Scientific Experimentation and Evaluation

Ananya Verma, Nandhini Shree Mathivanan, Rashid Ahamed Meeran Tajdeen

April 29, 2021

Contents

1 Motion Analysis of Differential Drive Robot

To analyze the locomotion error in the Differential drive LEGO EV3 robot by creating a probabilistic model that gives relation between the control and the observed behavior and comparing the actual pose of the robot to its expected pose.

1.1 Measurement Facility

The materials used to demonstrate the experiment are enlisted below:

- LEGO EV3 Kit
- A1 sheet with square grids of size 25mm
- Measuring tape (Least count : 1mm)
- Two pencil markers

1.1.1 Robot Design

- The top view of the LEGO EV3 differential drive robot can be seen in Figure 1. The entire robot is constructed using the LEGO EV3 kit.
- The basic assembly of the robot is done using the assembly manual as it was sturdy enough to prevent any wobbling along with some aesthetic additions.

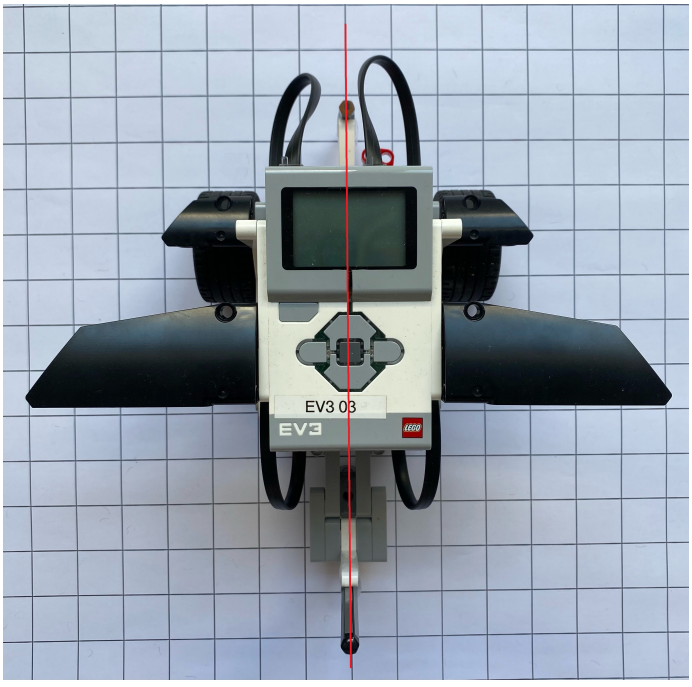

Figure 1: Top view of the robot

- The robot has a differential drive configuration with 2 independent motorised wheels at the front and a passive spherical wheel at the rear.

- The robot is stable as the centre of gravity lies within the area formed by the 3 points of contacts of the wheel with the floor.
- For measurement purposes, the robot is equipped with two markers. The idea to use two markers is to keep track of both the position and orientation of the robot. Given that the experiment has to be performed a number of times, this will help in placing the robot in the same initial position at all times. Both these markers are placed in the symmetrical axis (Figure 1) of the robot, perpendicular to the floor (Figure 2).

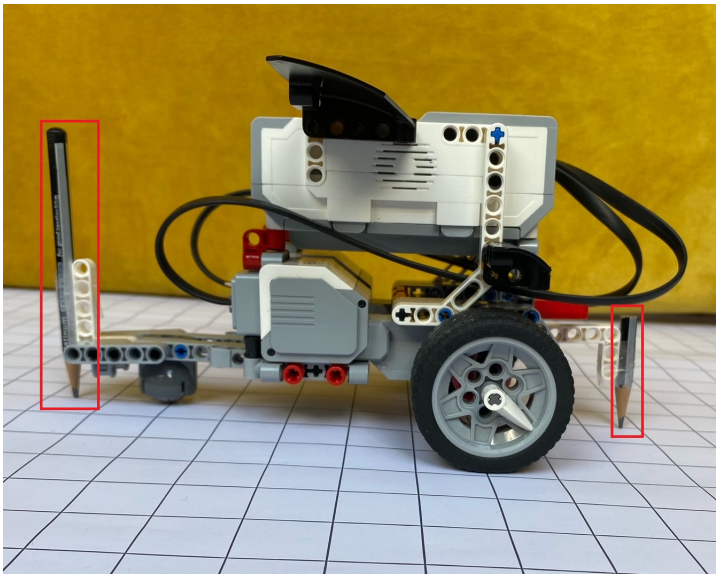

Figure 2: Side view of the robot

- One of the markers is placed at the front (Figure 3) and the other is placed at the rear (Figure 4). Here we have used a pencil as a marker. These two markers are placed parallel to each other so that the robot's position can be clearly estimated.

