TWO-WHEELED ROBOT CONSTRUCTION AND ODOMETRY ESTIMATION

NOVEMBER 30, 2021

Submitted to: Prof. Dr. Kemalettin Erbatur

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

TABLE OF CONTENTS

INTRODUCTION

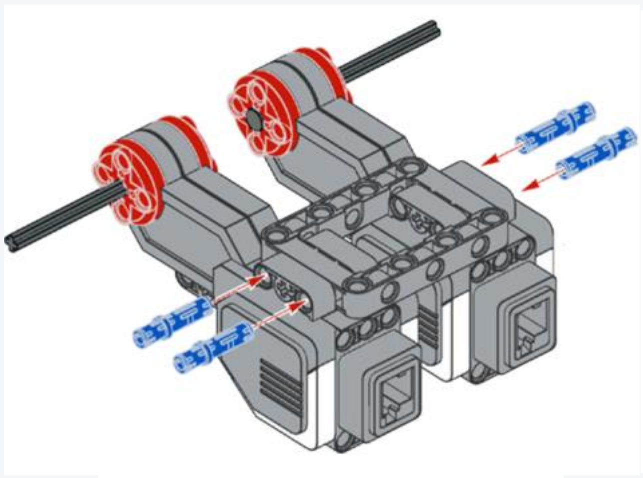
This lab involves the construction of a two-wheeled robot followed by the implementation of control mechanisms using MATLAB to move the robot in different direction in order to estimate the odometry.

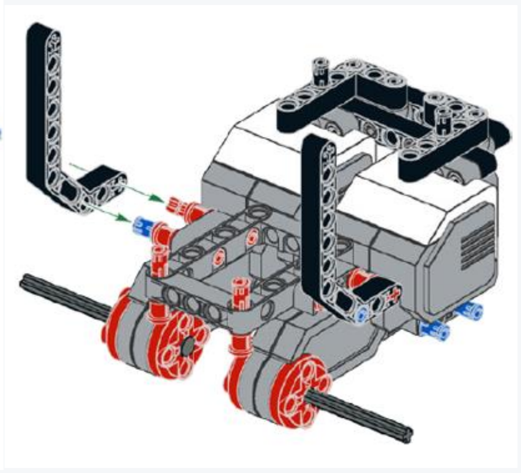
The major aim of this lab is to examine the trajectory formed during different forms of motion by a two-wheeled robot. Basically, we apply the differential drive principle on a two-wheeled robot. In this case, the robot is built using parts from the EV3 Mindstorm toolkit.

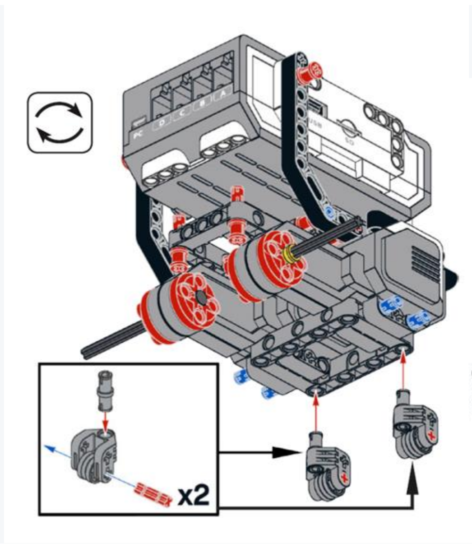
Linear motion and circular motion are implemented to obtain data for the calculation of the odometry. With the data, calculated trajectories are plotted and compared with desired trajectories.

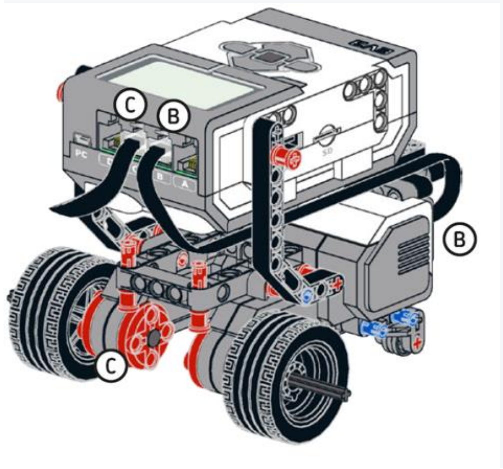
PROCEDURE

Firstly, the robot is assembled using the necessary parts as shown in the figures below:









The major components used are:

- Two large motors - One touch sensor
- One infrared sensor - A remote infrared beacon
- Two actuated wheels - Two castor wheels - The EV3 Brick

MATLAB is the underpinning software for this application.

