line 2. The final pose will be at P2.



From my understanding, due to the constraint that the turn must occur at the intersection point, there'll be no arc. So, I solved the question using this. However, I believe that this trajectory is kind of unrealistic given that a robot on two wheels is expected to make a turn on a spot. It seems impossible for the robot to turn without the wheels sliding.

Just in case (to be on the safe side), I attached another MATLAB code in the appendix where I assumed an arc radius of 1.2m (as obtained from v = wr).

MATLAB CODE

```
close all; clear;

% DATA ENTRY
%----------------------------------------
% Simulation Parameters - properly outline all the given
parameters.
step_time = 0.001; % We want to have data for every
millisecond.

% Robot Dimensions
r = 0.1; % radius of the wheel in meters.
l = 0.3; % Half of the width of the robot in meters.

% Initial Pose - [x1, y1, theta1]'
x1 = 0.6; % in meters.
y1 = 0.8; % in meters.
theta1 = 0.15; % Angles are always in radians ~ 10 degrees.
P1 = [x1; y1]; % A column vector

% Final Pose - [x2, y2, theta2]'
x2 = 3; % in meters.
y2 = 2; % in meters.
theta2 = 1.2; % in radians ~ 70 degrees
P2 = [x2; y2]; % A column vector

% Speed Parameters
VT = 0.12; % Tangential VELOCITY in m/s - it has x and y
components which
% will serve as X_dot (= VT*cos(...)) and Y_dot (=
VT*sin(...))
arc_omega = 0.1; % in radians/second.
%----------------------------------------
% End of Data Entry

% COMPUTATION OF POINTS
%----------------------------------------
% Computation of the line-to-line intersection point
data_matrix = [cos(theta1) -cos(theta2); sin(theta1) -
sin(theta2)];
lambda_P6 = (data_matrix)\(P2 - P1);
lambda1_P6 = lambda_P6(1);
```

```
P6 = P1 + lambda1_P6*[cos(theta1); sin(theta1)];

% Rotational Motion at intersection
delta_theta = theta2 - theta1;

% Plot the lines and points P1 to P6
figure %1
plot(P1(1), P1(2), 'ro')
hold
plot(P6(1), P6(2), 'go')
plot(P2(1), P2(2), 'mo')
plot([P1(1) P6(1)], [P1(2) P6(2)], 'k')
plot([P6(1) P2(1)], [P6(2) P2(2)], 'k')
axis([0 x2+1 0 y2+1])
xlabel('x [m]')
ylabel('y [m]')
grid
%----------------------------------------
% End of Computation of Points

% COMPUTATION OF MOTION SEGMENT DURATIONS
%----------------------------------------
% Distance Travelled
s_line1 = lambda1_P6;
rotational_distance = delta_theta;
lambda2_P6 = lambda_P6(2);
% lambda2_P6 is negative, and distance can't be negative;
so, we take its
% absolute value.
s_line2 = abs(lambda2_P6);

% Durations
duration_line1 = s_line1/VT;
duration_intersection = rotational_distance/arc_omega;
duration_line2 = s_line2/VT;
duration_total = duration_line1 + duration_intersection +
duration_line2;

% Transition Times
t1 = 0;
t2 = duration_line1;
t3 = t2 + duration_intersection;
```

```
t4 = duration_total;
%---------------------------------------
% End of Computation of Motion Segments Durations.

% GEOMETRIC CONSTRUCTION OF THE CARTESIAN REFERENCE CURVE
%---------------------------------------
% Data Storage Dimension - The +1 is for time = 0s.
array_length = ceil(duration_total/step_time) + 1;

% X, Y, and Theta Trajectory Computations.
x_list = zeros(array_length,1);
y_list = zeros(array_length,1);
theta_list = zeros(array_length,1);
time_list = zeros(array_length,1);

for iteration_index = 1:1:array_length
    % Convert the step time to instantaneous times
    time = (iteration_index - 1)* step_time;
    % Fill in the time list
    time_list(iteration_index) = time;

    % Find the pose at every millisecond within each
segment
    % Use P = Pi + lambda_i*[cos(theta1_2); sin(theta1_2)]
for the lines
    % Where lambda = (ins_time - last_cumulative_time)*VT
    % For the arc, use P6

    if (t1 <= time) && (time < t2)
        P = P1 + time*VT*[cos(theta1); sin(theta1)];
        x = P(1);
        y = P(2);
        theta = theta1;
    end

    if (t2 <= time) && (time < t3)
        x = P6(1);
        y = P6(2);
        theta = theta1 + (time - t2)*arc_omega;
    end

    if (t3 <= time) && (time <= t4)
```

```
        P = P6 + (time - t3)*VT*[cos(theta2); sin(theta2)];
        x = P(1);
        y = P(2);
        theta = theta2;
    end

    x_list(iteration_index) = x;
    y_list(iteration_index) = y;
    theta_list(iteration_index) = theta;
end

figure %2
plot(x_list, y_list)
hold
plot(P1(1), P1(2), 'ro')
plot(P6(1), P6(2), 'go')
plot(P2(1), P2(2), 'mo')
axis([0 x2+1 0 y2+1])
xlabel('x [m]')
ylabel('y [m]')
grid

figure %3
subplot(3,1,1)
plot(time_list, x_list)
xlabel('Time [s]')
ylabel('x [m]')
subplot(3,1,2)
plot(time_list, y_list)
xlabel('Time [s]')
ylabel('y [m]')
subplot(3,1,3)
plot(time_list, theta_list)
xlabel('Time [s]')
ylabel('theta [rad]')
%----------------------------------------
% End of the Geometric Construction of the Reference
Cartesian Curves

% GEOMETRIC CONSTRUCTION OF THE REFERENCE CARTESIAN SPEEDS
%----------------------------------------
x_dot_list = zeros(array_length,1);
```

```
y_dot_list = zeros(array_length,1);
theta_dot_list = zeros(array_length,1);
time_list = zeros(array_length,1);

for iteration_index = 1:1:array_length
    % Convert the step time to instantaneous times
    time = (iteration_index - 1)* step_time;
    % Fill in the time list
    time_list(iteration_index) = time;

    % Find the linear/angular velocities at every ms within
each segment
    if (t1 <= time) && (time < t2)
        x_dot = VT*cos(theta1);
        y_dot = VT*sin(theta1);
        theta_dot = 0;
    end

    if (t2 <= time) && (time < t3)
        theta = theta1 + (time - t2)*arc_omega;
        x_dot = 0;
        y_dot = 0;
        theta_dot = arc_omega;
    end

    if (t3 <= time) && (time <= t4)
        x_dot = VT*cos(theta2);
        y_dot = VT*sin(theta2);
        theta_dot = 0;
    end

    x_dot_list(iteration_index) = x_dot;
    y_dot_list(iteration_index) = y_dot;
    theta_dot_list(iteration_index) = theta_dot;
end

figure %4
subplot(3,1,1)
plot(time_list, x_dot_list)
xlabel('Time [s]')
ylabel('x dot [m/s]')
subplot(3,1,2)
```

```
plot(time_list, y_dot_list)
xlabel('Time [s]')
ylabel('y dot [m/s]')
subplot(3,1,3)
plot(time_list, theta_dot_list)
xlabel('Time [s]')
ylabel('theta dot [rad/s]')
%---------------------------------------
% End of the Geometric Construction of the Reference
Cartesian Speeds

% INVERSE KINEMATICS FOR WHEEL SPEEDS
%---------------------------------------
phi1_dot_list = zeros(array_length,1);
phi2_dot_list = zeros(array_length,1);

for iteration_index = 1:1:array_length

    phi1_dot = (cos(theta_list(iteration_index))...
        *x_dot_list(iteration_index))...
        /r...
        + (sin(theta_list(iteration_index))...
        *y_dot_list(iteration_index))...
        /r...
        + l...
        *theta_dot_list(iteration_index)...
        /r;
     phi2_dot = (cos(theta_list(iteration_index))...
        *x_dot_list(iteration_index))...
        /r...
        + (sin(theta_list(iteration_index))...
        *y_dot_list(iteration_index))...
        /r...
        - l...
        *theta_dot_list(iteration_index)...
        /r;

    phi1_dot_list(iteration_index) = phi1_dot;
    phi2_dot_list(iteration_index) = phi2_dot;
end

figure %5
```

```
subplot(2,1,1)
plot(time_list, phi1_dot_list)
xlabel('Time [s]')
ylabel('Phi1 dot [rad/s]')
subplot(2,1,2)
plot(time_list, phi2_dot_list)
xlabel('Time [s]')
ylabel('Phi2 dot [rad/s]')
%---------------------------------------
% End of Inverse Kinematic for Wheel Speeds

% FORWARD KINEMATICS CROSSCHECK FOR CARTESIAN SPEEDS
%---------------------------------------
x_dot_forward_list = zeros(array_length,1);
y_dot_forward_list = zeros(array_length,1);
theta_dot_forward_list = zeros(array_length,1);

current_theta = theta1; % Use this to avoid using if
statements.

for iteration_index = 1:1:array_length

    pose_forward...
            = [cos(current_theta) -sin(current_theta) 0;
               sin(current_theta)  cos(current_theta) 0;
               0             0             1]...
               *[r*phi1_dot_list(iteration_index)/2 +
r*phi2_dot_list(iteration_index)/2;
                 0;
                 r*phi1_dot_list(iteration_index)/(2*l) -
r*phi2_dot_list(iteration_index)/(2*l)];

    x_dot_forward = pose_forward(1);
    y_dot_forward = pose_forward(2);
    theta_dot_forward = pose_forward(3);

    x_dot_forward_list(iteration_index) = x_dot_forward;
    y_dot_forward_list(iteration_index) = y_dot_forward;
    theta_dot_forward_list(iteration_index) =
theta_dot_forward;
```

```
    current_theta = current_theta +
step_time*theta_dot_forward; %compact
end

figure %6
subplot(3,1,1)
plot(time_list, x_dot_forward_list)
xlabel('Time [s]')
ylabel('x dot [m/s]')
subplot(3,1,2)
plot(time_list, y_dot_forward_list)
xlabel('Time [s]')
ylabel('y dot [m/s]')
subplot(3,1,3)
plot(time_list, theta_dot_forward_list)
xlabel('Time [s]')
ylabel('Theta dot [rad/s]')
%---------------------------------------
% End of Forward Kinematic Crosscheck for Cartesian Speeds

% CARTESIAN SPEED INTEGRATION CROSSCHECK FOR CARTESIAN
POSITION
%---------------------------------------
x_forward_list = zeros(array_length,1);
y_forward_list = zeros(array_length,1);
theta_forward_list = zeros(array_length,1);
time1_list = zeros(array_length,1);

current_x = x1; current_y = y1; current_theta = theta1;

for iteration_index = 1:1:array_length
    % Convert the step time to instantaneous times
    time1 = (iteration_index - 1)* step_time;
    % Fill in the time list
    time1_list(iteration_index) = time1;

    x_forward = current_x;
    y_forward = current_y;
    theta_forward = current_theta;

    x_forward_list(iteration_index) = x_forward;
    y_forward_list(iteration_index) = y_forward;
```

```
    theta_forward_list(iteration_index) = theta_forward;

    current_x = current_x +
step_time*x_dot_forward_list(iteration_index);
    current_y = current_y +
step_time*y_dot_forward_list(iteration_index);
    current_theta = current_theta +
step_time*theta_dot_forward_list(iteration_index);
end

figure %7
plot(x_forward_list, y_forward_list)
hold
plot(P1(1), P1(2), 'ro')
plot(P6(1), P6(2), 'go')
plot(P2(1), P2(2), 'mo')
axis([0 x2+1 0 y2+1])
xlabel('x [m]')
ylabel('y [m]')
grid

figure %8
plot(x_forward_list, y_forward_list, 'k')
hold
plot(x_list, y_list, 'r')
plot(P1(1), P1(2), 'ro')
plot(P6(1), P6(2), 'go')
plot(P2(1), P2(2), 'mo')
axis([0 x2+1 0 y2+1])
xlabel('x [m]')
ylabel('y [m]')
grid
%----------------------------------------
% End of Cartesian Speed Integration Crosscheck for
Cartesian Position

% INTRODUCTION OF A MODEL PARAMETER MISMATCH ON FORWARD
SPEED KINEMATICS
%----------------------------------------
r2 = r*0.95;
x_dot_forward_m_list = zeros(array_length,1);
y_dot_forward_m_list = zeros(array_length,1);
```