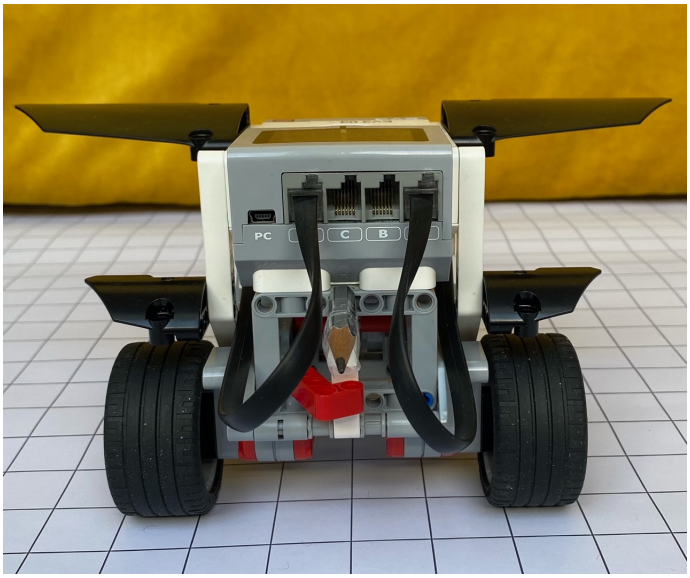

Figure 3: Front view of the robot

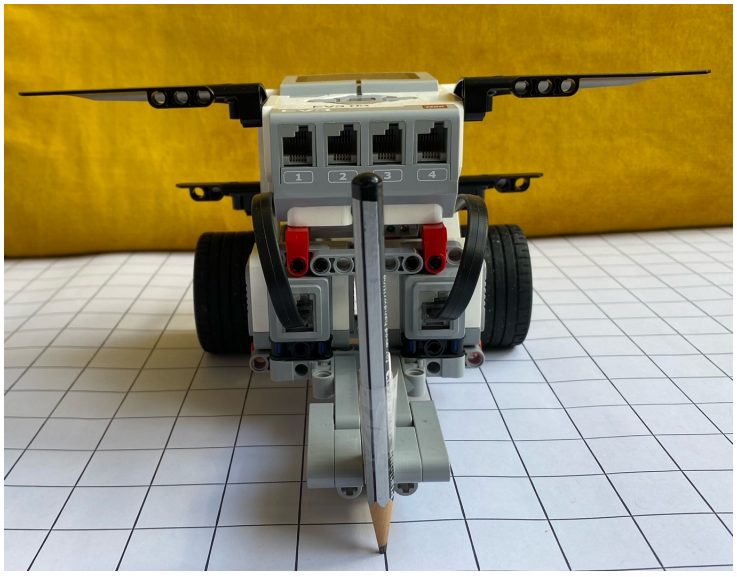

Figure 4: Rear view of the robot

- The distance between the 2 markers is 23.4 cm and the centre of the axle is at a distance of 6.7 cm from the front marker.
- Additionally, as the markers are only needed for placing the robot at the initial position and for marking the end position of the robot, they are placed in such a way that they can be moved up and down so that they do not interfere with the motion of the robot. This can be visualized in (Figure 5) for the first and the second marker respectively.
- The joint at the rear is pretty rigid that it holds upright by itself. But the joint in the front is freely moving, so an L-shaped lock (as marked in Figure 5) is attached to hold it upright .

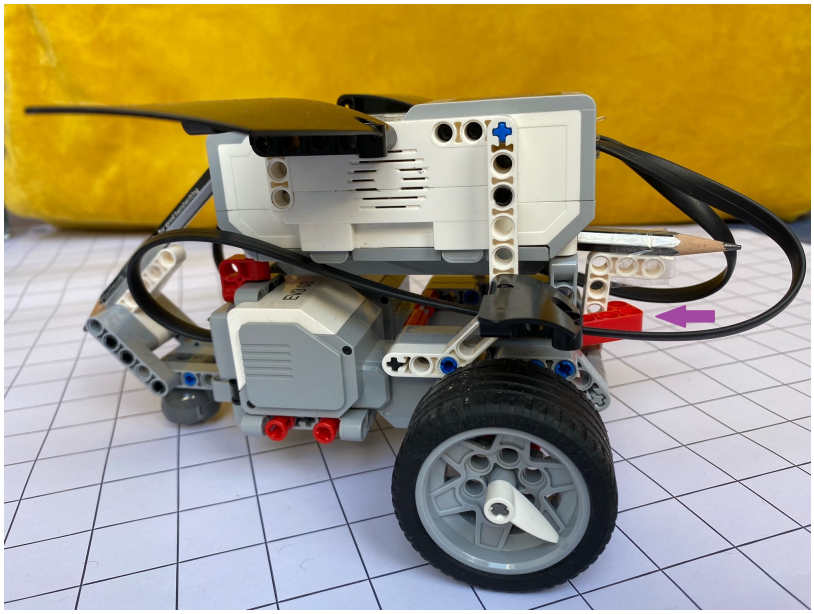

Figure 5: Upright position of the markers

- At the start of the experiment, the robot is placed at the initial position with the help of markers. The markers are lifted up when the robot is in motion. At the final position, the markers are put in the downward position to mark the final position of the robot.
- The LEGO brick is attached on top of the sturdy base in the middle ensuring
  - that the centre of gravity falls in the middle and
  - the weight distribution is even so that we could achieve highly stable motion.
- The start position of the each trial is fixed at the particular intersection of the grid. The robot is placed such that the front marker aligns with the intersection of the grid as explained above and the rear marker is at the Y axis of the intersection.
- A1 sheet is considered to be the working environment of the robot and the grids are used for the purpose of measuring the position of the robot easily. In addition to that, measuring tape is used for the more precise measurements.
- Additional steering mechanisms are avoided as we are using the differential drive mechanisms where the direction of robot's motion is controlled by the rotation of the front wheels.
- The measurand that are used are $[x, y, \theta]^\top$ where $[x, y]^\top$ are the robot translation along x axis and y axis and θ is defined as the rotation of the robot in degrees.

1.1.2 Expected problem and rough estimation of expected precision

- The measurement error is caused by two factors one is due to the measurement tape where the least count is taken as 1mm and the other one is grid marking error and so it may have the tolerance upto $\pm 1mm$.
- When the buttons on the robot's controller is pressed then it may encounter some pressure or tend to move which may cause error.