To implement linear and circular motion and plot the odometry of the robot, some programs are written in MATLAB. Prior to this, a USB connection is established between the robot and the PC on which the MATLAB code is written.

MATLAB Code

After creating a connection to the Ev3, the motors and IR sensors are properly defined.

- Arrays (x, y, and θ) are created to store the necessary data that will be obtained from the pose (position and orientation) of the robot.
- The first elements of the above arrays are initialized to 0 to ensure that the robots start from the origin with an initial orientation of 0.
- A counter (k) is created to facilitate storage of data in the aforementioned arrays.
- Robot parameters like wheel radius, R, and L (the distance between the two actuated wheels) are defined.
- A while loop is used to make the robot move for some time for proper data collection.
- Within the while loop, linear and circular motions are performed using inbuilt ev3 functions.
- A beacon channel is defined and, with the remote control, if statements are used to implement the motion.

Forward motion: the speeds of the left and right motors are set at the same value. Results are obtained for different speeds.

Backward motion: For this, the speeds of the left and right motors are set at the same position value. Results are obtained for different speeds.

Circular motion: The speed of the left motor is set to zero while that of the right motor is non-zero.

- After every motion run within the while loop, some computations are performed using the encoder speeds for the left and right motors($\triangle E_r \text{ and } \triangle E_l$):
  i. The distance travelled by each wheel is calculated by:

$$D_r = 2\pi R\frac{\triangle E_r}{360}, \qquad D_l = 2\pi R\frac{\triangle E_l}{360}$$

ii. With the above results, the distance travelled by the center of the wheels and the change in angle are calculated:

$$D_c = \frac{D_r + D_l}{2}, \qquad \Delta\theta = \frac{D_r - D_l}{L}$$

- Using the above results, the motion of the robot is calculated and stored in the earlier defined arrays before the counter is incremented for the next run.

$$x(k+1) = x(k) + D_c.cos\left(\theta(k) + \frac{\Delta\theta}{2}\right)$$
$$y(k+1) = y(k) + D_c.sin\left(\theta(k) + \frac{\Delta\theta}{2}\right)$$
$$\theta(k+1) = \theta(k) + \Delta\theta$$

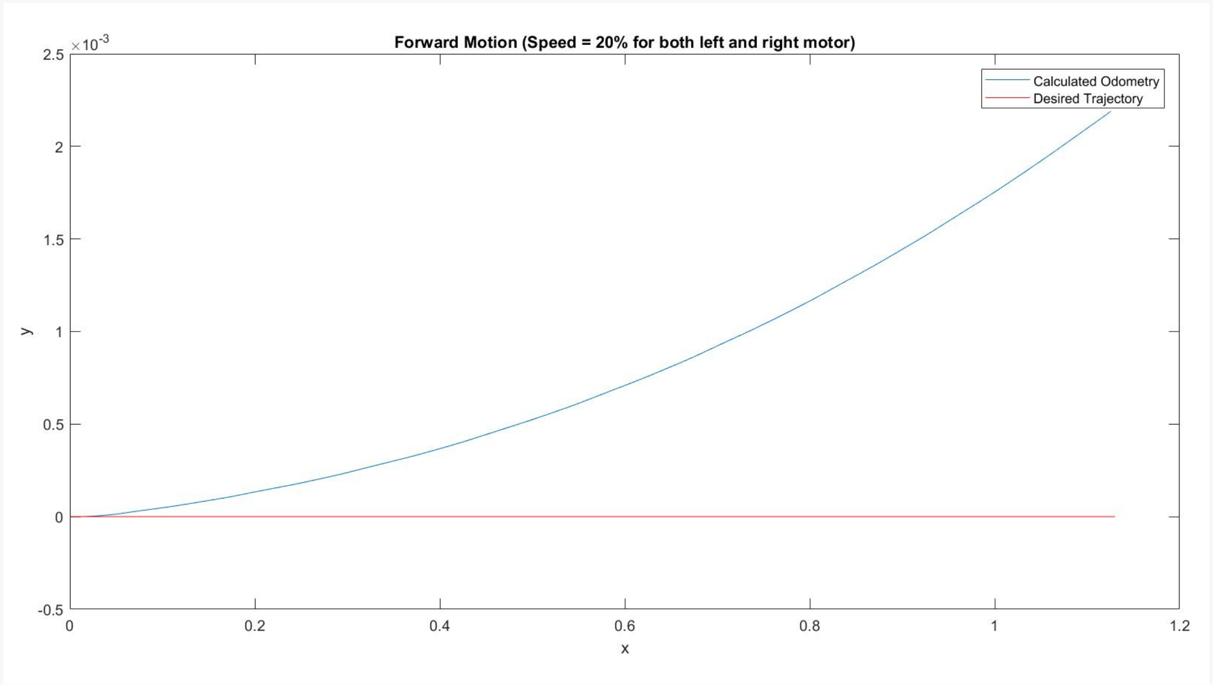
- Finally, the results are plotted.