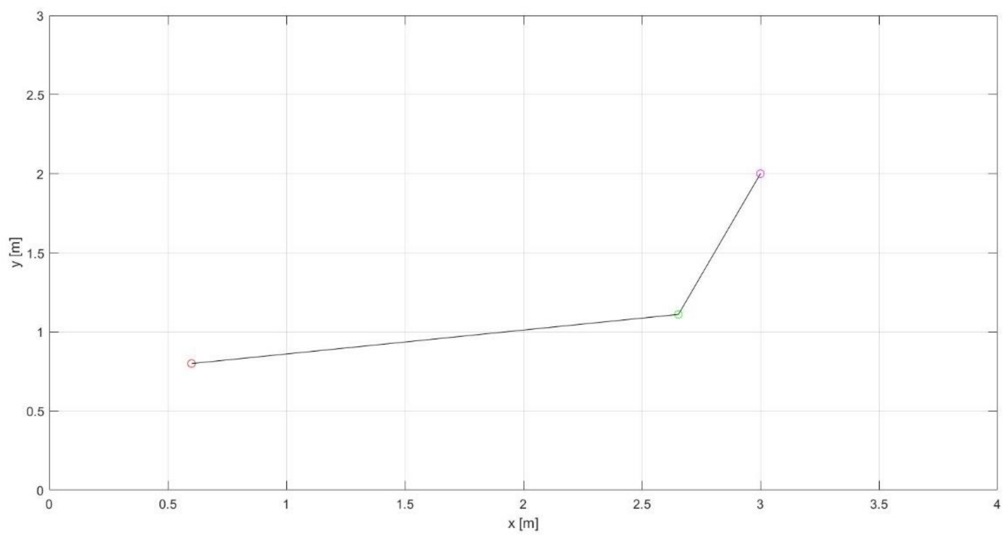
```
theta_dot_forward_m_list = zeros(array_length,1);

current_theta_m = theta1;

for iteration_index = 1:1:array_length

    pose_forward_mismatch...
            = [cos(current_theta_m) -sin(current_theta_m)
0;
               sin(current_theta_m)  cos(current_theta_m)
0;
               0                     0
1]...
               *[r*phi1_dot_list(iteration_index)/2 +
r2*phi2_dot_list(iteration_index)/2;
                 0;
                 r*phi1_dot_list(iteration_index)/(2*l) -
r2*phi2_dot_list(iteration_index)/(2*l)];

    x_dot_forward_m = pose_forward_mismatch(1);
    y_dot_forward_m = pose_forward_mismatch(2);
    theta_dot_forward_m = pose_forward_mismatch(3);

    x_dot_forward_m_list(iteration_index) =
x_dot_forward_m;
    y_dot_forward_m_list(iteration_index) =
y_dot_forward_m;
    theta_dot_forward_m_list(iteration_index) =
theta_dot_forward_m;

    current_theta_m = current_theta_m +
step_time*theta_dot_forward_m;
end

figure %9
subplot(3,1,1)
plot(time_list, x_dot_forward_m_list)
xlabel('Time [s]')
ylabel('x dot [m/s]')
subplot(3,1,2)
plot(time_list, y_dot_forward_m_list)
xlabel('Time [s]')
```

```
ylabel('y dot [m/s]')
subplot(3,1,3)
plot(time_list, theta_dot_forward_m_list)
xlabel('Time [s]')
ylabel('Theta dot [rad/s]')
%----------------------------------------
% End of Introduction of A Model Parameter Mismatch on
Forward Speed Kinematics

% INTEGRATION OF FORWARD KINEMATICS CARTESIAN SPEEDS WITH
PARAMETER MISMATCH
%----------------------------------------
x_forward_m_list = zeros(array_length,1);
y_forward_m_list = zeros(array_length,1);
theta_forward_m_list = zeros(array_length,1);

current_x_m = x1; current_y_m = y1; current_theta_m =
theta1;

for iteration_index = 1:1:array_length
    x_forward_m = current_x_m;
    y_forward_m = current_y_m;
    theta_forward_m = current_theta_m;

    x_forward_m_list(iteration_index) = x_forward_m;
    y_forward_m_list(iteration_index) = y_forward_m;
    theta_forward_m_list(iteration_index) =
theta_forward_m;

    current_x_m = current_x_m +
step_time*x_dot_forward_m_list(iteration_index);
    current_y_m = current_y_m +
step_time*y_dot_forward_m_list(iteration_index);
    current_theta_m = current_theta_m +
step_time*theta_dot_forward_m_list(iteration_index);
end

figure %10
plot(x_forward_m_list, y_forward_m_list)
hold
plot(P1(1), P1(2), 'ro')
plot(P6(1), P6(2), 'go')
```