- Parallax error can be observed when the reading is not measured correctly or may be due to the improper positioning of the cable.
- Slippage of wheel occurs when there is a command delay between the motors, improper alignment of wheels, friction between wheels and the paper etc.
- Since the markers present in the robot tend to move up and down each time there will be errors in the marker's position and this frequent movement of these joints can cause wear and tear.
- Some of the observed errors include reduction in the efficiency of the robot due to the low battery, loose connection between motor and the controller, paper stiffness etc.
- Experiments should be performed in such a way to avoid or minimise all the above mentioned errors.
- Rough estimation of expected precision
  - Measurement error - $\pm 1mm$
  - Grid error - $\pm 1mm$
  - Error due to pressing of buttons - $\pm 3mm$
  - Parallax error - $\pm 2mm$
  - Wheel slippage error - $\pm 5mm$
  - Marker mechanism error - $\pm 3mm$
  - Hence, the Rough estimation of total error - $\pm 15mm$

1.1.3 Error Propagation

Python program for the calculation of Error Propagation

```
import numpy as np
import sympy as sp
from IPython.display import display

def my_jac():
    theta, x1, y1, x2, y2 = sp.symbols('theta, x1, y1, x2, y2')
    unknowns = sp.Matrix([x1, x2, y1, y2]) # x1, x2, y1 and y2 are in cm
    angle = sp.atan2((y2-y1),(x2-x1))
    theta = sp.Matrix([angle]) # theta is in radian
    myjac = theta.jacobian(unknowns)
    display(myjac)
    return myjac

def upper_bound(myjac, x_1, y_1, x_2, y_2):
    theta, x1, y1, x2, y2 = sp.symbols('theta, x1, y1, x2, y2')
    delta = np.array([-0.1, 0.1, 0.1, -0.1])
    jac_eval = myjac.subs({x1:x_1, x2:y_1, y1:x_2, y2:y_2})
    error = float(np.dot(np.array(jac_eval), delta))
    error = round(error,4)
    return error

def lower_bound(myjac, x_1, y_1, x_2, y_2):
    theta, x1, y1, x2, y2 = sp.symbols('theta, x1, y1, x2, y2')
    delta = np.array([0.1, -0.1, -0.1, 0.1])
```

```
        jac_eval = myjac.subs({x1:x_1, x2:y_1, y1:x_2, y2:y_2})
        error = float(np.dot(np.array(jac_eval), delta))
        error = round(error,4)
        return error

myjac = my_jac()
```

- Even though we do the experiment in a static environment with every parameters fixed, some measurement errors can be observed due to the least count of the measuring equipment.
- This error propagation can be modeled using jacobians as shown below.

$$\theta = tan^{-1}(\frac{y_2 - y_1}{x_2 - x_1}) \tag{1}$$

$$d\theta = Jac(\theta) * \begin{bmatrix} dx_1 \\ dx_2 \\ dx_3 \\ dx_4 \end{bmatrix} \tag{2}$$

$$d\theta = \begin{bmatrix} \frac{(y_2-y_1)}{(x_2-x_1)^2+(y_2-y_1)^2} & \frac{(y_1-y_2)}{(x_2-x_1)^2+(y_2-y_1)^2} & \frac{(x_1-x_2)}{(x_2-x_1)^2+(y_2-y_1)^2} & \frac{(x_2-x_1)}{(x_2-x_1)^2+(y_2-y_1)^2} \end{bmatrix} * \begin{bmatrix} dx_1 \\ dx_2 \\ dx_3 \\ dx_4 \end{bmatrix}$$

$$d\theta = \frac{(y_2 - y_1) * dx_1}{(x_2 - x_1)^2 + (y_2 - y_1)^2} + \frac{(y_1 - y_2) * dx_2}{(x_2 - x_1)^2 + (y_2 - y_1)^2} + \frac{(x_1 - x_2) * dy_1}{(x_2 - x_1)^2 + (y_2 - y_1)^2} + \frac{(x_2 - x_1) * dy_2}{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

$$d\theta = \frac{(y_1 - y_2) * (dx_2 - dx_1)}{(x_2 - x_1)^2 + (y_2 - y_1)^2} + \frac{(x_1 - x_2) * (dy_1 - dy_2)}{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

$$d\theta = \frac{((y_1 - y_2) * (dx_2 - dx_1)) + ((x_1 - x_2) * (dy_1 - dy_2))}{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{3}$$

  - Here $x_1, x_2, y_1, y_2, dx_1, dx_2, dx_3$ and dx_4 are in cm whereas θ and $d\theta$ are in radian.
  - The least count of the measuring equipment is 1 mm, which means that the measuring error can be ± 1 on the values dx_1, dx_2, dy_1, dy_2. This results in 16 possible combinations. The upperbound can be calculated by taking the values $dx_1 = -0.1, dx_2 = 0.1, dy_1 = 0.1, dy_2 = -0.1$ and the lowebound can be found by taking the values $dx_1 = 0.1, dx_2 = -0.1, dy_1 = -0.1, dy_2 = 0.1$

$$d\theta_{UpperBound} = 0.2 \left(\frac{(y_1 - y_2) + (x_1 - x_2)}{(x_2 - x_1)^2 + (y_2 - y_1)^2} \right) \tag{4}$$

$$d\theta_{LowerBound} = -0.2 \left(\frac{(y_1 - y_2) + (x_1 - x_2)}{(x_2 - x_1)^2 + (y_2 - y_1)^2} \right) \tag{5}$$

