Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen International Center for Information Technology

Scientific Experimentation & Evaluation

Simon Deussen, Malika Navaratna, Shalaka Satheesh

Submitted to Hochschule Bonn-Rhein-Sieg,
Department of Computer Science
in partial fullfilment of the requirements for the degree
of Master of Science in Autonomous Systems

April 2021

Contents

1

Task 1.1

1.1 Deliverables 1.1

Write a report detailing your envisaged experimental setup, expected problems and expected performance. In your report, use terminology from the lecture (i.e. measurement system, measurand, measured quantity, and so forth) to describe your experiment. Your report should cover:

1. The relevant aspects of the design of the robot, especially how you mark the stop position and how you ensure identical start positions.
2. An estimate of the expected precision of the to-be-observed data (i.e. the measurement process), including how you arrived at these estimates and why they are plausible.
3. An estimate of the propagated orientation's uncertainty (including the upper and lower bounds) caused by the errors in the measurement process, using the method of Jacobian error propagation.

1.2 Design of Robot

- Our robot, 'Rosa' is designed based on the base design provided in the **Lego Mindstorms EV3**(evolution 3) kit (manual). Figure 1.1 shows our final design, with the measurement facility for the robot intact.
- The following are the materials used for building Rosa and hence represents the **measurement facility**:
    1. Lego EV3 kit
    2. Focusable Laser Module (2 units): It uses a working voltage of 3.0 - 5.0 V. It is powered from the USB port from the EV3 controller.
    3. Bread Board: for making connections between the laser module and the USB port
    4. Jumpers for powering the laser modules
- The robot is three-wheeled with a differential drive configuration. In this configuration (Figure 1.3, 1.4), two of the wheels placed in front of the robot on either sides, have independent actuators. The

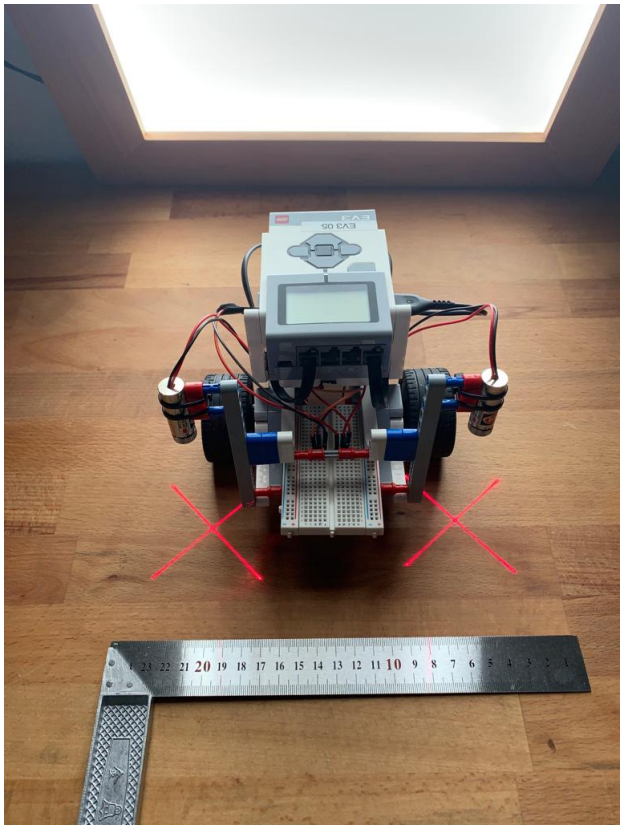


Figure 1.1: Rosa - Front View

last wheel in the back, is a castor wheel which is non-driven placed for increasing the stability of the bot.

- Commands for forward, right and left motions are made from the EV3 controller which is placed on top of the base of the bot.
- The measurements of the bot's motions are made on top a 96.8 × 68 cm sized grid sheet (grid size ≈ 2.5 cm).
- The measurement is done by using two focussable **laser modules** with a resolution of 1mm placed in the front of the robot. The lasers are mounted onto the lego blocks by using zip ties and tape.
- The two cross lasers provide precise marking of the pose of the robot. The aperture is 9.1 cm above ground, and 5.9 cm in front of the main axis. The distance between the two lasers is 14.8 cm.
- Since the lasers are working with 5V we are able to use the robots USB port for power supply, which makes any additional batteries unnecessary. We use a breadboard for making connections between the laser module and the USB port.

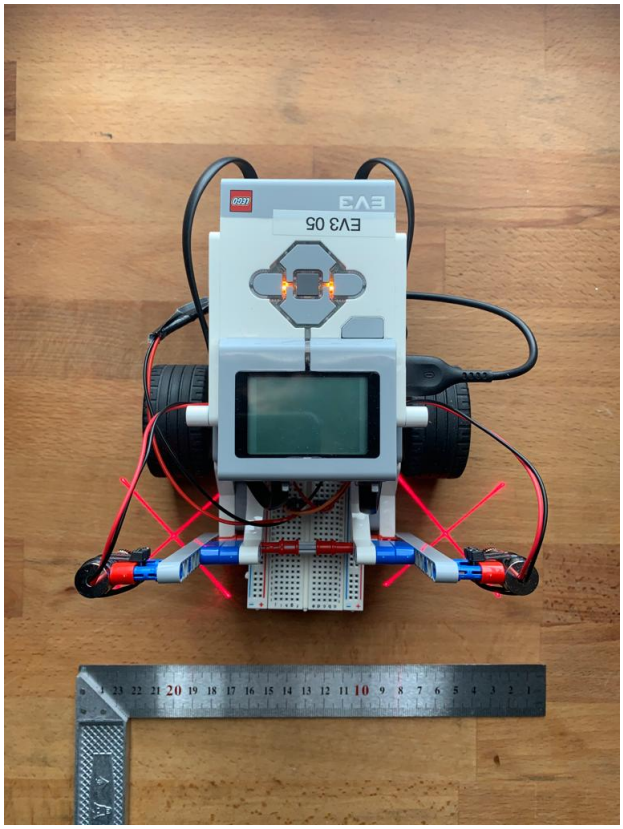


Figure 1.2: Rosa - Top View

- The **pose** of the robot, which is the **device under test (DUT)**, is described by the vector $[x, y, \theta]^T$.
- Here, the **measurand** is $[x, y]^T$ and it represents the **translation** of the robot which is measured in **millimeters**. The angle, θ represents the robot's **orientation** which is the **measurement result** obtained from the measured translation, in **degrees**.
- The Figure 1.6, 1.7 represents the distances between the two cross lasers and the center of the robot.
- Further explanations about the measurement facility is provided in section 1.3 Measurement Process
- **Ensuring identical start positions**:
  1. Securing the 96.8 × 68 cm sized grid sheet to a wooden plank with tape to prevent errors from changes in it's position.
  2. Marking the start points with the help of laser crosses of the resolution 1mm.
- **Marking stop positions**:

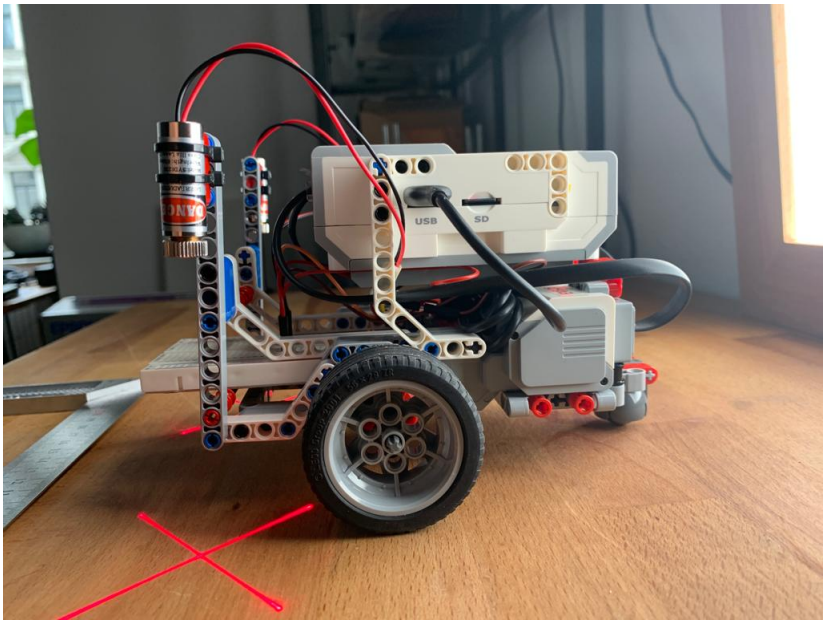


Figure 1.3: Rosa - Side View

1. The stop positions are marked with the help of the laser crosses, rulers and pencils on the grid. The mid point of the laser cross is marked with a pencil and the distance between the starting and end positions are measured using a ruler.

1.3 Measurement Process

- The cross lasers allow us to mark easily positions on the supplied grid paper.

1. The robot gets put onto the start position. For this we are using the lasers to position the robot exactly on its start position marks. For a faster positioning, we use a wooden plank on the back of the start position for guidance.
2. We start the robot's program and let it run.
3. Using a ruler we make small cross marks at both laser positions. Each cross marks get a number for later data survey.
4. The process (2 and 3) is repeated until enough data is gathered.
5. Now the cross marks needs to be measured. For this, we use a two step measurement process. We count the number of grids until the grid with the marked cross. Then, the mid point of the laser cross is marked with a pencil and the distance between the starting and end positions are measured using a ruler.
6. Each measurement is made by keeping in reference, the global coordinate system which is positioned at the centre of the axis connecting the wheel base.

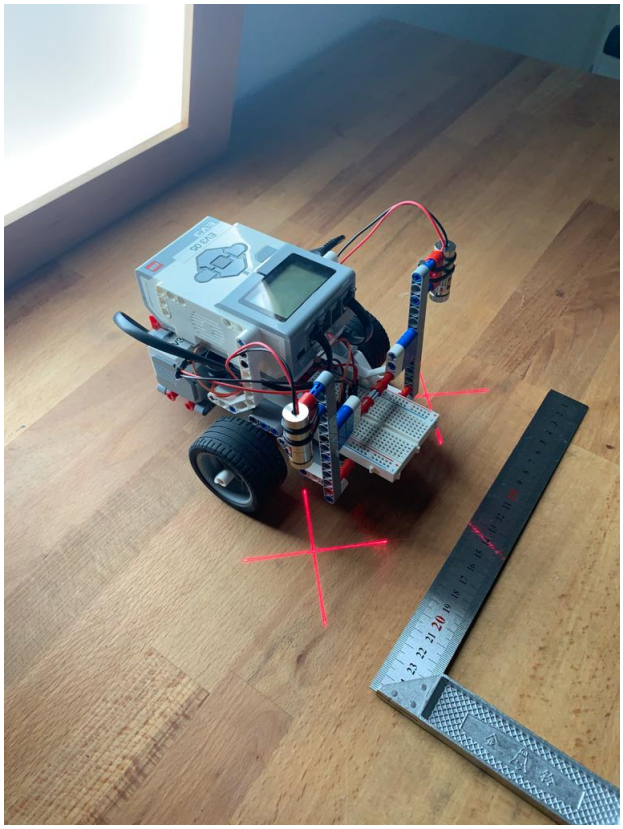


Figure 1.4: Rosa - Another Side View

- We define the coordinate system of the robot as following:

- The origin of the robot will lie on the mid point of the axial line connecting the wheels as seen on Figure 1.8

- The start position will always be set by aligning the left laser marker on a pre-marked position on the grid. The global coordinate system will also originate from this point, even when measuring the laser crosses for translation and orientation change.

- The global coordinate system originate at the initial position of the mid point of the axis connecting the robot's wheels(initial position of the robot's origin). Since the magnitude change of x and y (Δx & Δy) will be the approximately same for the two markers and the origin of the robot, to take the distance travelled by the robot origin we take the average change in x and y of the two markers. This distance will also be the x and y values with respect to the global coordinate system. The change in orientation will be found using the direction vector connecting the front and rear markers . See Figure 1.9 and equation 1.1 for details.

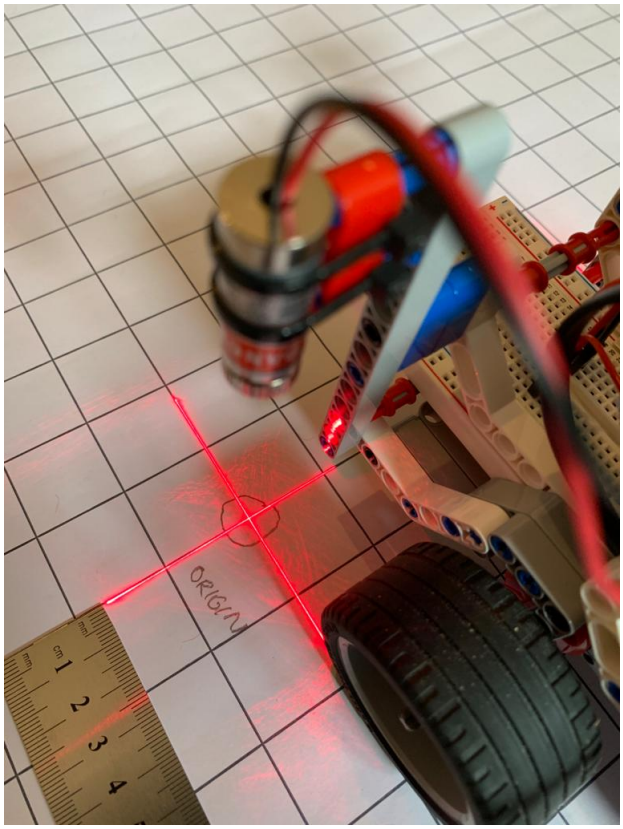


Figure 1.5: Rosa on the grid sheet

1.3.1 Estimate of Error

- The robot's distance measurements will be subjected to many errors. Some of them are listed below along with the rough estimate of their precision:

1. Error when marking the laser cross due to the thickness of the laser crosses: $\pm 2mm$
2. Parallax error when measuring the laser cross with the ruler used for our measurements: $\pm 1mm$
3. Pressing the button pushes the robot from one position and can put extra weight on one side of the wheel: $\pm 4mm$
4. Slippage: during multiple runs the laser markers can slightly change position: $\pm 1mm$
5. If the grid is not held tightly there will be folds and the surface will not be flat: $\pm 1mm$
6. Jerk: The sudden deceleration of the robot can overshoot the robot past its position specially if the momentum of the controller. This will not be same in every run.: $\pm 2mm$ (not a measurement error)

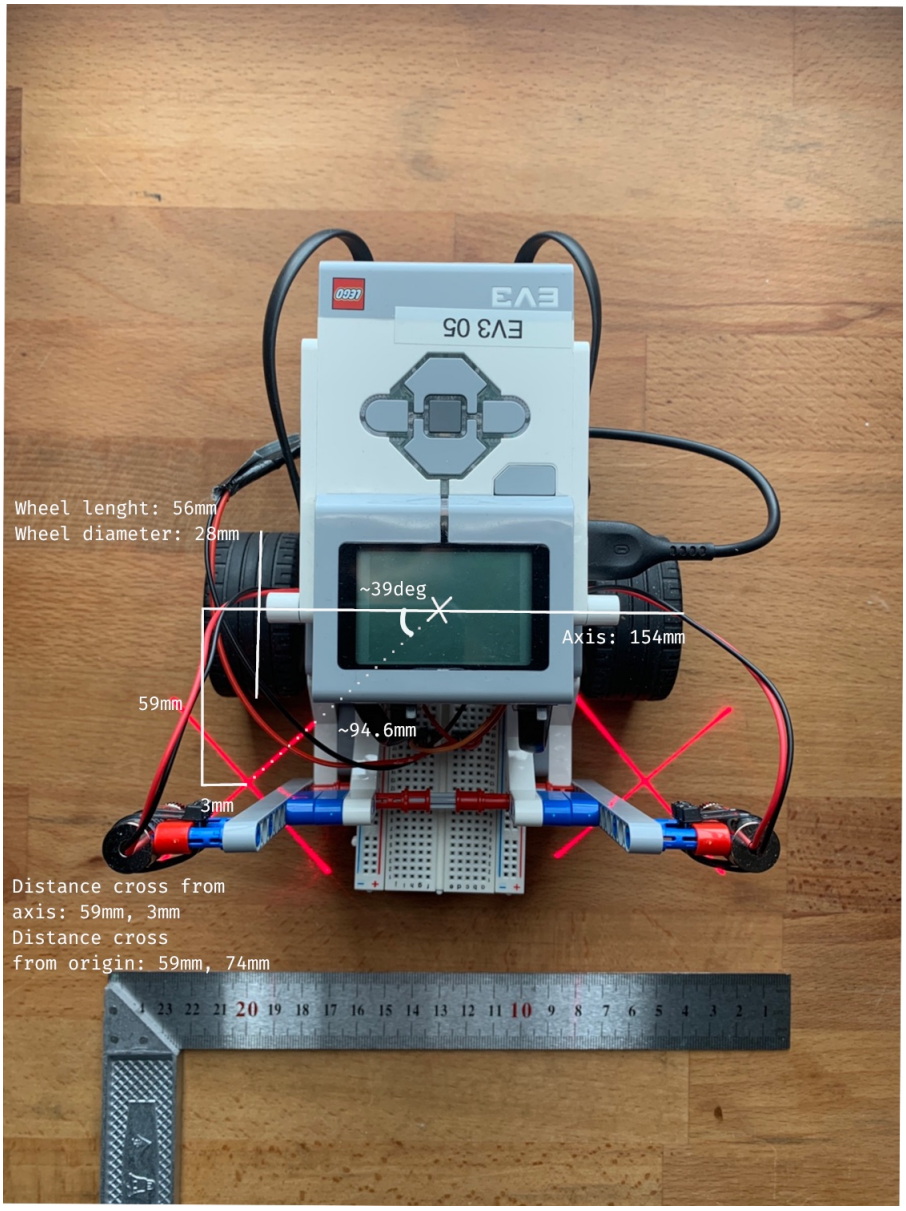


Figure 1.6: Rosa with the measurement facility

1.4 Error Propagation

- The errors mentioned above will propagate when making the distance measurements. This can be easily calculated using a Jacobian.

- Since there are 4 variables namely x_1, x_2, y_1, y_2 (refer Figure 1.9 for naming convention), the error associated to each of them contribute to the propagated error.