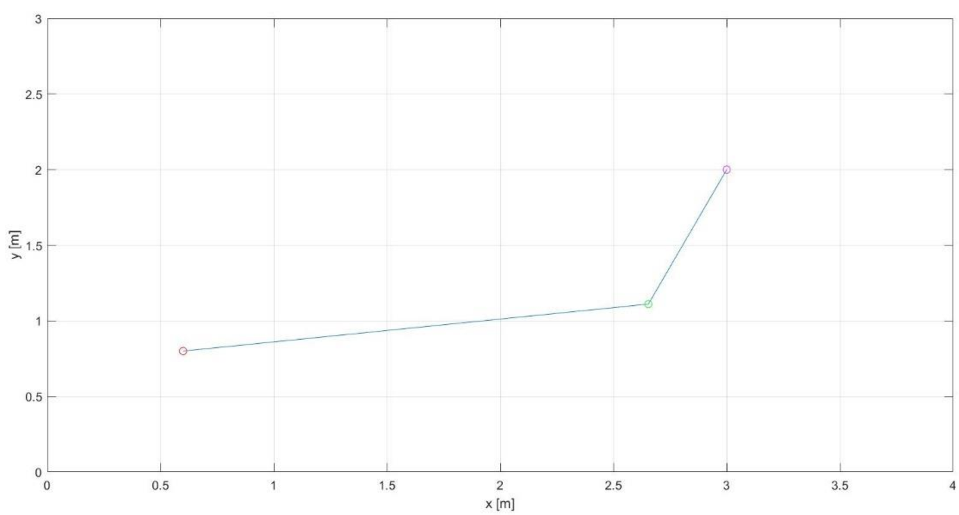
```
plot(P2(1), P2(2), 'mo')
axis([0 x2+1 0 y2+1])
xlabel('x [m]')
ylabel('y [m]')
grid

figure %11
plot(x_forward_m_list, y_forward_m_list, 'k')
hold
plot(x_list, y_list, 'r')
plot(P1(1), P1(2), 'ro')
plot(P6(1), P6(2), 'go')
plot(P2(1), P2(2), 'mo')
axis([0 x2+1 0 y2+1])
xlabel('x [m]')
ylabel('y [m]')
grid
%----------------------------------------
% End of Integration of Forward Kinematics Cartesian Speeds
with Parameter Mismatch

figure %12
subplot(3,1,1)
plot(time_list, x_forward_m_list, 'k', time_list, x_list,
'r')
xlabel('Time [s]')
ylabel('x [m/s]')
subplot(3,1,2)
plot(time_list, y_forward_m_list, 'k', time_list, y_list,
'r')
xlabel('Time [s]')
ylabel('y [m/s]')
subplot(3,1,3)
plot(time_list, theta_forward_m_list, 'k', time_list,
theta_list, 'r')
xlabel('Time [s]')
ylabel('Theta [rad/s]')
```