1.2 Programs and Parameters

```
from time import sleep
import sys
import math
from ev3dev2.motor import LargeMotor, OUTPUT_A, OUTPUT_B, OUTPUT_C,
OUTPUT_D, SpeedPercent,
MoveTank
```

```
from ev3dev2.sensor import INPUT_1, INPUT_2, INPUT_3, INPUT_4
from ev3dev2.sound import Sound
from ev3dev2.button import Button

WHEEL_DIAMETER = 5.6 # cm
MAIN_AXIS_LENGTH = 12.0 # cm

buttons = Button()
move = MoveTank(OUTPUT_A, OUTPUT_D)
spkr = Sound()
motor_1 = LargeMotor(OUTPUT_A)
motor_2 = LargeMotor(OUTPUT_D)

motor_1_path = [] # in rad
motor_2_path = [] # in rad

motor_1_path.append(motor_1.position)
motor_2_path.append(motor_2.position)

robot_orientation = 0.0 # in rad
robot_position_x  = 0.0 # in cm
robot_position_y  = 0.0 # in cm

distance_traveled_wheel_1 = 0.0 # in cm
distance_traveled_wheel_2 = 0.0 # in cm

spkr.speak('Press a button')
while True:
    if buttons.left:
        move.on_for_seconds(SpeedPercent(30), SpeedPercent(40), 2.2, block=False)

    elif buttons.up:
        move.on_for_seconds(SpeedPercent(40), SpeedPercent(40), 2.2, block=False)

    elif buttons.right:
        move.on_for_seconds(SpeedPercent(40), SpeedPercent(30), 2.2, block=False)

    if (motor_1.is_running): motor_1_path.append((motor_1.position * math.pi)
    / 180.0)
    if (motor_2.is_running): motor_2_path.append((motor_2.position * math.pi)
    / 180.0)

    if (motor_1.is_holding and motor_2.is_holding):
        data_length = min(len(motor_1_path), len(motor_2_path))

        with open('both_motors_path.csv', "a") as f_wheels_path:
            for i in range(data_length):
                f_wheels_path.write(str(str(motor_1_path[i]) + " "
                + str(motor_2_path[i]) + "\n")) # in rad

        with open('robot_path.csv', "a ") as f_robot_path:
```

```
            for i in range(data_length):
                if i is 0:
                    distance_traveled_wheel_1 = (WHEEL_DIAMETER * math.pi *
                    motor_1_path[0]) / (2 * math.pi)
                    distance_traveled_wheel_2 = (WHEEL_DIAMETER * math.pi *
                    motor_2_path[0]) / (2 * math.pi)
                else:
                    distance_traveled_wheel_1 = (WHEEL_DIAMETER * math.pi *
                    (motor_1_path[i] - motor_1_path[i - 1])) / (2 * math.pi)
                    distance_traveled_wheel_2 = (WHEEL_DIAMETER * math.pi *
                    (motor_2_path[i] - motor_2_path[i - 1])) / (2 * math.pi)

                delta_distance = (distance_traveled_wheel_1 + distance_traveled_wheel_2) / 2
                delta_angle    = (distance_traveled_wheel_1 - distance_traveled_wheel_2)
                / MAIN_AXIS_LENGTH

                robot_orientation = robot_orientation + delta_angle
                robot_position_x  = robot_position_x  + delta_distance *
                math.sin(robot_orientation)
                robot_position_y  = robot_position_y  + delta_distance *
                math.cos(robot_orientation)

                f_robot_path.write(str(robot_position_x) + " " + str(robot_position_y)
                + " " + str((math.pi / 2) - robot_orientation) + "\n") # in cm and rad

        spkr.speak('Motion completed')
        break

    # don't let this loop use 100% CPU
    sleep(0.001)
```

1.2.1 Experiment

- The start pose of the robot is taken as (0,0) which is the position of the first marker placed at the front, while the second marker placed at the rear side of the robot is at (0,-27). The markings are in cm.
- At the end of the experiment, the marker positions will be at (x_1, y_1) and (x_2, y_2) for the first and second marker respectively. The center of the wheel axis of the robot is marked as (x,y). The length of the robot is 22 cm. From the second marker (x_2, y_2) to the center (x,y) the length is 16.7 cm, while the length from the first marker (x_1, y_1) to the center (x,y) is 5.3 cm.