For the circular motion, the time period is altered using the pause function.

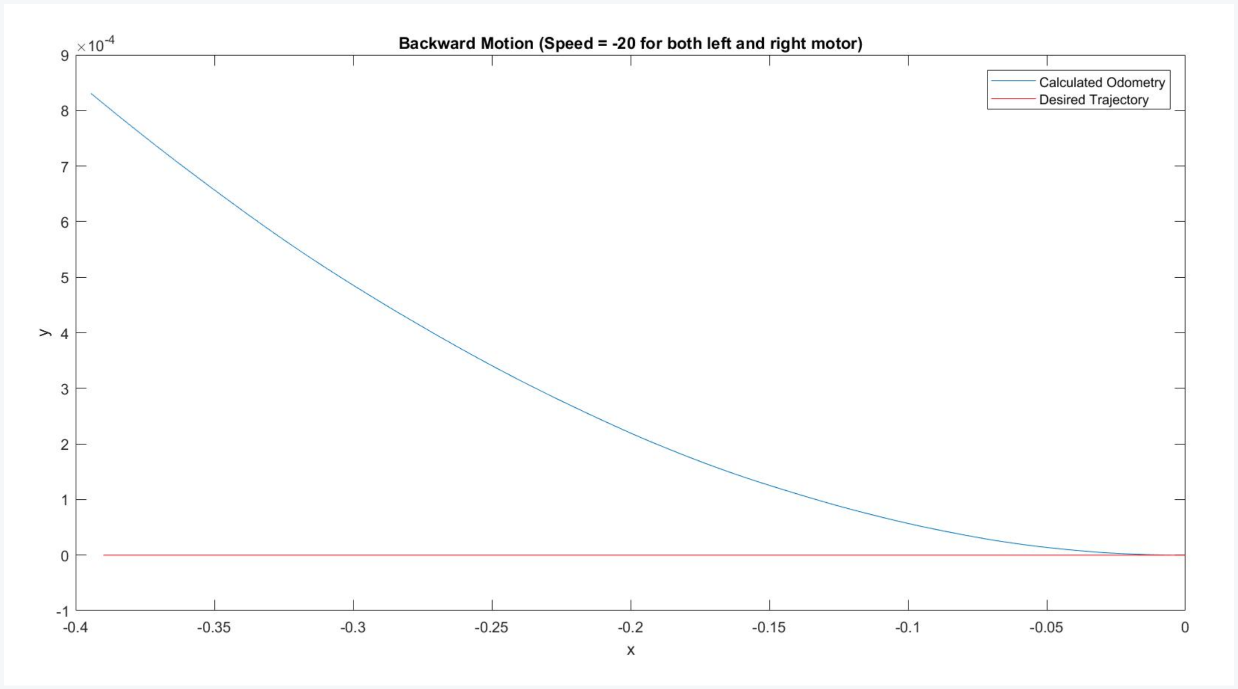
RESULTS

The following results were obtained from the implementation of the MATLAB code on the two-wheeled robot.