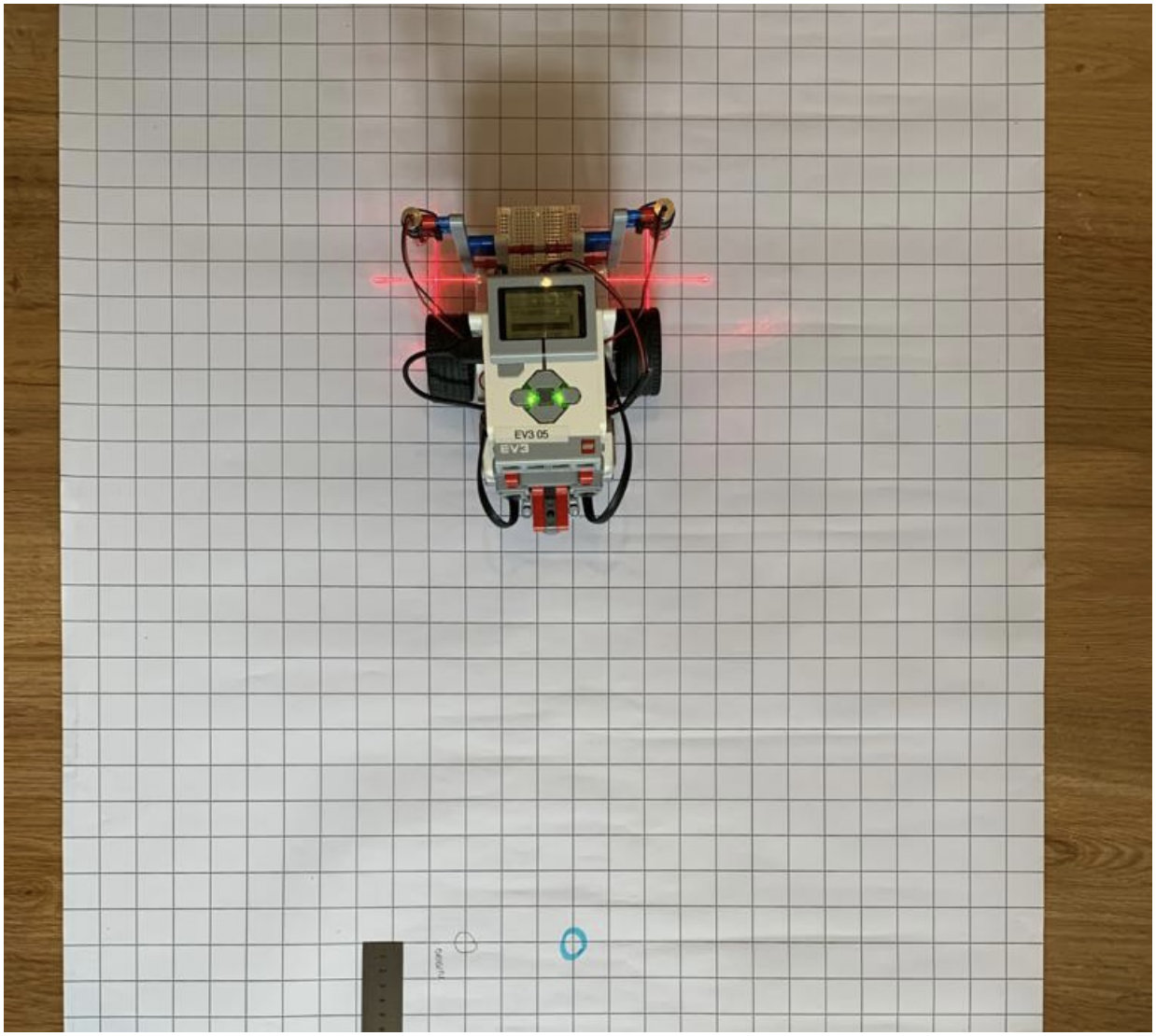


Figure 1.7: Rosa with the measurement facility 2

- The orientation is given by the equation 1.1. Since in our design the laser pointers are aligned along the x axis instead of the y axis the we calculate $\frac{\pi}{2} - \theta$ instead of directly finding θ therefore the orientation finding equation is slightly modified as below:

$$\frac{\pi}{2} - \theta = \tan^{-1}\left(\frac{y_2 - y_1}{x_2 - x_1}\right) \tag{1.1}$$

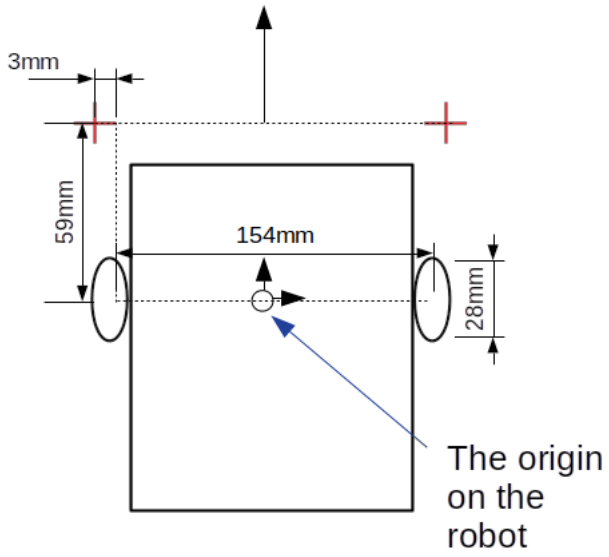


Figure 1.8: Dimensions of the robot

- If the variable matrix is defined as:

$$\begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{bmatrix} \tag{1.2}$$

- Then the Jacobian will be:

$$\begin{bmatrix} -\frac{y_1-y_2}{(-x_1+x_2)^2+(-y_1+y_2)^2} & \frac{y_1-y_2}{(-x_1+x_2)^2+(-y_1+y_2)^2} & -\frac{-x_1+x_2}{(-x_1+x_2)^2+(-y_1+y_2)^2} & \frac{-x_1+x_2}{(-x_1+x_2)^2+(-y_1+y_2)^2} \end{bmatrix}$$

- The upper bound and lower bound of errors were defined according to Figure 1.10 . The green arrows represent the vector in which direction the error should propagate for it to be the upper or lower bounds of the error.

- The python code given below is used to calculate the propagated error. In order to demonstrate the propagation of error, 5 runs of the straight, left and right motions were carried out.

- Note that for this example we used **only** the parallax error of measuring the laser cross with the ruler, with an estimated error of ±1mm. Therefore, the upper bound the error values will be $[0.1, -0.1, -0.1, 0.1]$ (in centimeters) and the lower bound will be $[-0.1, 0.1, 0.1, -0.1]$ (in centimeters);

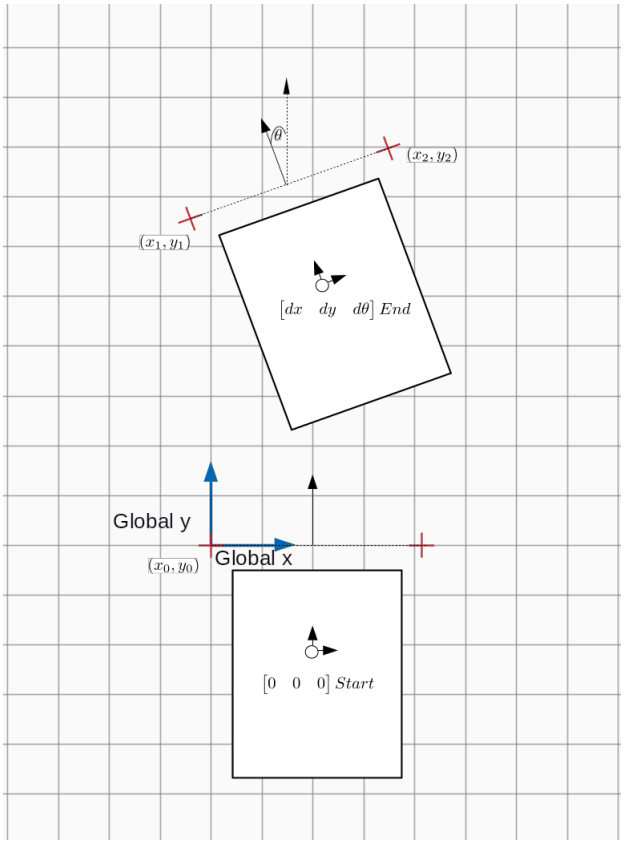


Figure 1.9: Robot orientation

where the variables are in the order of $[dx_1, dx_2, dy_1, dy_2]$.

- The figure 1.10 represent how we arrived at the order for the variables for the lower and upper bound error. We define the error vectors which will account for the direction and the magnitude of the error.