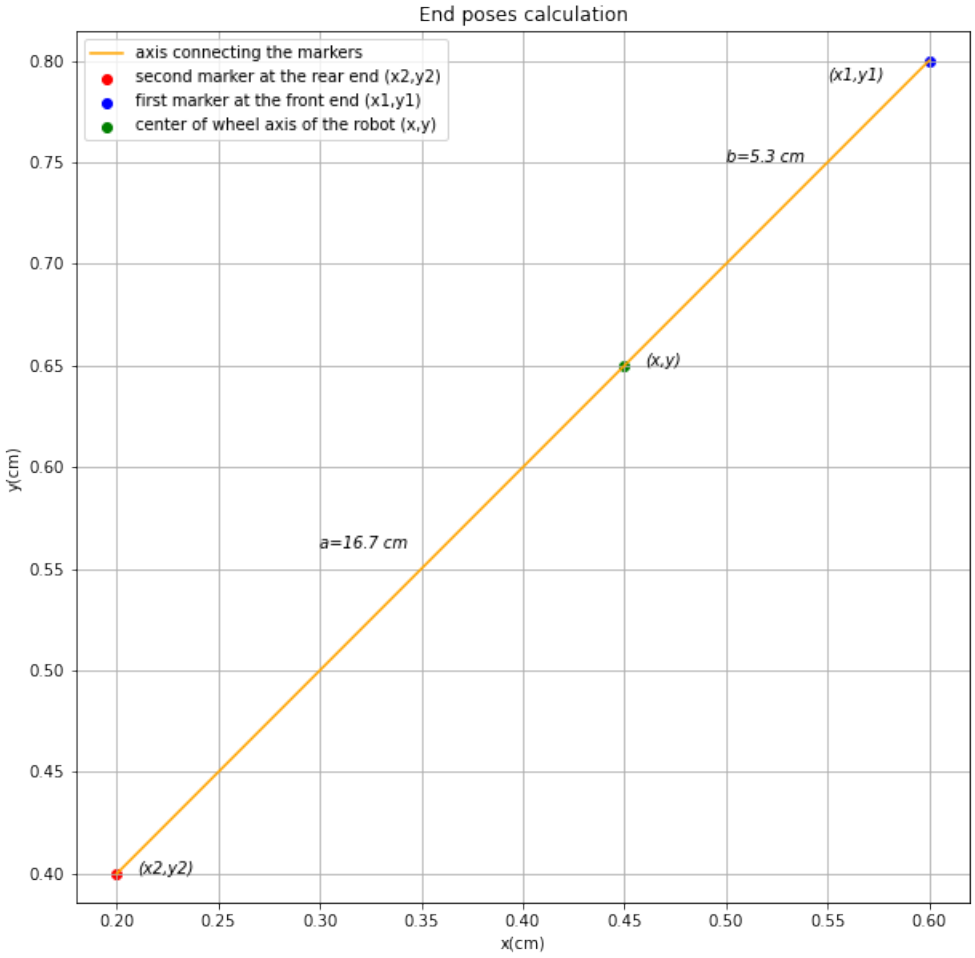

Figure 6: End poses calculation

- The formulas to calculate the value of x and y coordinates of the center of the wheel axis are:

$$x = x_2 + \frac{a}{a+b}(x_1 - x_2)$$

$$y = y_2 + \frac{a}{a+b}(y_1 - y_2)$$

- Substituting the values in the above equation will yield the below mentioned formula:

$$x = x_2 + \frac{16.7}{22}(x_1 - x_2)$$

$$y = y_2 + \frac{16.7}{22}(y_1 - y_2)$$

- Once the values of x and y are calculated, the angle θ can be calculated using the formula

$$\theta = \tan^{-} 1(\frac{y_1 - y_2}{x_1 - x_2}) \tag{2}$$

- Below is the code written to explain the above theoretical explanation

```
import numpy as np
import matplotlib.pyplot as plt
def find_robot_pose(reading):
    pose = np.array([[0,-5.3, i[2]+((i[0]-i[2])*(16.7/22)),
    i[3]+((i[1]-i[3])*(16.7/22))]
    for i in reading])
    print(pose)
    return pose

def find_translation(pose):
    translation = np.array([[i[2]-i[0],i[3]-i[1]] for i in pose])
    print(translation)

    return translation

def find_orientation(reading):
    orientation = np.array([[np.arctan2((i[1]-i[3]),(i[0]-i[2]))]
    for i in reading])
    print(orientation)
    return orientation

reading_front = np.loadtxt('SEE front reading.csv')
reading_left = np.loadtxt('SEE left reading.csv')
reading_right = np.loadtxt('SEE right reading.csv')

pose_front = find_robot_pose(reading_front)
pose_left = find_robot_pose(reading_left)
pose_right = find_robot_pose(reading_right)

translation_front = find_translation(pose_front)
translation_left = find_translation(pose_left)
translation_right = find_translation(pose_right)

orientation_front = find_orientation(reading_front)
orientation_left = find_orientation(reading_left)
orientation_right = find_orientation(reading_right)
```

- The experiment is done by carrying out three types of motions namely straight run, arc to the left and arc to the right of the robot. This experiment is carried out 20 times for each setting. The observation table for the straight, left and right run are given in the Table 1,, 2 and 3 simultaneously.

- Errors are expected to occur while lowering the markers or parallex error, yet we have taken sufficient care to limit these errors.

- The readings are plotted and presented in the figures 7,while the individual readings for each motion are presented in the figure 8, 9 and 10.