PLOTS

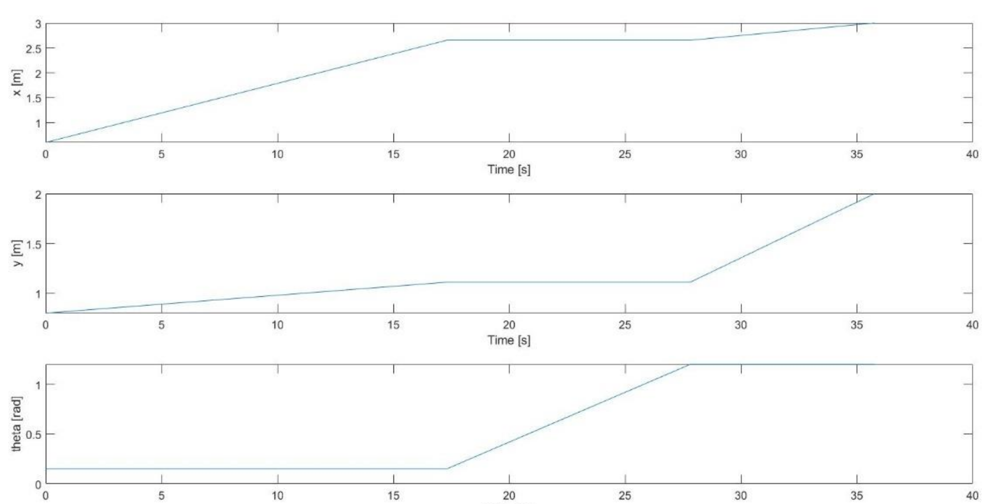
Points 1 – 6 and the Line Segments



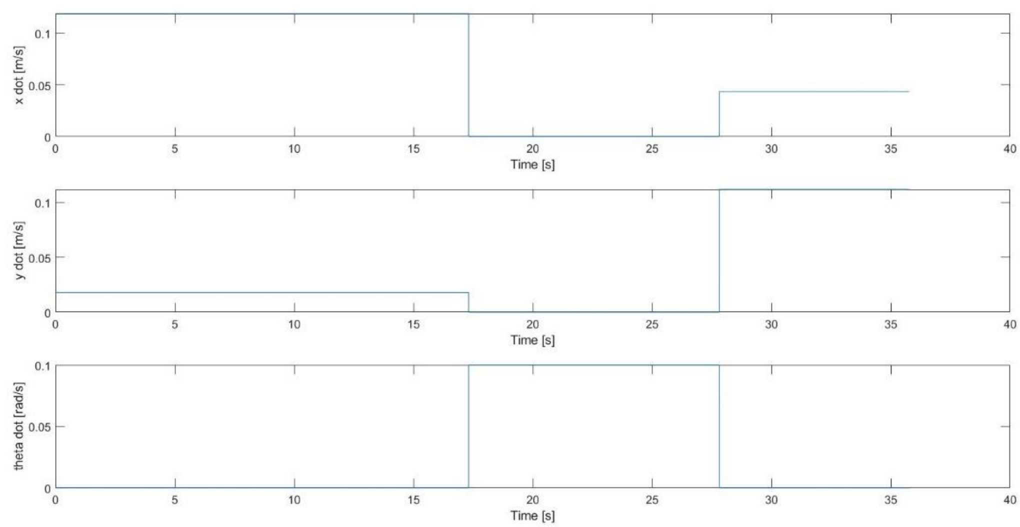
Cartesian Trajectory



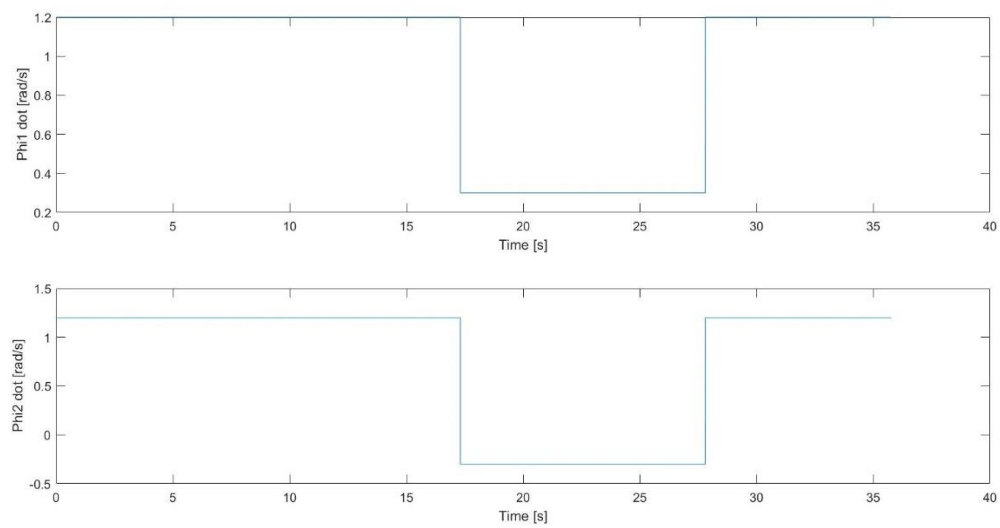
Instantaneous Pose Plots



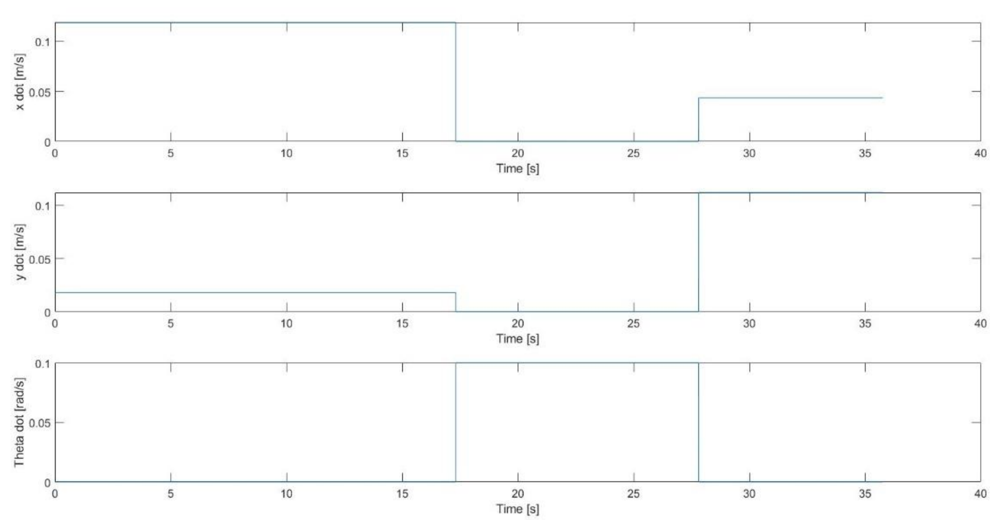
X_dot, Y_dot, and theta_dot



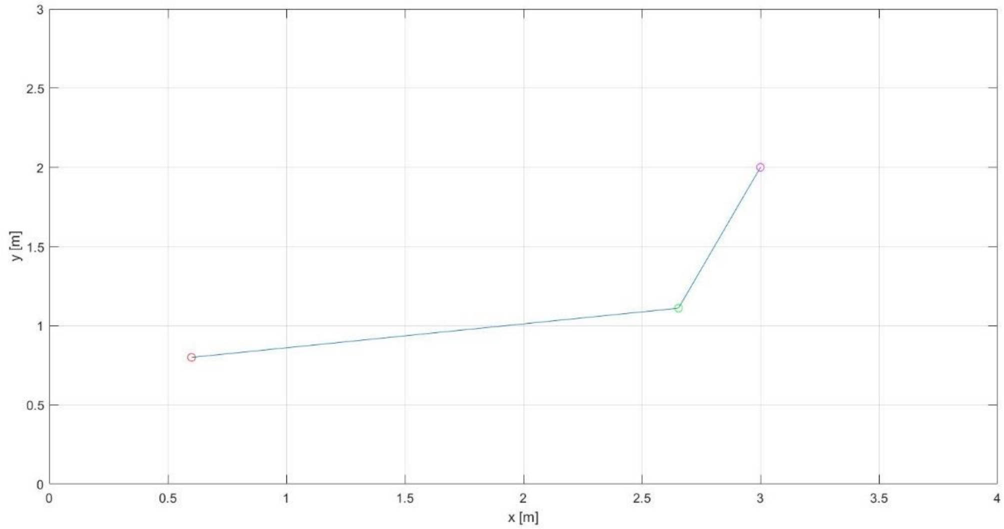
Wheel Speeds from Inverse Kinematics



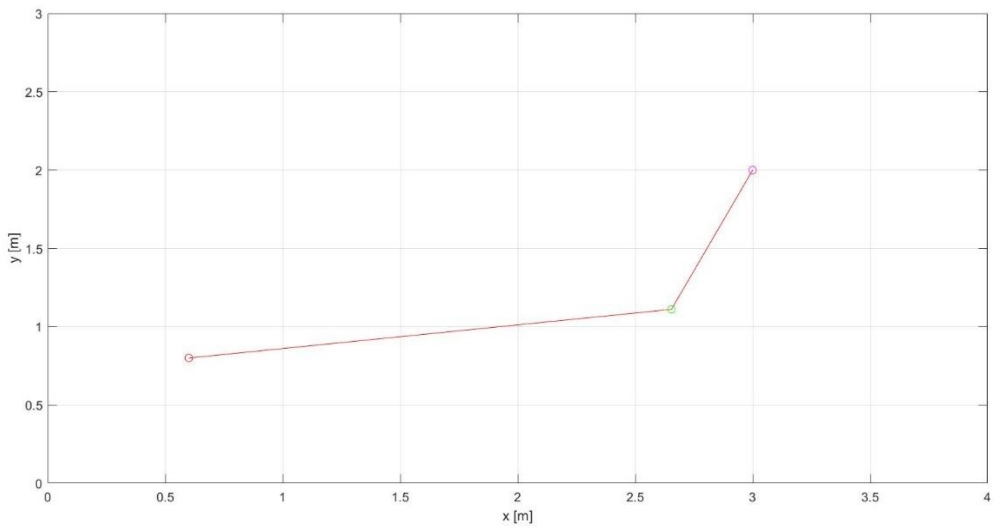
X_dot, Y_dot, and theta_dot from Forward Kinematics



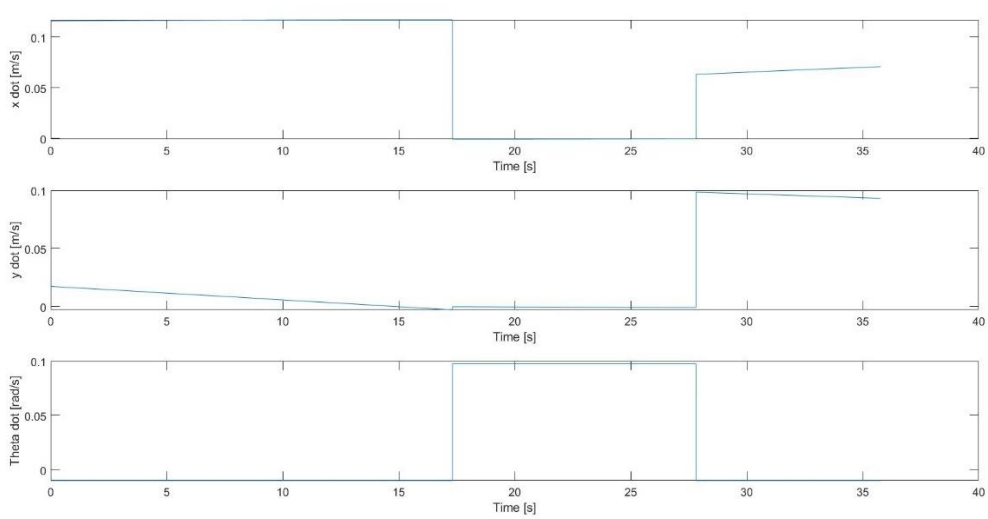
Cartesian Trajectory from Wheel Speeds (Cross Check)



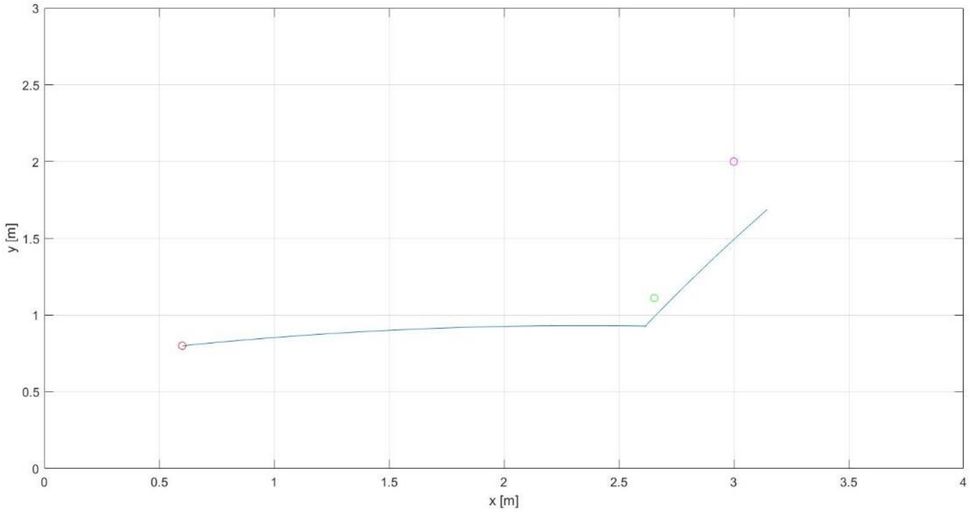
Cartesian Trajectory from Wheel Speeds and Original Cartesian Trajectory



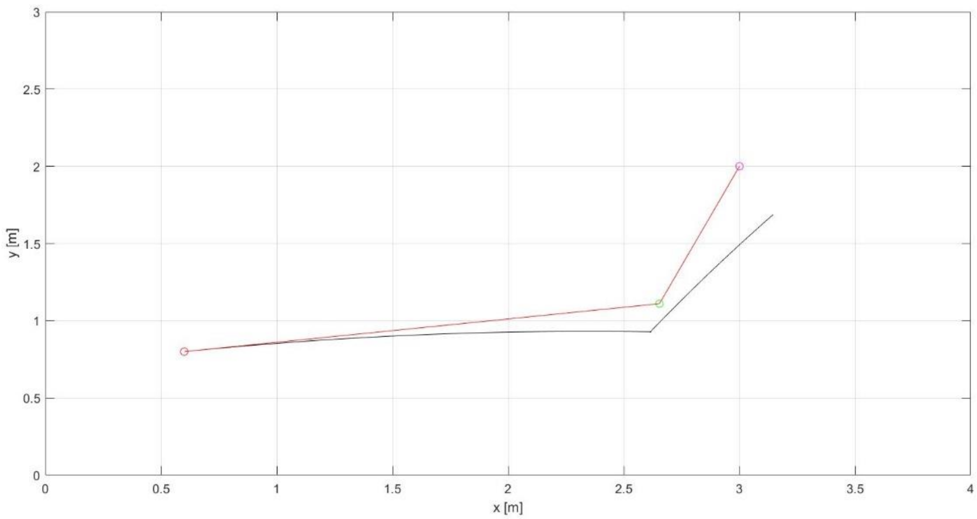
Forward Speed Kinematics for Parameter Mismatch