- Example:

1. For straight motion of measurements in order [x1,x2,y1,y2] [1.5 16.3 48.1 47.4]

   a. The error upper bound for straight motion is 0.7 °

   b. The error lower bound for straight motion is -0.7 °

2. For left motion of measurements in order [x1,x2,y1,y2] [-19.3 -11.2 24.6 36.8]

   a. The error upper bound for left motion is 1.1 °

   b. The error lower bound for left motion is -1.1 °

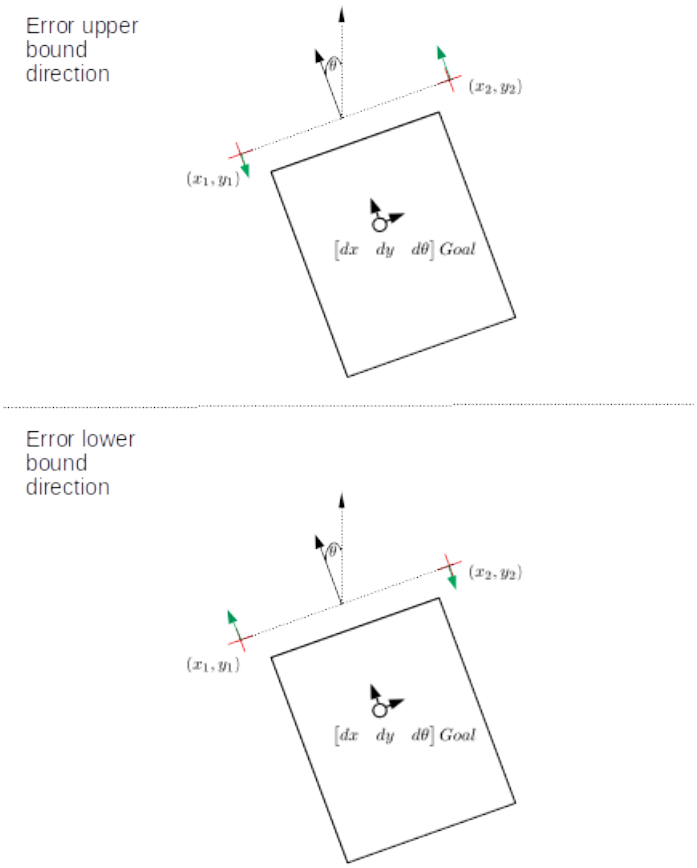


Figure 1.10: Error Bounds

3. For right motion of measurements in order [x1,x2,y1,y2] [27.4 35.3 37.1 24.5]

   a. The error upper bound for right motion is -0.3 °

   b. The error lower bound for right motion is 0.3 °

```
import numpy as np
import sympy as sp
import os
import sys

## All distances measurement are in centimeters(cm) and angles in radians


def error_upper_bound(lamd_jacobian, lamd_theta_eq,
                      coordinate_list, error_model=[0.1,-0.1,-0.1,0.1]):
```

```
    '''
    Calculate the upper error bound

    Parameters
    ----------------
    lamd_jacobian  : A lamdified jacobian matrix acting like a numpy array
    lamd_theta_eq  : A lamdified variable matrix acting like a numpy array
    coordinate_list: The current coordinates of the robot
    error_model    : The estimated error bounds for robot


    Returns
    ----------------
    A numpy array of propagated upper bound error


    '''

    ## The variables are in the order x1, x2, y1, y2
    current_jacobian = lamd_jacobian(coordinate_list[0], coordinate_list[1],
                                     coordinate_list[2], coordinate_list[3])
    current_theta_eq = lamd_theta_eq(error_model[0], error_model[1],
                                     error_model[2], error_model[3])

    return np.dot(current_jacobian, current_theta_eq)



def error_lower_bound(lamd_jacobian, lamd_theta_eq,
                      coordinate_list, error_model=[-0.1,0.1,0.1,-0.1]):
    '''
    Calculate the lower error bound

    Parameters
    ----------------
    lamd_jacobian  : A lamdified jacobian matrix acting like a numpy array
    lamd_theta_eq  : A lamdified variable matrix acting like a numpy array
    coordinate_list: The current coordinates of the robot
```

```
        error_model    : The estimated error bounds for robot


        Returns
        ---------------
        A numpy array of propagted lower bound error


        '''
        ## The variables are in the order x1, x2, y1, y2
        current_jacobian = lamd_jacobian(coordinate_list[0], coordinate_list[1],
                                         coordinate_list[2], coordinate_list[3])
        current_theta_eq = lamd_theta_eq(error_model[0], error_model[1],
                                         error_model[2], error_model[3])


        return np.dot(current_jacobian, current_theta_eq)

    ## Create the jacobian for the matrix
    ## Even if the angle theta is 90-calulated angle the error propagation is the same

    ## Creating the Jacobian

    x1,x2,y1,y2,theta = sp.symbols('x1,x2,y1,y2,theta')

    ## Check here if it should be atan or atan2. Using sp.arctan2 as per
    ## comments recevied during peer-review
    eq1 = sp.arctan2((y2-y1),(x2-x1))

    theta_eq = sp.Matrix([eq1])
    measurement_eq = sp.Matrix([x1, x2, y1, y2])

    error_jacobian = theta_eq.jacobian(measurement_eq)
    print("The Jacobian is \n")
    display(error_jacobian)

    print(f"The error matrix is \n")
    display(measurement_eq)
```

```
print("\n \n The error function is \n ")
display(error_jacobian*measurement_eq)
lamd_jacobian = sp.lambdify([x1, x2, y1, y2], error_jacobian,'numpy')
lamd_theta_eq = sp.lambdify([x1, x2, y1, y2], measurement_eq,'numpy')

##This is a test run data from 5 runs in each category(straigh,left and right)
## This dictionary is here to easily show how the data was collected
## A sepearate list will be made to make this calling of data easier
sample_test_data_1={
    "straight": [
        {"x1": 1.5, "y1": 48.1, "x2": 16.3, "y2": 47.4},
        {"x1": -1.5, "y1": 46.9, "x2": 13.2, "y2": 47.3},
        {"x1": -3.1, "y1": 45.5, "x2": 14.2, "y2": 46.4},
        {"x1": 0.7, "y1": 45.1, "x2": 15.4, "y2": 44.7},
        {"x1": -0.2, "y1": 45.8, "x2": 14.7, "y2": 45.9}
    ],
    "left": [
        {"x1": -19.3, "y1": 24.6, "x2": -11.2, "y2": 36.8},
        {"x1": -20.1, "y1": 26.5, "x2": -12.1, "y2": 39.0},
        {"x1": -19.8, "y1": 26.8, "x2": -11.7, "y2": 39.2},
        {"x1": -21.1, "y1": 24.9, "x2": -13.4, "y2": 37.5},
        {"x1": -18.6, "y1": 25.6, "x2": -10.0, "y2": 37.6}
    ],
    "right": [
        {"x1": 27.4, "y1": 37.1, "x2": 35.3, "y2": 24.5},
        {"x1": 28.1, "y1": 36.9, "x2": 35.7, "y2": 24.1},
        {"x1": 27.8, "y1": 39.1, "x2": 35.6, "y2": 26.4},
        {"x1": 27.1, "y1": 38.1, "x2": 35.0, "y2": 25.6},
        {"x1": 27.6, "y1": 39.2, "x2": 35.1, "y2": 26.7}
    ],
    "origin": {"x0": 0, "y0": 0, "x0_2": 14.8, "y0_2": 0}
}

def call_data(cat,index):
    '''
    This functions retuns the values from the dictionary of sample run data
    Parameters
    ----------
    cat    : The category of motion eg:Left -> char
```

```
        index : The index of run                -> int

        Returns
        -----------------------
        out_list: output the list of data in order [x1,x2,y1,y2] -> list


        '''

        if cat=='s':
            out_list = np.array([sample_test_data_1['straight'][index]['x1'],
                                 sample_test_data_1['straight'][index]['x2'],
                                 sample_test_data_1['straight'][index]['y1'],
                                 sample_test_data_1['straight'][index]['y2']])

        if cat=='l':
            out_list = np.array([sample_test_data_1['left'][index]['x1'],
                                 sample_test_data_1['left'][index]['x2'],
                                 sample_test_data_1['left'][index]['y1'],
                                 sample_test_data_1['left'][index]['y2']])

        if cat=='r':
            out_list=np.array([sample_test_data_1['right'][index]['x1'],
                               sample_test_data_1['right'][index]['x2'],
                               sample_test_data_1['right'][index]['y1'],
                               sample_test_data_1['right'][index]['y2']])


        return out_list


        ## Use this line to manually enter for a error bound



    ## The variables are in the order x1, x2, y1, y2


    coordinate_list = call_data('s', 0)
```

```
error_upper_straight = error_upper_bound(lamd_jacobian,lamd_theta_eq, coordinate_list)
error_lower_straight = error_lower_bound(lamd_jacobian, lamd_theta_eq, coordinate_list)


coordinate_list=call_data('l',0)
error_upper_left=error_upper_bound(lamd_jacobian,lamd_theta_eq,coordinate_list)
error_lower_left=error_lower_bound(lamd_jacobian,lamd_theta_eq,coordinate_list)


coordinate_list = call_data('r', 0)
error_upper_right = error_upper_bound(lamd_jacobian,lamd_theta_eq, coordinate_list)
error_lower_right = error_lower_bound(lamd_jacobian, lamd_theta_eq, coordinate_list)


print(f'The error upper bound for straight motion is {error_upper_straight[0][0]} radians \n')
print(f'The error lower bound for straight motion is {error_lower_straight[0][0]} radians \n')

print(f'The error upper bound for left motion is {error_upper_left[0][0]} radians \n')
print(f'The error lower bound for left motion is {error_lower_left[0][0]} radians \n')

print(f'The error upper bound for right motion is {error_upper_right[0][0]} radians \n')
print(f'The error lower bound for right motion is {error_lower_right[0][0]} radians \n')


##Use this to read from text file of the positions and calculate error for each reading

file_path = os.path.join(os.getcwd(),'coordinate_list.txt')
with open(file_path) as file:
    data = file.readlines()


propagated_error=np.zeros((len(data),6))

file_name='Propagated_Error'
try:


    f=open(file_path+file_name,'x') ##Open/Create a file to store data
```

```
except :

    print("\n Maps already exist in results folder \n")



for position in data:

    error_upper=error_upper_bound(lamd_jacobian,lamd_theta_eq,position)
    error_lower=error_lower_bound(lamd_jacobian,lamd_theta_eq,position)
    propagated_error[0:3]=position
    propagated_error[4]=error_upper
    propagated_error[5]=error_lower
    f.write(propagated_error+'\n')


f.close()
```