- The graphs for the end position of the robot and for the end poses because of the right side arc, left side arc and due to the straight motion are plotted using the code given below:

```
import matplotlib.pyplot as plt

# for the end poses of the robot

plt.figure(figsize=(10,12))
plt.title("End poses of the robot")
plt.ylabel('y(cm)')
plt.xlabel('x(cm)')
plt.xticks([-20,-15,-10,-5,0,5,10,15,20])
plt.yticks([-5,0,5,10,15,20,25,30,35,40,45,50,55])
plt.axis('equal')
plt.grid(True)

plt.scatter(0,0,color = 'black',label='Start Pose')
plt.scatter(translation_front[:,0],translation_front[:,1],s=6,color='red',
label='End Pose of straight run')
plt.scatter(translation_left[:,0],translation_left[:,1],s=6,color='green',
label='End Pose of left turn')
plt.scatter(translation_right[:,0],translation_right[:,1],s=6,color='orange',
label='End Pose of right turn')

for i in range(len(reading_front)):
    plt.arrow(translation_front[i,0],translation_front[i,1],
    3*np.cos(orientation_front[i,0]),
    3*np.sin(orientation_front[i,0]),lw=0.1,color='red')
for i in range(len(reading_left)):
    plt.arrow(translation_left[i,0],translation_left[i,1],
    3*np.cos(orientation_left[i,0]),
    3*np.sin(orientation_left[i,0]),lw=0.1,color='green')
for i in range(len(reading_right)):
    plt.arrow(translation_right[i,0],translation_right[i,1],
    3*np.cos(orientation_right[i,0]),
    3*np.sin(orientation_right[i,0]),lw=0.1,color='orange')

plt.legend()
plt.savefig('end poses.png')
plt.show()

# straight end poses of the robot

plt.figure(figsize=(10,12))
plt.title("Straight end poses of the robot")
plt.ylabel('y(cm)')
plt.xlabel('x(cm)')
plt.axis('equal')
plt.grid(True)
plt.scatter(translation_front[:,0],translation_front[:,1],s=10,color='red',
label='End Pose of straight run')
for i in range(len(reading_front)):
```

```
        plt.arrow(translation_front[i,0],translation_front[i,1],
        0.5*np.cos(orientation_front[i,0]),
        0.5*np.sin(orientation_front[i,0]),lw=0.3,color='red')

    plt.legend()
    plt.savefig('Straight end poses.png')
    plt.show()

    # left end poses of the robot

    plt.figure(figsize=(10,12))
    plt.title("Left end poses of the robot")
    plt.ylabel('y(cm)')
    plt.xlabel('x(cm)')
    plt.axis('equal')
    plt.grid(True)
    plt.scatter(translation_left[:,0],translation_left[:,1],s=10,color='green',
    label='End Pose of left turn')
    for i in range(len(reading_left)):
        plt.arrow(translation_left[i,0],translation_left[i,1],
        0.5*np.cos(orientation_left[i,0]),
        0.5*np.sin(orientation_left[i,0]),lw=0.3,color='green')

    plt.legend()
    plt.savefig('Left end poses.png')
    plt.show()

    # right end poses of the robot

    plt.figure(figsize=(10,12))
    plt.title("Right end poses of the robot")
    plt.ylabel('y(cm)')
    plt.xlabel('x(cm)')
    plt.axis('equal')
    plt.grid(True)
    plt.scatter(translation_right[:,0],translation_right[:,1],s=10,color='orange',
    label='End Pose of right turn')
    for i in range(len(reading_right)):
        plt.arrow(translation_right[i,0],translation_right[i,1],
        0.5*np.cos(orientation_right[i,0]),
        0.5*np.sin(orientation_right[i,0]),lw=0.3,color='orange')

    plt.legend()
    plt.savefig('Right end poses.png')
    plt.show()
```

would be nice to include images and observations made while carrying out the experiment.

see 'Deliverables 1.2', point no: 2 from the SEE manual for Practical Experiments.

1.2.2 Observed Data

- The reading for all the three motions of the robot that is straight run, left run and right run are provided in the table 1, 2 and 3 respectively below.

Straight run			
Sr. no	**x(cm)**	**y(cm)**	**angle(degree)**
1	-0.37227273	46.52409091	1.55709855
2	0.50363636	46.7	1.55261651
3	-0.09636364	46.1	1.55261651
4	0.05545455	46.1	1.54353036
5	0.15545455	46.1	1.54353036
6	-0.27227273	46.42409091	1.55709855
7	0.07954545	46.3	1.54807297
8	0.17954545	46.2	1.54807297
9	0.27954545	45.77590909	1.54817575
10	0.05181818	46.85181818	1.56178756
11	0.12772727	45.8	1.55716081
12	0.27954545	46.6	1.54807297
13	-0.34818182	45.87590909	1.5617468
14	-0.24818182	46.5	1.56170567
15	0.10363636	46.0	1.55261651
16	0.05181818	45.87590909	1.5617468
17	0.42772727	46.7	1.55716081
18	0.42772727	45.57590909	1.5572225
19	0.55181818	45.9	1.56170567
20	-0.04818182	46.5	1.56170567

Table 1: End poses reading for the straight run

Left run			
Sr. no	**x(cm)**	**y(cm)**	**angle(degree)**
1	-16.26	34.92363636	2.48370405
2	-17.18772727	34.81636364	2.52081491
3	-17.18772727	34.51636364	2.52081491
4	-16.48409091	34.76818182	2.50267867
5	-16.26	35.32	2.49809154
5	-17.21181818	34.49227273	2.51446861
6	-16.93590909	34.94409091	2.50444932
7	-16.26	34.74772727	2.48727141
8	-17.01181818	34.84409091	2.50714107
9	-17.03590909	35.67181818	2.49358707
10	-16.93590909	34.94409091	2.50444932
11	-17.01181818	34.56818182	2.51079504
12	-16.66	35.29590909	2.49446509
13	-17.18772727	34.56818182	2.51346104
14	-16.46	34.48863636	2.52403568
15	-17.06727273	35.0	2.51928978
16	-16.40818182	34.54772727	2.48174423
17	-16.43227273	34.54772727	2.47895077
19	-17.18772727	35.09227273	2.51712823
20	-16.96	35.29590909	2.49446509

Table 2: End poses reading for the left run

Right run			
Sr. no	**x(cm)**	**y(cm)**	**angle(degree)**
1	16.96363636	35.04045455	0.6144448
2	17.53954545	34.81272727	0.60064416
3	18.095	34.30909091	0.57052623
4	17.06363636	34.58863636	0.60698766
5	17.33954545	34.26090909	0.59308515
5	17.71909091	33.80909091	0.57296614
6	17.69136364	34.51272727	0.59553069
7	17.69136364	34.285	0.58420995
8	17.39136364	34.61272727	0.59553069
9	17.06363636	34.31272727	0.60322985
10	17.71545455	34.23681818	0.59431565
11	17.79136364	34.20909091	0.58039819
12	17.23954545	33.71272727	0.60064416
13	17.76727273	33.76090909	0.58549005
14	18.24681818	34.30909091	0.56570147
15	18.27090909	33.90909091	0.56810473
16	17.59136364	34.16090909	0.5880026
17	17.11181818	33.78863636	0.61223255
19	18.11909091	33.90909091	0.57296614
20	17.94318182	33.76090909	0.58299639

Table 3: End poses reading for the right run

Missing visualisation of :
① Motion path of robo using encoder data
② Motion path of robo using encoder data v/s that using manual measurement

1.2.3 Visualization of the end poses

- Multiple readings for each pose resembles a cluster which proves that the readings are precise.