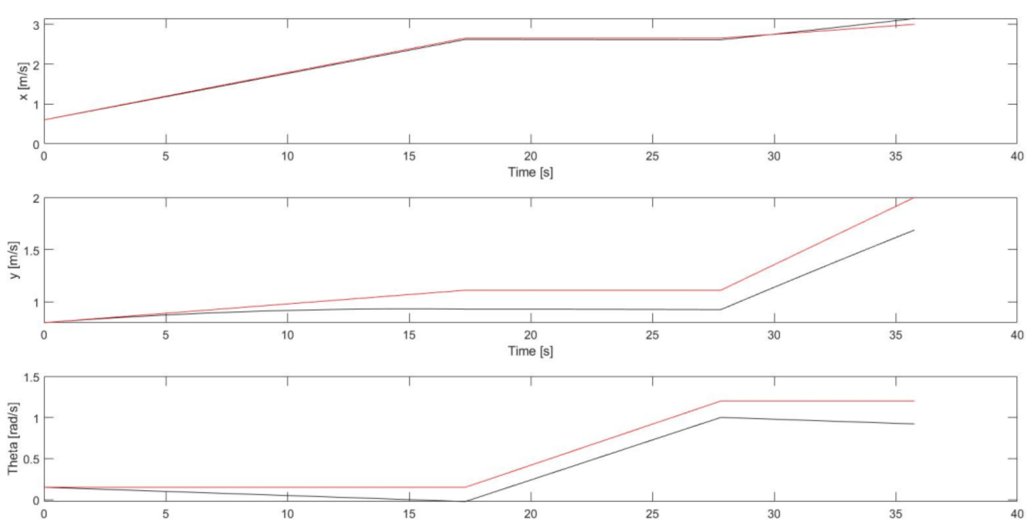
Cartesian Trajectory with Parameter Mismatch



Comparison – Original Cartesian Trajectory and Mismatch Cartesian Trajectory



Instantaneous Pose Plots with Parameter Mismatch



DISCUSSION

Comments on the Parameter Mismatch

- Since the radius of the right wheel (r1) is reduced, the robot is bound to veer towards the right. The cartesian trajectory in the previous section proves this.
- Also, the parameter mismatch applied was really significant because:
  - It changed the trajectory completely such that the final pose is different.
  - It altered the line segments to form curves with really large radii.
  - It caused y_dot to be negative for a brief period, while it caused theta_dot to be negative for most of the journey, except during the turn.
- A mismatch of this nature is realistic especially in uneven terrains like rocky plains, etc.

REFERENCES

Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza. 2011. Introduction to Autonomous Mobile Robots (2nd. ed.). The MIT Press.

Soto, Borja Garcia de and Miroslaw J. Skibniewski , "Future of robotics and automation in construction" , in Construction 4.0 ed. Anil Sawhney , Mike Riley and Javier Irizarry (Abingdon: Routledge, 12 Feb 2020), accessed 30 Oct 2021 , Routledge Handbooks Online.

https://en.wikipedia.org/wiki/Robot_locomotion#Walking

https://www.javatpoint.com/robot-locomotion

https://www.youtube.com/watch?v=rVlhMGQgDkY&t=161s

https://robots.ieee.org/robots/hrp5p/?gallery=video1

https://robots.ieee.org/robots/atlas2016/

https://whyps.com/hydraulic-system-advantages-and-disadvantages

https://kyr.fel.cvut.cz/en/atlas-next-generation

https://emerj.com/ai-adoption-timelines/biped-robot-timelines/

https://robots.ieee.org/robots/cassie/?gallery=interactive1

https://news.mit.edu/2019/mit-mini-cheetah-first-four-legged-robot-to-backflip-0304