2

Task 1.2

2.1 Deliverables 1.2

Write a report detailing your execution of the experiment, including all observations made while running the robot. Especially detail and visualize the observed robot end poses and report statistical precision of the observed end poses (combined data). Summarize your findings, does the observed behaviour of the robot matches your expectations? Your report should cover:

1. The program and parameters used to drive the robot
2. Any observation made during the execution that may help one understand the outcome of the experiments (see also next week).
3. The observed data (spread-out of the manually measured end poses) as Excel or LibreOffice Calc files (stored in .csv file format and structured as presented in table 3 (section A.2)). For the data collected by the control script (using the encoders' readings) you can take already generated files from the EV3 brick.
4. Visualization of the i) robot's end poses (combined data from manual measurements), ii) complete robot's paths (combined data from encoder measurements) as well as iii) visual documentation of all aspects of setup and execution you deem important. See figures 10, 11, 12 (section A.1) that show some examples on how the data can be visualized.
5. Three videos showing robot's behaviour during the experiment (one for each type of motion)

2.2 Program & Parameters

```
#!/usr/bin/env micropython

from time import sleep, time
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

start_time = str(time()).split(".")[0]
times_motor_1 = []
times_motor_2 = []

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

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

```
        file_name = 'up_' + start_time

    elif buttons.right:
        move.on_for_seconds(SpeedPercent(40), SpeedPercent(30), 2.2, block=False)
        file_name = 'right_' + start_time

    if (motor_1.is_running):
        motor_1_path.append((motor_1.position * math.pi) / 180.0)
        times_motor_1.append(time())
    if (motor_2.is_running):
        motor_2_path.append((motor_2.position * math.pi) / 180.0)
        times_motor_2.append(time())

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

        with open(file_name + '_both_motors_path.csv', "w") as f_wheels_path:
            header = 'motor_1_path_rad, motor_2_path_rad, time_motor_1, time_motor_2\n'
            f_wheels_path.write(header)
            for motor_1, motor_2, time_1, time_2 in zip(motor_1_path, \
            motor_2_path, times_motor_1, times_motor_2):
                f_wheels_path.write(str(motor_1) + ', ' + str(motor_2) + ', ' +
                                    str(time_1) + ', ' + str(time_2) + '\n')  # in rad

        with open(file_name + '_robot_path.csv', "w") as f_robot_path:
            header = 'robot_position_x, robot_position_y, robot_orientation\n'

            f_robot_path.write(header)
            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)

                str_robot_position_x = str(robot_position_x)
                str_robot_position_y = str(robot_position_y)
                str_robot_orientation = str((math.pi / 2) - robot_orientation)

                f_robot_path.write(str(str_robot_position_x + ', ' + str_robot_position_y +
                                       ', ' + str_robot_orientation + '\n'))

        spkr.speak('Motion completed')
        break

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

2.3 Observations

2.3.1 Before the experiment

- Modifications were made to our initial measurement facility with the help of the pictures for the design of the measurement facility of Group 6. It consisted of 2 pencils attached with tape to hinges made of some lego blocks. After finishing the assembly several points were noted and thus some modifications were needed:

  1. It was observed that tape is bad for attaching the pencils because it does not exert sufficient pressure for a straight marking. So instead of tape we used cable ties as seen in Figure 2.1, 2.2.
  2. The front hinge was wobbly. During testing it freed itself from the original locking mechanism. This is why we needed to develop a guidance system, which holds the pen in place with an elastic band and guides the pen evenly to the ground during marking as seen in Figure 2.3.
  3. The design did not contain any help for resetting the robot to its start position. Since we did not need the lasers for marking positions any more, we attached them to the side of the robot's main corpus for easy resetting as seen in Figure 2.4, 2.5. Another thing we did, was to use a