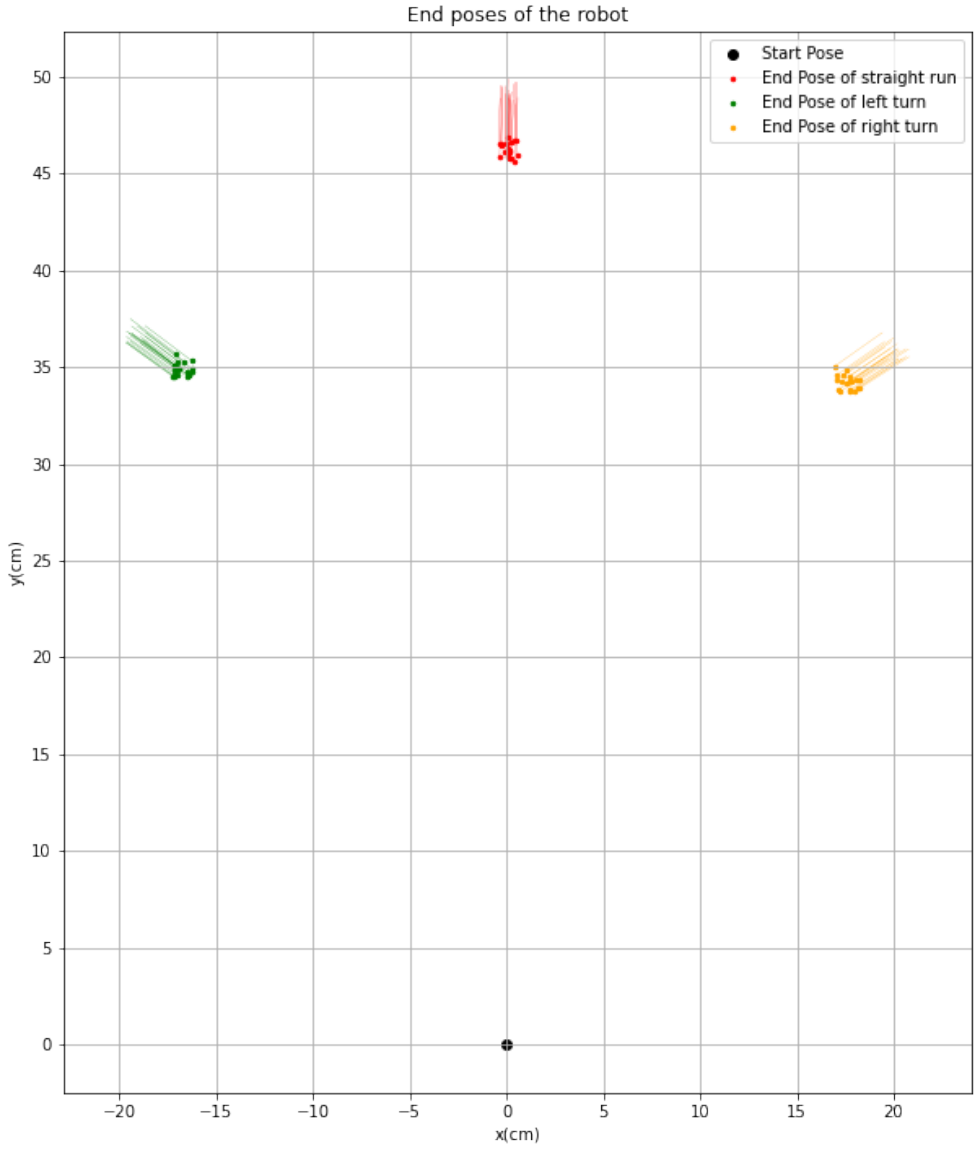

Figure 7: End poses of the robot as a cluster

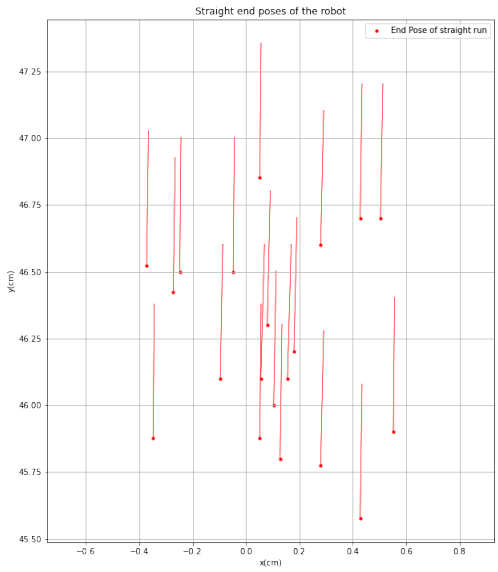

Figure 8: Straight end poses of the robot as a cluster

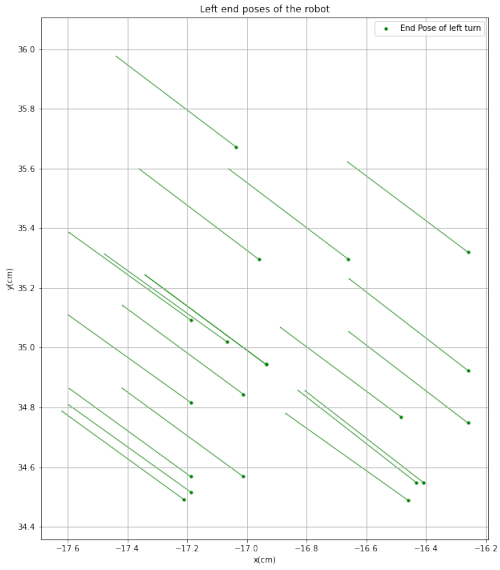

Figure 9: Left End poses of the robot as a cluster

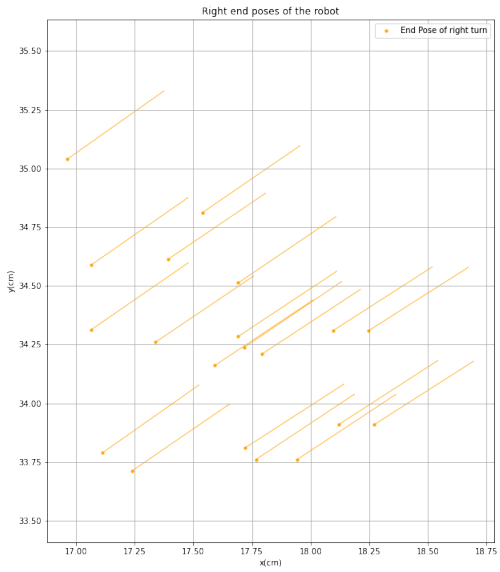

Figure 10: Right end poses of the robot