https://www.bostondynamics.com/spot?utm_source=robots.ieee.org

https://robots.ieee.org/robots/anymal/?gallery=video3

https://www.youtube.com/watch?v=a7eqvg5rnus

APPENDIX

The following code is added for an arc radius of 1.2m just in case.

```
close all; clear;

% DATA ENTRY
%----------------------------------------
% Simulation Parameters - properly outline all the given
parameters.
step_time = 0.001; % We want to have data for every
millisecond.

% Robot Dimensions
r = 0.1; % radius of the wheel in meters.
l = 0.3; % Half of the width of the robot in meters.

% Initial Pose - [x1, y1, theta1]'
x1 = 0.6; % in meters.
y1 = 0.8; % in meters.
theta1 = 0.15; % Angles are always in radians ~ 10 degrees.
P1 = [x1; y1]; % A column vector

% Final Pose - [x2, y2, theta2]'
x2 = 3; % in meters.
y2 = 2; % in meters.
theta2 = 1.2; % in radians ~ 70 degrees
P2 = [x2; y2]; % A column vector

% Arc Parameters - the second segment of the trajectory is
an arc.
initial_arc_angle = -(pi/2) + theta1; % in radians .

% Speed Parameters
VT = 0.12; % Tangential VELOCITY in m/s - it has x and y
components which
% will serve as X_dot (= VT*cos(...)) and Y_dot (=
VT*sin(...))
arc_omega = 0.1; % in radians/second.
rc = VT/arc_omega; % in meters.
%----------------------------------------
% End of Data Entry

% COMPUTATION OF POINTS
```

```
%-----------------------------------------
% Computation of the line-to-line intersection point
data_matrix = [cos(theta1) -cos(theta2); sin(theta1) -
sin(theta2)];
lambda_P6 = (data_matrix)\(P2 - P1);
lambda1_P6 = lambda_P6(1);
P6 = P1 + lambda1_P6*[cos(theta1); sin(theta1)];

% Computation of the line-to-arc transition point
delta_theta = theta2 - theta1;
d = rc*tan(delta_theta/2);
P3 = P6 - d*[cos(theta1); sin(theta1)];

% Computation of the arc-to-line transition point
P4 = P6 + d*[cos(theta2); sin(theta2)];

% Computation of the center point of the circle
P5 = P3 + rc*[-sin(theta1); cos(theta1)];

% Plot the lines and points P1 to P6
figure %1
plot(P1(1), P1(2), 'ro')
hold
plot(P3(1), P3(2), 'mo')
plot(P6(1), P6(2), 'ko')
plot(P4(1), P4(2), 'co')
plot(P2(1), P2(2), 'bo')
plot(P5(1), P5(2), 'go')
plot([P1(1) P6(1)], [P1(2) P6(2)], 'k')
plot([P6(1) P2(1)], [P6(2) P2(2)], 'k')
axis([0 x2+1 0 y2+1])
xlabel('x [m]')
ylabel('y [m]')
grid
%-----------------------------------------
% End of Computation of Points

% COMPUTATION OF MOTION SEGMENT DURATIONS
%-----------------------------------------
% Distance Travelled
s_line1 = lambda1_P6 - d;
s_arc = rc*delta_theta;
```

```
lambda2_P6 = lambda_P6(2);
% lambda2_P6 is negative, and distance can't be negative;
so, we take its
% value.
s_line2 = abs(lambda2_P6) - d;

% Durations
duration_line1 = s_line1/VT;
duration_arc = s_arc/VT;
duration_line2 = s_line2/VT;
duration_total = duration_line1 + duration_arc +
duration_line2;

% Transition Times
t1 = 0;
t2 = duration_line1;
t3 = t2 + duration_arc;
t4 = duration_total;
%----------------------------------------
% End of Computation of Motion Segments Durations.

% GEOMETRIC CONSTRUCTION OF THE CARTESIAN REFERENCE CURVE
%----------------------------------------
% Data Storage Dimension - The +1 is for time = 0s.
array_length = ceil(duration_total/step_time) + 1;

% X, Y, and Theta Trajectory Computations.
x_list = zeros(array_length,1);
y_list = zeros(array_length,1);
theta_list = zeros(array_length,1);
time_list = zeros(array_length,1);

for iteration_index = 1:1:array_length
    % Convert the step time to instantaneous times
    time = (iteration_index - 1)* step_time;
    % Fill in the time list
    time_list(iteration_index) = time;

    % Find the pose at every millisecond within each
segment
    % Use P = Pi + lambda_i*[cos(theta1_2); sin(theta1_2)]
for the lines
```

```
    % Where lambda = (ins_time - last_cumulative_time)*VT
    % For the arc, use P = P5 + rc*[cos(arc_angle);
sin(arc_angle)]
    % Where arc_angle is the instantaneous angle -
    % initial_arc_angle + (ins_time -
last_cumulative_time)*VT

    if (t1 <= time) && (time < t2)
        P = P1 + time*VT*[cos(theta1); sin(theta1)];
        x = P(1);
        y = P(2);
        theta = theta1;
    end

    if (t2 <= time) && (time < t3)
        arc_angle = initial_arc_angle + (time -
t2)*arc_omega;
        P = P5 + rc*[cos(arc_angle); sin(arc_angle)];
        x = P(1);
        y = P(2);
        theta = theta1 + (time - t2)*arc_omega;
    end

    if (t3 <= time) && (time <= t4)
        P = P4 + (time - t3)*VT*[cos(theta2); sin(theta2)];
        x = P(1);
        y = P(2);
        theta = theta2;
    end

    x_list(iteration_index) = x;
    y_list(iteration_index) = y;
    theta_list(iteration_index) = theta;
end

figure %2
plot(x_list, y_list)
hold
plot(P1(1), P1(2), 'ro')
plot(P3(1), P3(2), 'mo')
plot(P6(1), P6(2), 'ko')
plot(P4(1), P4(2), 'co')
```

```
plot(P2(1), P2(2), 'bo')
plot(P5(1), P5(2), 'go')
axis([0 x2+1 0 y2+1])
xlabel('x [m]')
ylabel('y [m]')
grid

figure %3
subplot(3,1,1)
plot(time_list, x_list)
xlabel('time [s]')
ylabel('x [m]')
subplot(3,1,2)
plot(time_list, y_list)
xlabel('time [s]')
ylabel('y [m]')
subplot(3,1,3)
plot(time_list, theta_list)
xlabel('time [s]')
ylabel('theta [rad]')
%----------------------------------------
% End of the Geometric Construction of the Reference
Cartesian Curves

% GEOMETRIC CONSTRUCTION OF THE REFERENCE CARTESIAN SPEEDS
%----------------------------------------
x_dot_list = zeros(array_length,1);
y_dot_list = zeros(array_length,1);
theta_dot_list = zeros(array_length,1);
time_list = zeros(array_length,1);

for iteration_index = 1:1:array_length
    % Convert the step time to instantaneous times
    time = (iteration_index - 1)* step_time;
    % Fill in the time list
    time_list(iteration_index) = time;

    % Find the linear/angular velocities at every ms within
each segment
    if (t1 <= time) && (time < t2)
        x_dot = VT*cos(theta1);
        y_dot = VT*sin(theta1);
```

```
        theta_dot = 0;
    end

    if (t2 <= time) && (time < t3)
        theta = theta1 + (time - t2)*arc_omega;
        x_dot = VT*cos(theta);
        y_dot = VT*sin(theta);
        theta_dot = arc_omega;
    end

    if (t3 <= time) && (time <= t4)
        x_dot = VT*cos(theta2);
        y_dot = VT*sin(theta2);
        theta_dot = 0;
    end

    x_dot_list(iteration_index) = x_dot;
    y_dot_list(iteration_index) = y_dot;
    theta_dot_list(iteration_index) = theta_dot;
end

figure %4
subplot(3,1,1)
plot(time_list, x_dot_list)
xlabel('time [s]')
ylabel('x dot [m/s]')
subplot(3,1,2)
plot(time_list, y_dot_list)
xlabel('time [s]')
ylabel('y dot [m/s]')
subplot(3,1,3)
plot(time_list, theta_dot_list)
xlabel('time [s]')
ylabel('theta dot [rad/s]')
%----------------------------------------
% End of the Geometric Construction of the Reference
Cartesian Speeds

% INVERSE KINEMATICS FOR WHEEL SPEEDS
%----------------------------------------
phi1_dot_list = zeros(array_length,1);
phi2_dot_list = zeros(array_length,1);
```

```
for iteration_index = 1:1:array_length

    phi1_dot = (cos(theta_list(iteration_index))...
        *x_dot_list(iteration_index))...
        /r...
        + (sin(theta_list(iteration_index))...
        *y_dot_list(iteration_index))...
        /r...
        + l...
        *theta_dot_list(iteration_index)...
        /r;
     phi2_dot = (cos(theta_list(iteration_index))...
        *x_dot_list(iteration_index))...
        /r...
        + (sin(theta_list(iteration_index))...
        *y_dot_list(iteration_index))...
        /r...
        - l...
        *theta_dot_list(iteration_index)...
        /r;

    phi1_dot_list(iteration_index) = phi1_dot;
    phi2_dot_list(iteration_index) = phi2_dot;
end

figure %5
subplot(2,1,1)
plot(time_list, phi1_dot_list)
xlabel('time [s]')
ylabel('Phi1 dot [rad/s]')
subplot(2,1,2)
plot(time_list, phi2_dot_list)
xlabel('time [s]')
ylabel('Phi2 dot [rad/s]')
%----------------------------------------
% End of Inverse Kinematic for Wheel Speeds

% FORWARD KINEMATICS CROSSCHECK FOR CARTESIAN SPEEDS
%----------------------------------------
x_dot_forward_list = zeros(array_length,1);
y_dot_forward_list = zeros(array_length,1);
```

```
theta_dot_forward_list = zeros(array_length,1);

current_theta = theta1;

for iteration_index = 1:1:array_length

    pose_forward...
                = [cos(current_theta) -sin(current_theta) 0;
                   sin(current_theta)  cos(current_theta) 0;
                   0             0            1]...
                   *[r*phi1_dot_list(iteration_index)/2 +
r*phi2_dot_list(iteration_index)/2;
                     0;
                     r*phi1_dot_list(iteration_index)/(2*l) -
r*phi2_dot_list(iteration_index)/(2*l)];

    x_dot_forward = pose_forward(1);
    y_dot_forward = pose_forward(2);
    theta_dot_forward = pose_forward(3);

    x_dot_forward_list(iteration_index) = x_dot_forward;
    y_dot_forward_list(iteration_index) = y_dot_forward;
    theta_dot_forward_list(iteration_index) =
theta_dot_forward;

    current_theta = current_theta +
step_time*theta_dot_forward;
end

figure %6
subplot(3,1,1)
plot(time_list, x_dot_forward_list)
xlabel('time [s]')
ylabel('x dot [m/s]')
subplot(3,1,2)
plot(time_list, y_dot_forward_list)
xlabel('time [s]')
ylabel('y dot [m/s]')
subplot(3,1,3)
plot(time_list, theta_dot_forward_list)
xlabel('time [s]')
ylabel('Theta dot [rad/s]')
```

```
%-------------------------------------------
% End of Forward Kinematic Crosscheck for Cartesian Speeds

% CARTESIAN SPEED INTEGRATION CROSSCHECK FOR CARTESIAN
POSITION
%-------------------------------------------
x_forward_list = zeros(array_length,1);
y_forward_list = zeros(array_length,1);
theta_forward_list = zeros(array_length,1);

current_x = x1; current_y = y1; current_theta = theta1;

for iteration_index = 1:1:array_length
    x_forward = current_x;
    y_forward = current_y;
    theta_forward = current_theta;

    x_forward_list(iteration_index) = x_forward;
    y_forward_list(iteration_index) = y_forward;
    theta_forward_list(iteration_index) = theta_forward;

    current_x = current_x +
step_time*x_dot_forward_list(iteration_index);
    current_y = current_y +
step_time*y_dot_forward_list(iteration_index);
    current_theta = current_theta +
step_time*theta_dot_forward_list(iteration_index);
end

figure %7
plot(x_forward_list, y_forward_list)
hold
plot(P1(1), P1(2), 'ro')
plot(P3(1), P3(2), 'mo')
plot(P6(1), P6(2), 'ko')
plot(P4(1), P4(2), 'co')
plot(P2(1), P2(2), 'bo')
plot(P5(1), P5(2), 'go')
axis([0 x2+1 0 y2+1])
xlabel('x [m]')
ylabel('y [m]')
grid
```

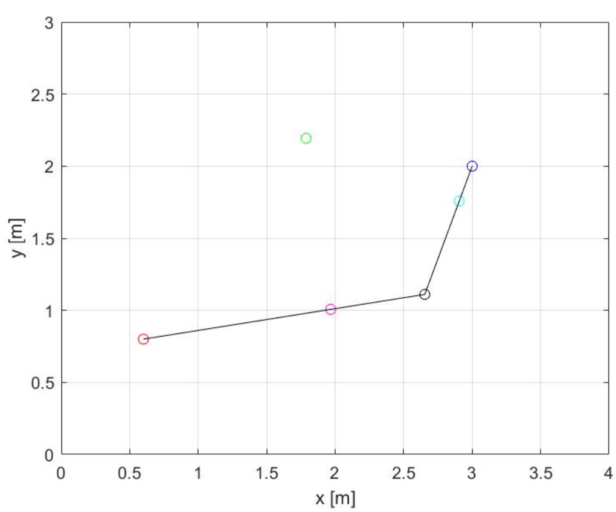
```
figure %8
plot(x_forward_list, y_forward_list, 'k')
hold
plot(x_list, y_list, 'r')
plot(P1(1), P1(2), 'ro')
plot(P3(1), P3(2), 'mo')
plot(P6(1), P6(2), 'ko')
plot(P4(1), P4(2), 'co')
plot(P2(1), P2(2), 'bo')
plot(P5(1), P5(2), 'go')
axis([0 x2+1 0 y2+1])
xlabel('x [m]')
ylabel('y [m]')
grid
%---------------------------------------
% End of Cartesian Speed Integration Crosscheck for
Cartesian Position

% INTRODUCTION OF A MODEL PARAMETER MISMATCH ON FORWARD
SPEED KINEMATICS
%---------------------------------------
% Parameter mismatch on the right wheel (r1).
r1 = r*0.95;
x_dot_forward_m_list = zeros(array_length,1);
y_dot_forward_m_list = zeros(array_length,1);
theta_dot_forward_m_list = zeros(array_length,1);

current_theta_m = theta1;

for iteration_index = 1:1:array_length

    pose_forward_mismatch...
            = [cos(current_theta_m) -sin(current_theta_m)
0;
               sin(current_theta_m)  cos(current_theta_m)
0;
               0                     0
1]...
               *[r1*phi1_dot_list(iteration_index)/2 +
r*phi2_dot_list(iteration_index)/2;
                 0;
```