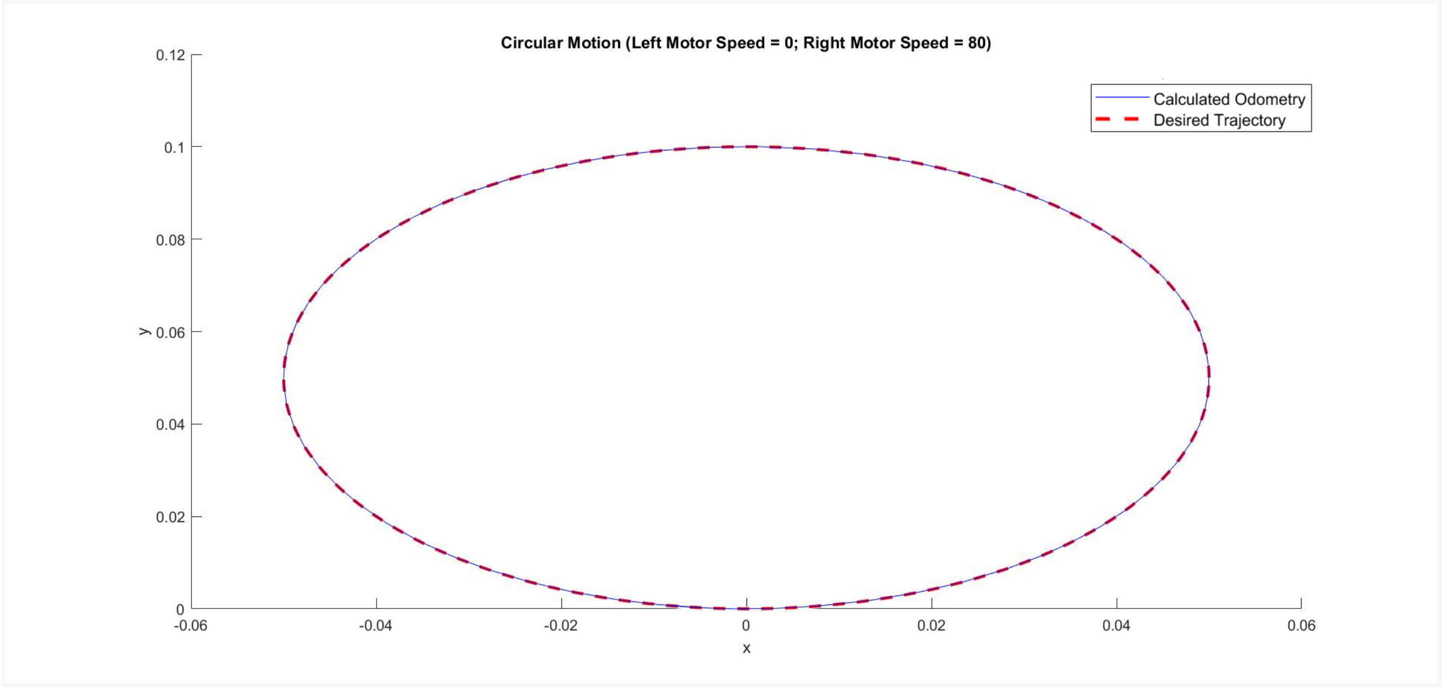
Forward Motion:



Backward Motion:



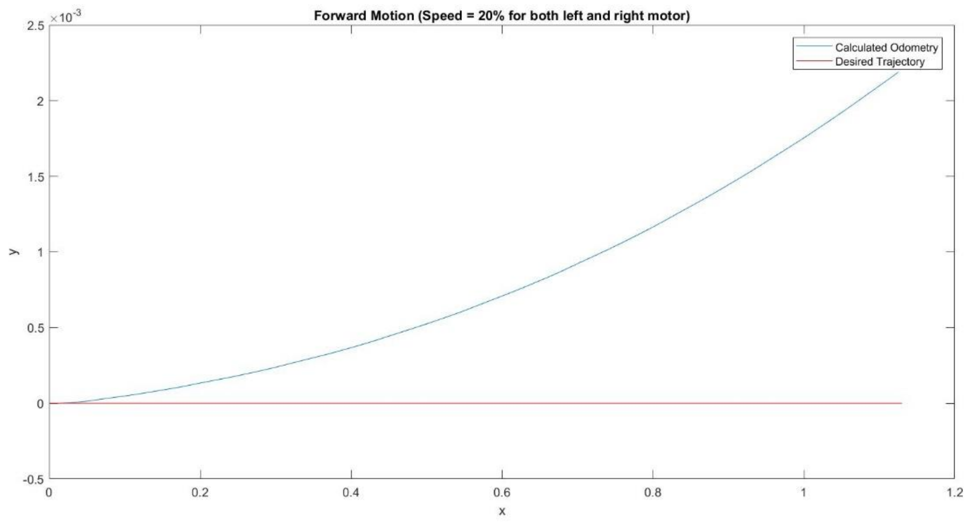
Circular Motion:



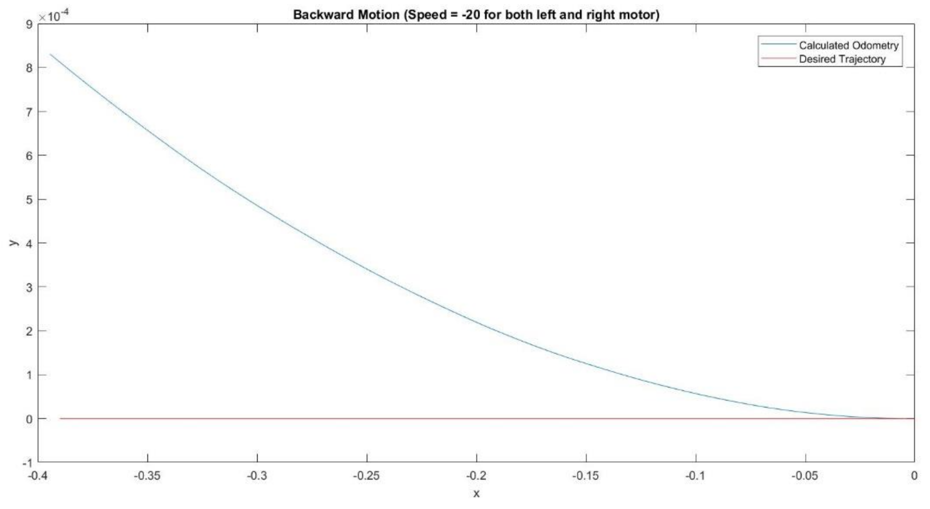
DISCUSSION

Linear Motion Q1

- Calculated Odometry: partly due to the initial position of the castor wheels and the motors used, the calculated odometry does not follow a straight line. Rather, it gradually curves away from the horizontal axis as seen below:



- Desired Trajectory: this would be in line with (parallel with) the horizontal axis as shown above.
- Error: Comparing the calculated odometry and the desired trajectory, the error at any instant **is the distance from the desired trajectory to the calculated odometry along the vertical axis**.



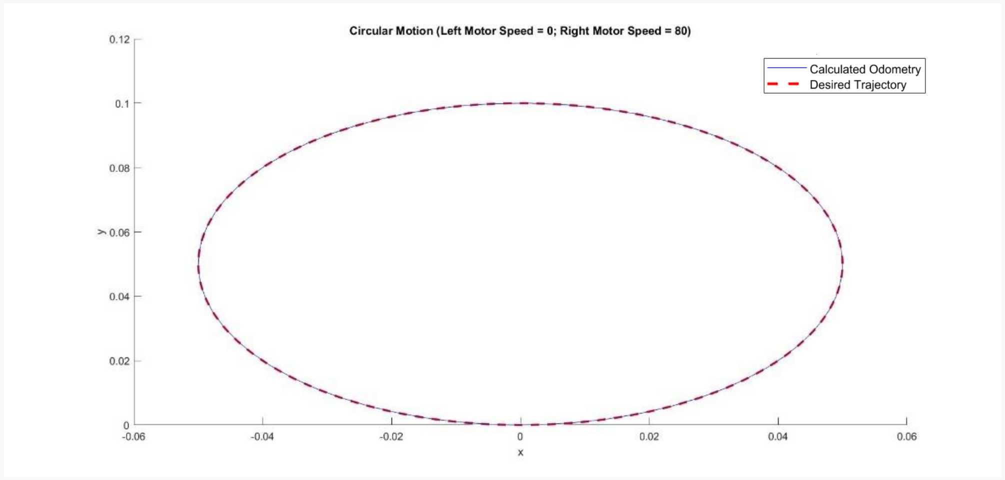
- The error for the forward motion ranges from 0 to 2.2x10^{-3}, while that for the backward motion ranges from 0 to 8x10^{-3} for the region considered.
- In this case, the error is cumulative; this implies that it keeps adding up as the distance travelled increases.

Circular Motion

Q2

- Calculated Odometry: The shape is an ellipse (as expected) with a major axis length of 0.1, and a minor axis length of 0.1. This corresponds to the distance between the two wheels, L.