1.2.4 Data Visualization and Comparison

- For checking the correctness of our data, we have plotted our data with the data collected from the other groups.
- The figure 11 clearly shows that our data is well aligned with the data provided by others.

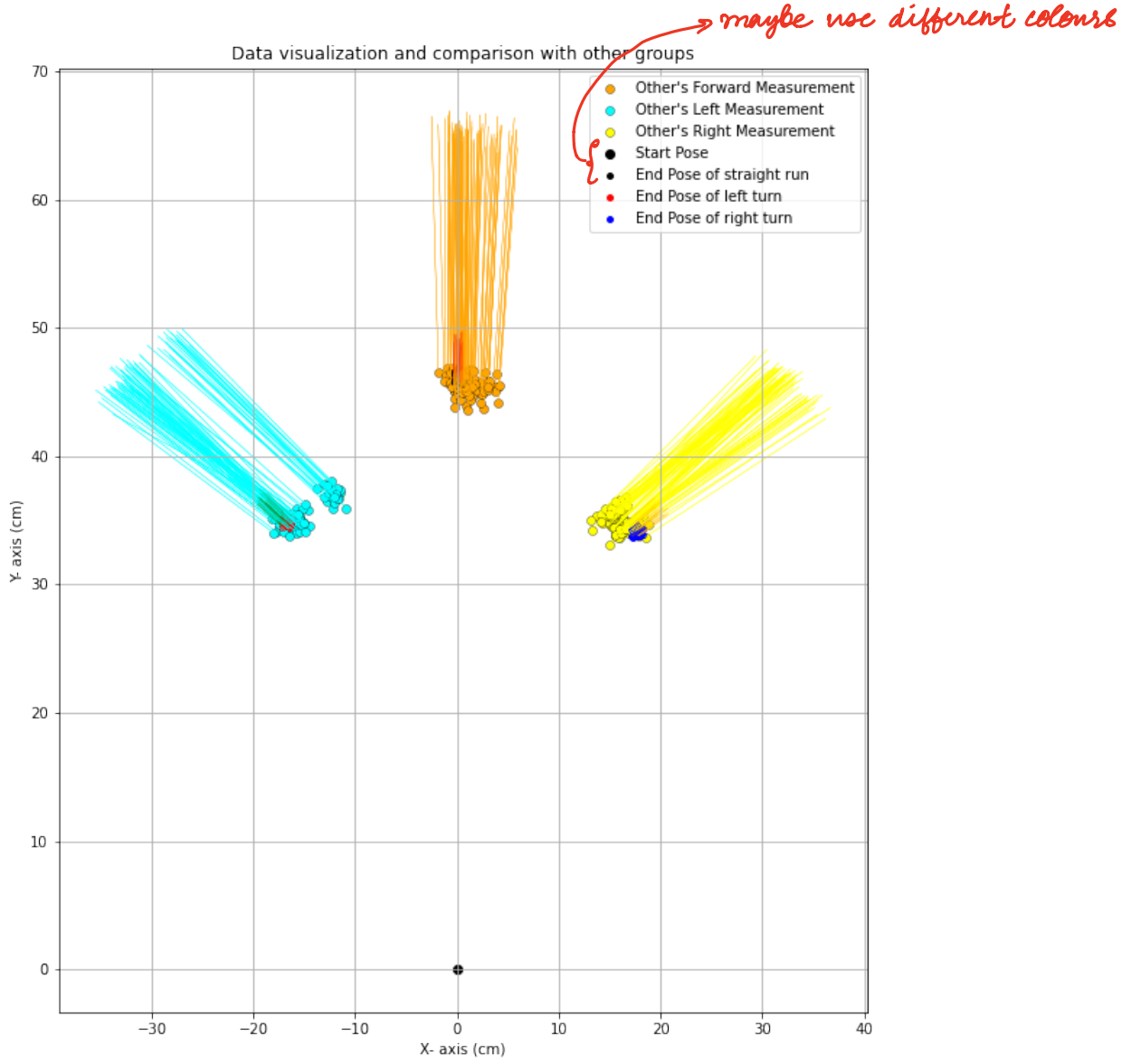

Figure 11: Data visualization and comparison with other groups