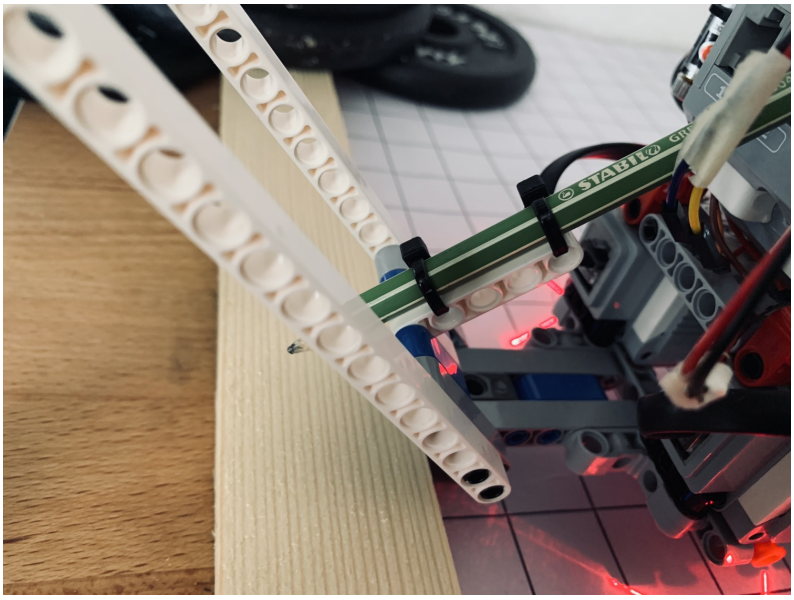


Figure 2.1: Attaching the pencils to Rosa

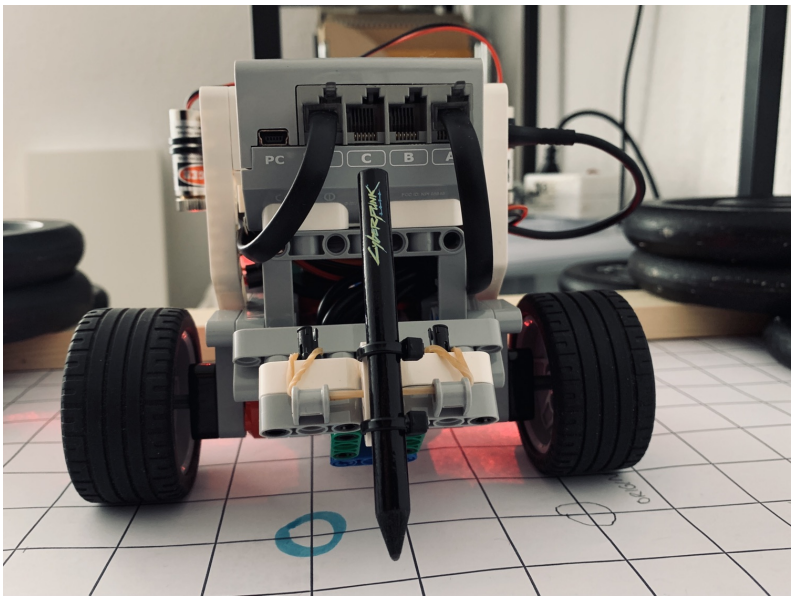


Figure 2.2: Attaching the pencils to Rosa

wooden plank (Figure 2.6) and some weights for creating a barrier to which we could roll the robot for a coarse positioning and the use the lasers for fast fine positioning.

4. The back hinge hold up fine, but it was a hassle to use the pen. So we attached some rails for easier one hand handling.

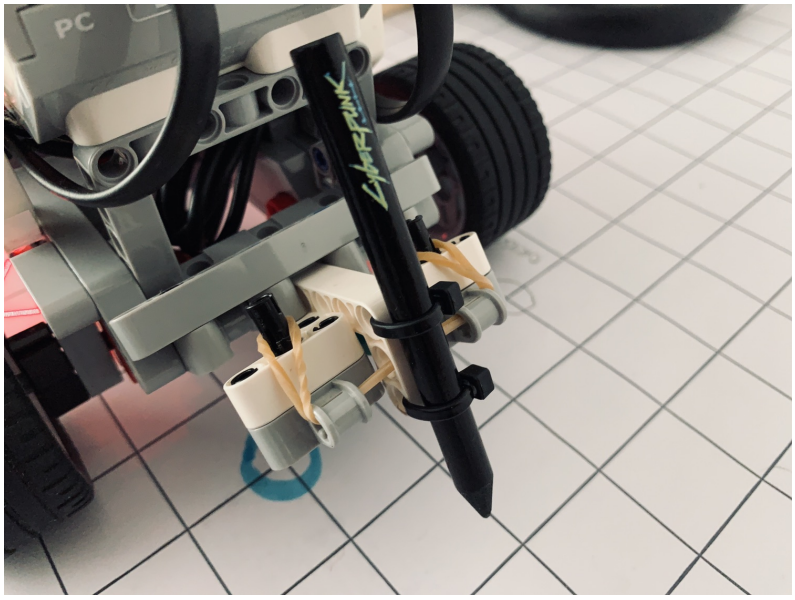


Figure 2.3: Elastic bands for guidance

2.3.2 During the experiment

1. Marking the positions with this new apparatus was a bit slower than using the lasers. The 2 pens had to be put down carefully. One hand had to hold the robot in position while the other hand pressed down the pen. As seen in Figure 2.7, this created a small dot and then carefully a small cross was put on the paper next to the experiments index.
2. With the old design we only needed to put the cross on the paper and did not need to touch the robot at all (besides during resetting the robot). This had saved some time and prevented accidentally moving the robot when using our old design.
3. For keeping track of the robot's logs easily, we modified the control script to name each run in the following pattern: [up — left-right][start_time][robot_path — both_motors].csv
4. After marking all 60 runs, we measured carefully the x and y position of each cross to a precision of 1 mm.

2.3.3 After the experiment

1. The handwritten measures are written down into .csv files for further analyses.

2.4 Data Collected

- Tables 2.1, 2.2, 2.3 show the data collected during the forward, right and left runs respectively.