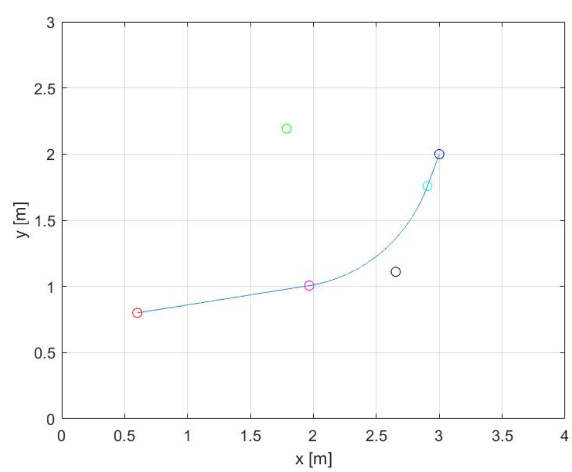
```
                 r1*phi1_dot_list(iteration_index)/(2*l) -
r*phi2_dot_list(iteration_index)/(2*l)];

    x_dot_forward_m = pose_forward_mismatch(1);
    y_dot_forward_m = pose_forward_mismatch(2);
    theta_dot_forward_m = pose_forward_mismatch(3);

    x_dot_forward_m_list(iteration_index) =
x_dot_forward_m;
    y_dot_forward_m_list(iteration_index) =
y_dot_forward_m;
    theta_dot_forward_m_list(iteration_index) =
theta_dot_forward_m;

    current_theta_m = current_theta_m +
step_time*theta_dot_forward_m;
end

figure %9
subplot(3,1,1)
plot(time_list, x_dot_forward_m_list)
xlabel('time [s]')
ylabel('x dot [m/s]')
subplot(3,1,2)
plot(time_list, y_dot_forward_m_list)
xlabel('time [s]')
ylabel('y dot [m/s]')
subplot(3,1,3)
plot(time_list, theta_dot_forward_m_list)
xlabel('time [s]')
ylabel('Theta dot [rad/s]')
%---------------------------------------
% End of Introduction of A Model Parameter Mismatch on
Forward Speed Kinematics

% INTEGRATION OF FORWARD KINEMATICS CARTESIAN SPEEDS WITH
PARAMETER MISMATCH
%---------------------------------------
x_forward_m_list = zeros(array_length,1);
y_forward_m_list = zeros(array_length,1);
theta_forward_m_list = zeros(array_length,1);
```

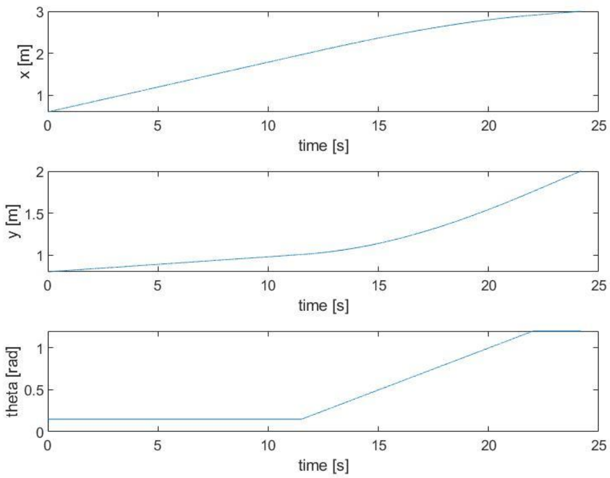
```
current_x_m = x1; current_y_m = y1; current_theta_m =
theta1;

for iteration_index = 1:1:array_length
    x_forward_m = current_x_m;
    y_forward_m = current_y_m;
    theta_forward_m = current_theta_m;

    x_forward_m_list(iteration_index) = x_forward_m;
    y_forward_m_list(iteration_index) = y_forward_m;
    theta_forward_m_list(iteration_index) =
theta_forward_m;

    current_x_m = current_x_m +
step_time*x_dot_forward_m_list(iteration_index);
    current_y_m = current_y_m +
step_time*y_dot_forward_m_list(iteration_index);
    current_theta_m = current_theta_m +
step_time*theta_dot_forward_m_list(iteration_index);
end

figure %10
plot(x_forward_m_list, y_forward_m_list)
hold
plot(P1(1), P1(2), 'ro')
plot(P3(1), P3(2), 'mo')
plot(P6(1), P6(2), 'ko')
plot(P4(1), P4(2), 'co')
plot(P2(1), P2(2), 'bo')
plot(P5(1), P5(2), 'go')
axis([0 x2+1 0 y2+1])
xlabel('x [m]')
ylabel('y [m]')
grid

figure %11
plot(x_forward_m_list, y_forward_m_list, 'k')
hold
plot(x_list, y_list, 'r')
plot(P1(1), P1(2), 'ro')
plot(P3(1), P3(2), 'mo')
plot(P6(1), P6(2), 'ko')
```

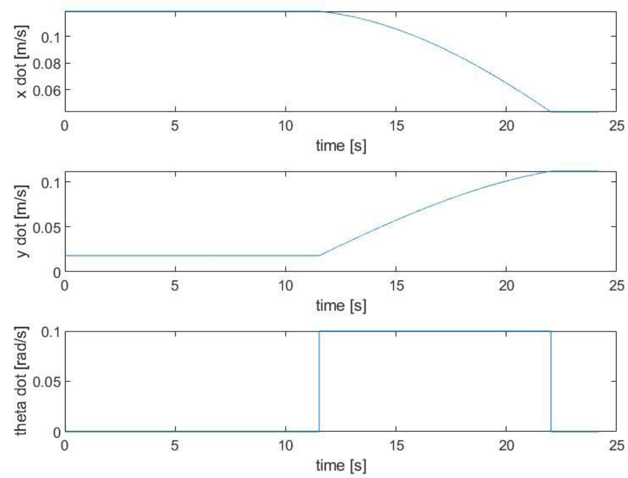
```
plot(P4(1), P4(2), 'co')
plot(P2(1), P2(2), 'bo')
plot(P5(1), P5(2), 'go')
axis([0 x2+1 0 y2+1])
xlabel('x [m]')
ylabel('y [m]')
grid
%---------------------------------------
% End of Integration of Forward Kinematics Cartesian Speeds
with Parameter Mismatch

figure %12
subplot(3,1,1)
plot(time_list, x_forward_m_list, 'k', time_list, x_list,
'r')
xlabel('time [s]')
ylabel('x [m/s]')
subplot(3,1,2)
plot(time_list, y_forward_m_list, 'k', time_list, y_list,
'r')
xlabel('time [s]')
ylabel('y [m/s]')
subplot(3,1,3)
plot(time_list, theta_forward_m_list, 'k', time_list,
theta_list, 'r')
xlabel('time [s]')
ylabel('Theta [rad/s]')
```