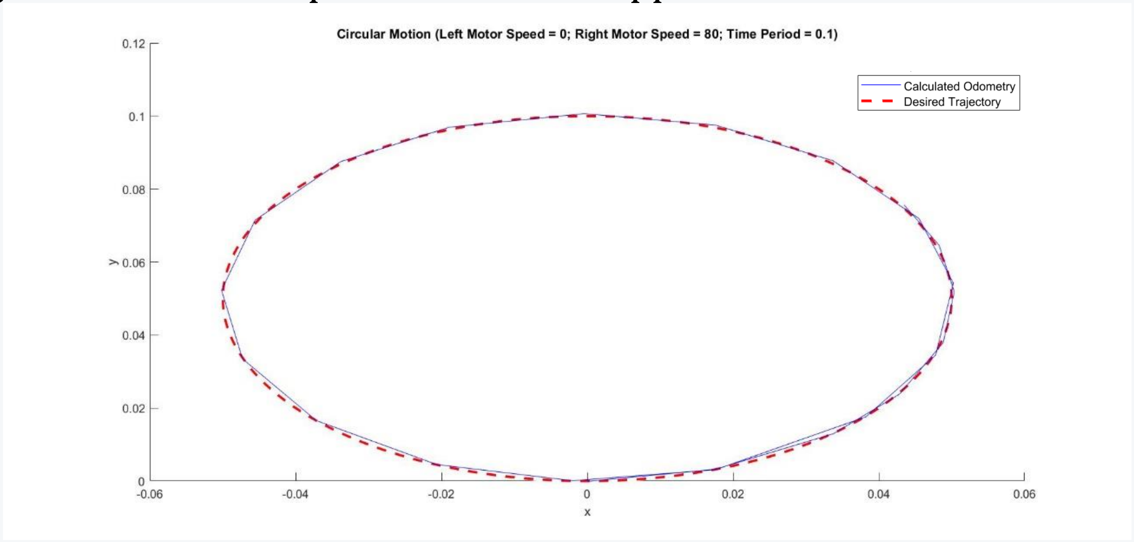
- Desired Trajectory: The desired trajectory is also an ellipse with major and minor axis lengths of 0.1. This is the case because it must tally with the distance between the two wheels given that the right wheel rotates around the static left wheel.



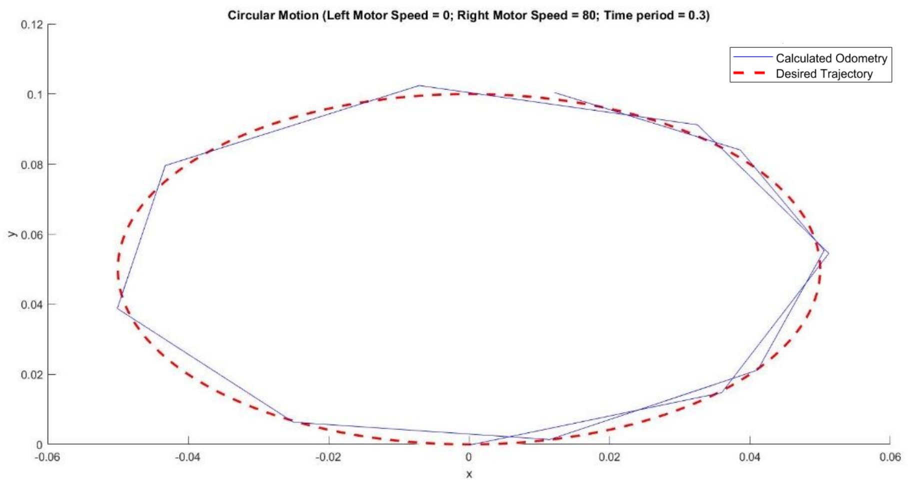
- Error: As seen above, a delightful result is obtained with **next-to-no error**. In the actual sense, a minor error might exist; however, from direct observation, whatever error that may be present is negligible.

Q3 – Varying Time Periods

- The odometry estimation is directly affecting by changing the time period.
- Once we start increasing the time period from 0, we immediately notice that the ellipse develops sharp corners at regular intervals around its circumference. In the figure below, a time period of 0.1 was applied:



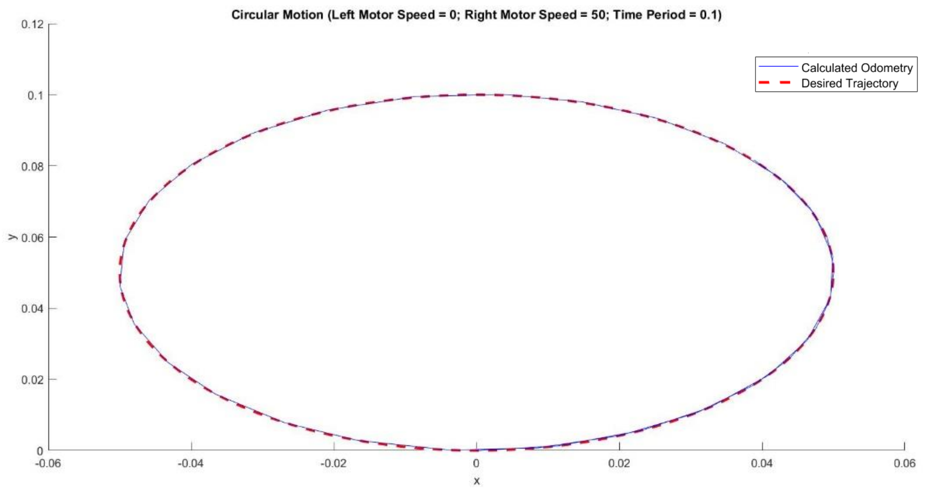
- Notice how the new trajectory differs from the desired trajectory.
- Also, the errors increase due the change in the time period.
- Further increasing the time period to 0.3 for the same speed, we obtain the following:

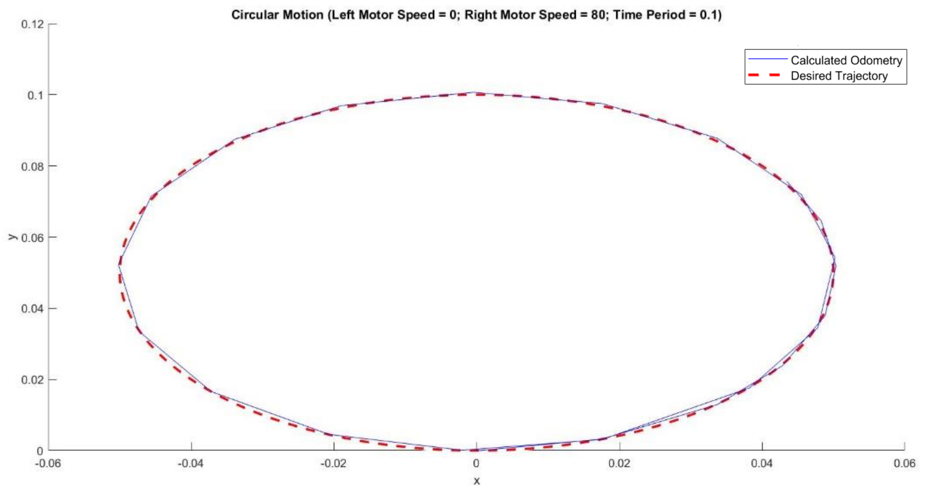


- The odometry estimation changes even more dramatically, while the error increases.

Q4 – Varying Power/Speed of the Right Wheel

- For the same time period (0.1), the experiment was run at right wheel speeds of 50 and 80, and the following results were obtained:

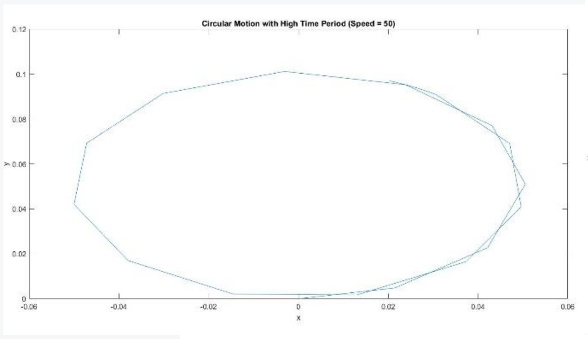


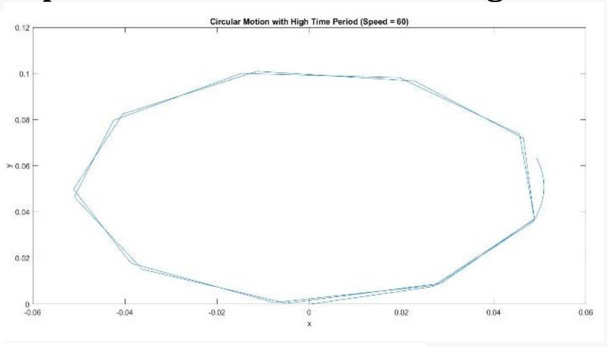


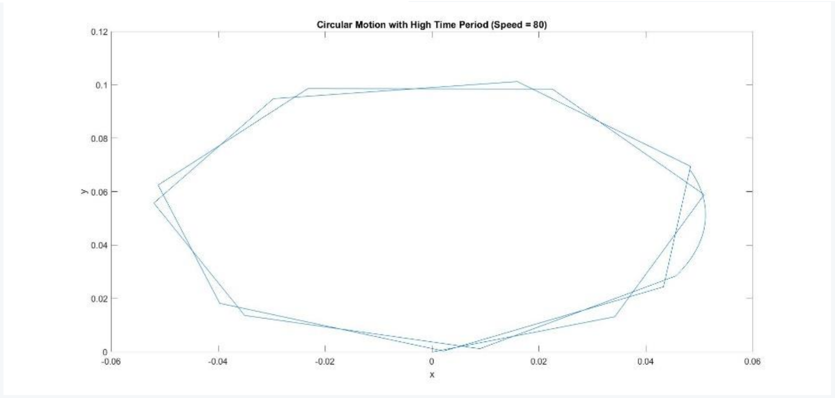
- From the above, we observe that the odometry estimations are better at lower speeds than at higher speeds. This could be due to vibrations or other factors that come into the fold when power is increased.
- Also, the error increases significantly as we increase the speed. The reason would also be credited to the aforementioned factors.

More Results

For the same time period, the odometry at speed = 50, 60, and 80 are given below:







- The odometry estimation gets worse as we increase the power, and the error also increases.

CONCLUSION

This lab includes a condensed study on application of the differential drive principle on a wheeled robot. In this case, a robot with two actuated (front) and two passive (rear) castor wheels which could be modeled as a two-wheeled robot was used. One of the major topics of discuss in this lab was the comparison of experimental/calculated odometry with ideal trajectory. This was also accompanied by error calculation to determine the deviation of experimental results from theoretical ones. Furthermore, the effects of varying speeds and time periods were explored to ascertain their impact on odometry estimation an error generation. In general, the outcome of this experiment was expected barring some inevitable discrepancies due to hardware constraints that introduced errors. For example, the deviation odometry estimation while varying speeds could be as a result of vibration, wear and tear, friction, etc.

APPENDIX

Major Code

```
clear all;
clc;

mylego = legoev3('usb');

% Set up infrared sensor on port 1
myirsensor = irSensor(mylego, 2);

% Set up the motors
Rightmotor = motor(mylego,'B');
Leftmotor = motor(mylego,'C');

% The Position and Orientation Arrays.
x_array = [];
y_array = [];
theta_array = [];

x_array_d = [];
y_array_d = [];
theta_array_d = [];

x_array(1) = 0;
y_array(1) = 0;
theta_array(1) = 0;

x_array_d(1) = 0;
y_array_d(1) = 0;
theta_array_d(1) = 0;

% Robot Parameters
R = 0.02;
L = 0.1;
% Counter
i = 1;
```

```
while ~readButton(mylego, 'up')
    resetRotation(Rightmotor);
    resetRotation(Leftmotor);
    % To change the time period
    pause(0.1)
    % Read the button from the beacon
    button = readBeaconButton(myirsensor,2);

    % Forward Motion
    if button == 1
        Rightmotor.Speed = 50;
        Leftmotor.Speed = 50;
        start(Rightmotor);
        start(Leftmotor);
    end

    % Read the Encoder values
    El = double(readRotation(Leftmotor));
    Er = double(readRotation(Rightmotor));

    % Calculate the required parameters
    Dl = 2*pi*R*(El/360);
    Dr = 2*pi*R*(Er/360);

    Dc = (Dr+Dl)/2;

    delta_theta=(Dr-Dl)/L;

    theta_array(i+1) = theta_array(i) + delta_theta;
    x_array(i+1) = x_array(i) + Dc*cos(theta_array(i) +
(delta_theta/2));
    y_array(i+1) = y_array(i) + Dc*sin(theta_array(i) +
(delta_theta/2));

    % Desired Trajectory for Linear Motion
    Er_d = double(readRotation(Rightmotor));
    El_d = Er_d;

    Dl_d = 2*pi*R*(El_d/360);
    Dr_d = 2*pi*R*(Er_d/360);

    Dc_d = (Dr_d+Dl_d)/2;
    delta_theta_d =(Dr_d-Dl_d)/L;

    theta_array_d(i+1) = theta_array_d(i) + delta_theta_d;
    x_array_d(i+1) = x_array_d(i) + Dc*cos(theta_array_d(i)
+ (delta_theta_d/2));
    y_array_d(i+1) = y_array_d(i) + Dc*sin(theta_array_d(i)
+ (delta_theta_d/2));

    i=i+1;
end

stop(Rightmotor);
stop(Leftmotor);

% Plot the results
figure
plot(x_array, y_array, 'b')
hold
plot(x_array_d, y_array_d, 'r')
legend('Calculated Odometry', 'Desired Trajectory')
```