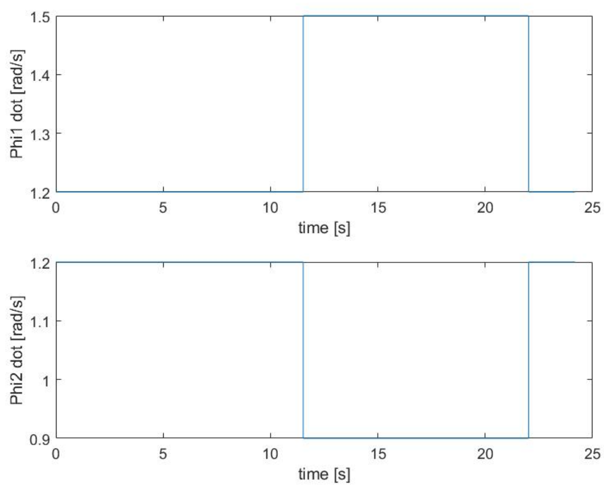
RESULTS

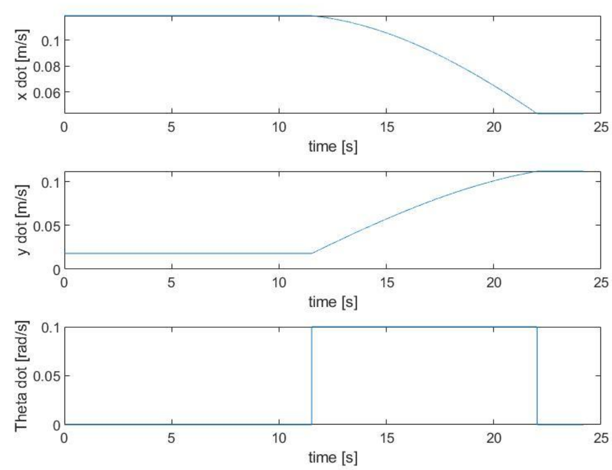


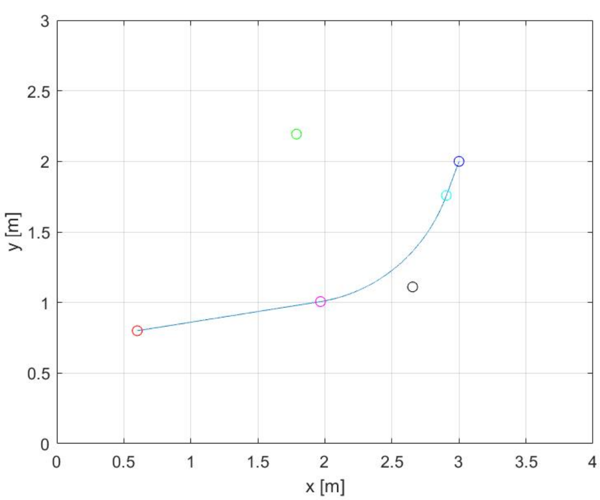


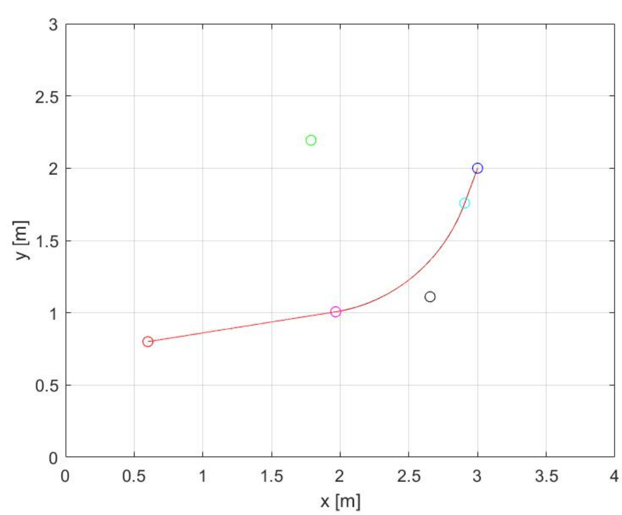

x [m]
time [s]
y [m]
time [s]
theta [rad]
time [s]

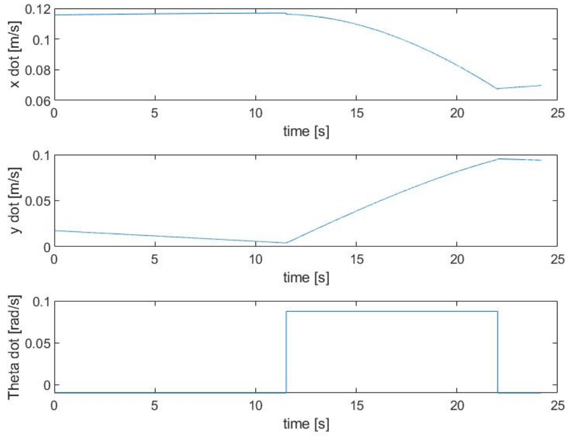

x dot [m/s]
time [s]
y dot [m/s]
time [s]
theta dot [rad/s]
time [s]

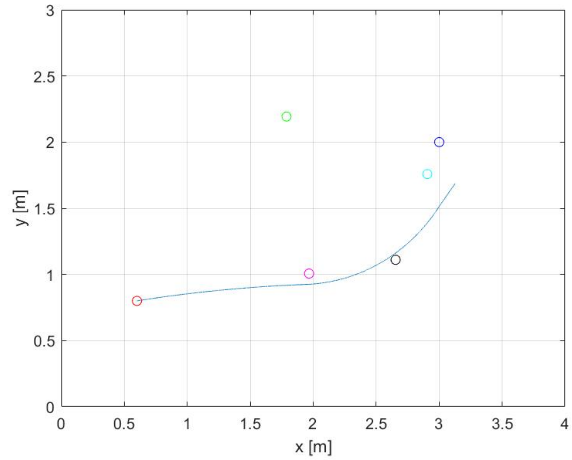

Phi1 dot [rad/s]
time [s]
Phi2 dot [rad/s]
time [s]

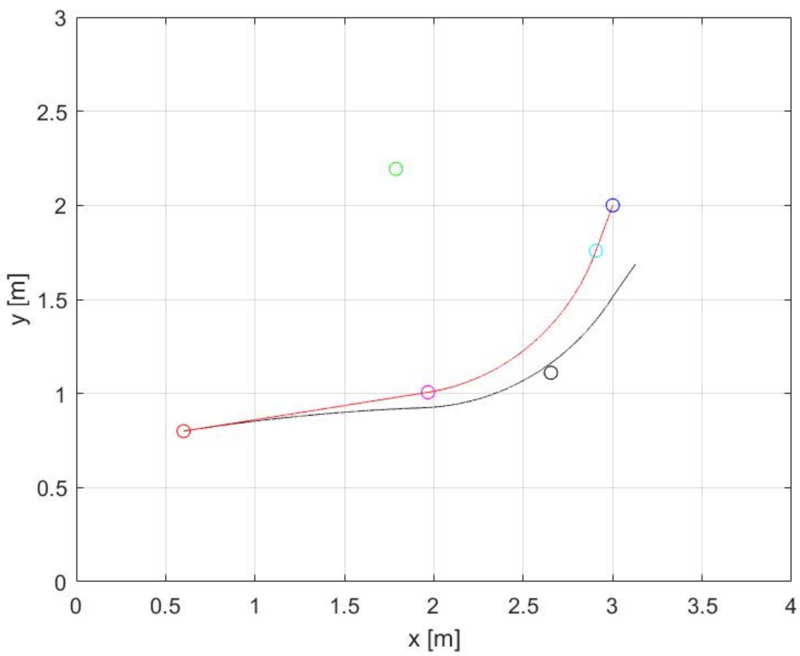

x dot [m/s]
time [s]
y dot [m/s]
time [s]
Theta dot [rad/s]
time [s]

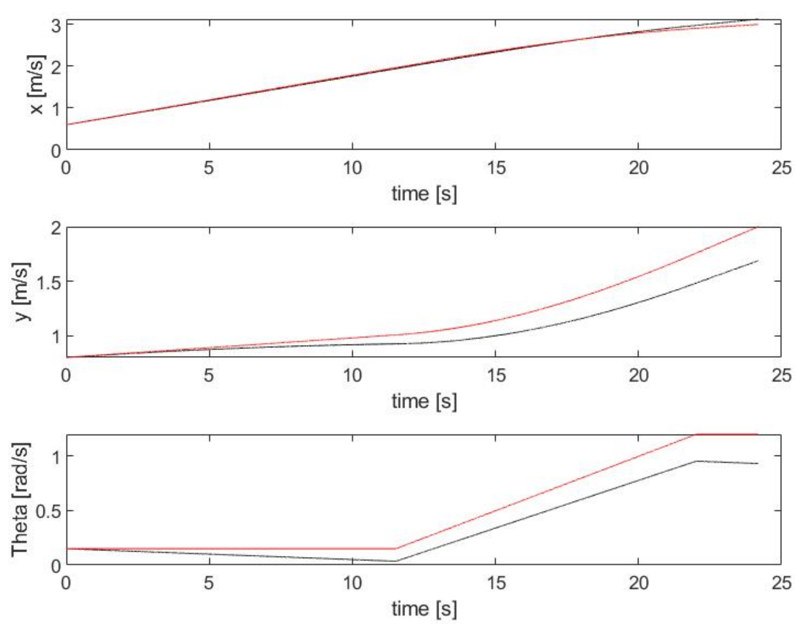

y [m]
x [m]


y [m]
x [m]


x dot [m/s]
time [s]
y dot [m/s]
time [s]
Theta dot [rad/s]
time [s]


y [m]
x [m]


y [m]
x [m]


x [m/s]
time [s]
y [m/s]
time [s]
Theta [rad/s]
time [s]