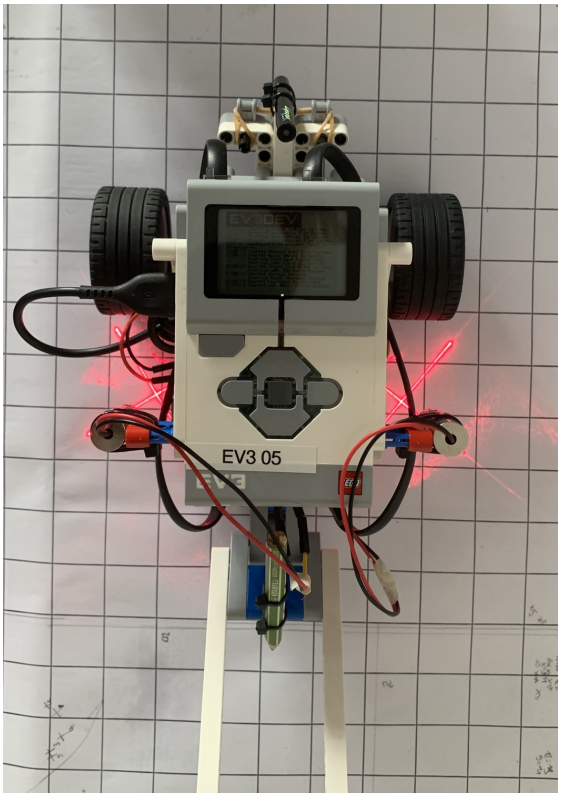


Figure 2.4: Lasers for ease of resetting position

2.5 Visualisations

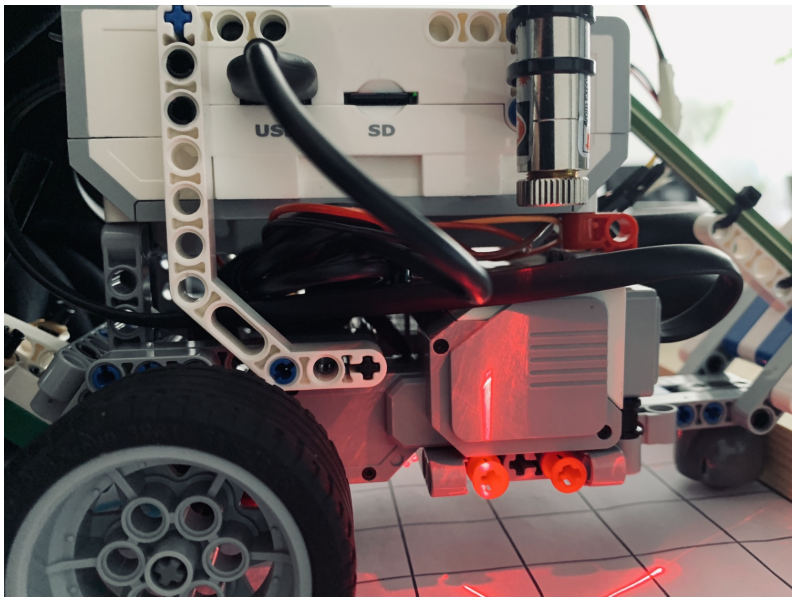


Figure 2.5: Lasers for ease of resetting position

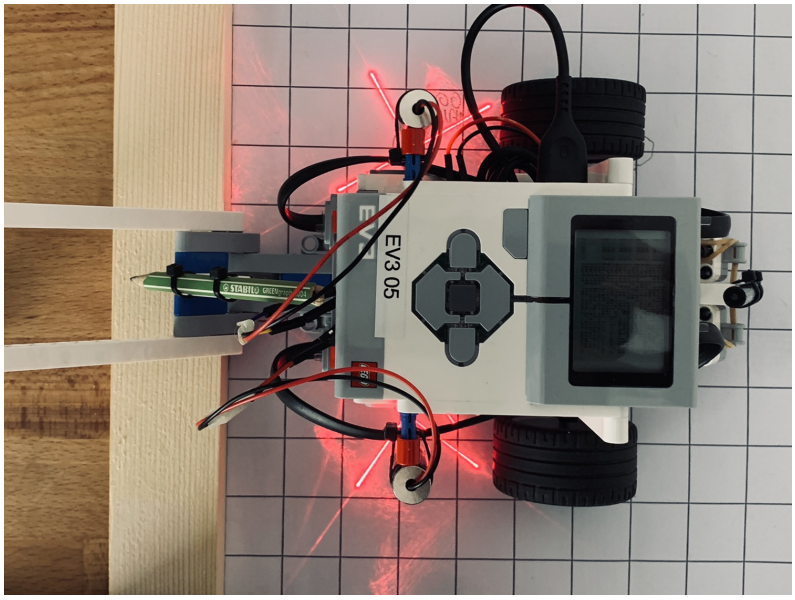


Figure 2.6: Wooden plank for creating a barrier

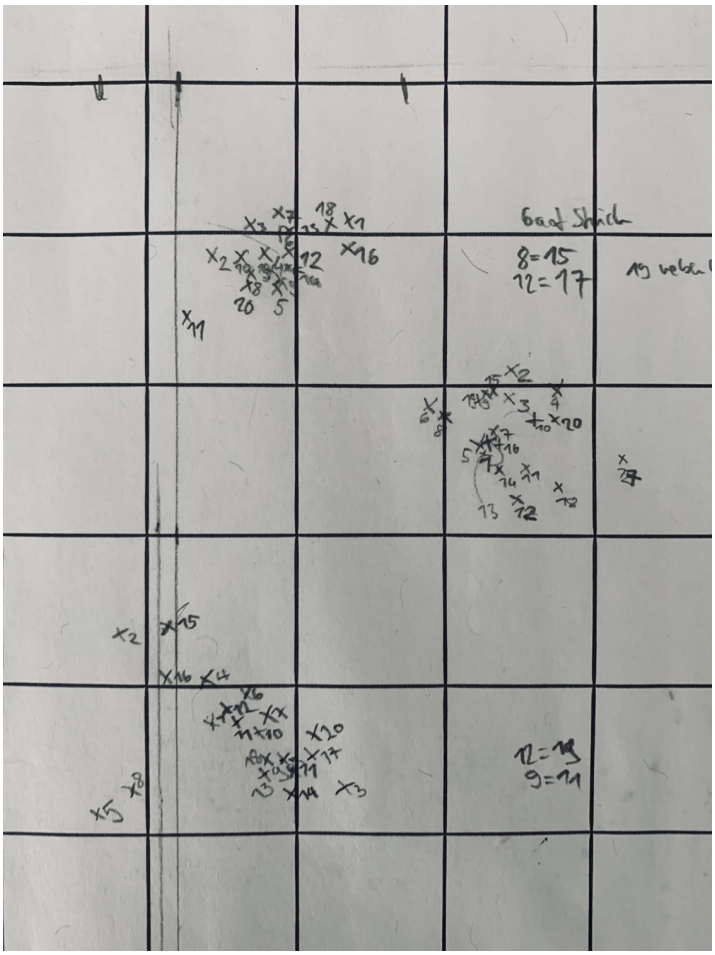


Figure 2.7: Making markings on the grid

	X axis (cm)	Y axis(cm)	Orientation(deg)
1	0.6	45.4	-1.4
2	-1.3	45.9	1.7
3	-0.6	45.9	0.7
4	-0.6	46.6	0.7
5	0.4	45.3	-0.5
6	-0.2	44.5	1.9
7	0.4	45.5	1.2
8	0.2	44.7	0.9
9	-0.4	45.4	-0.2
10	0.1	46.2	-0.7
11	1.1	46.1	-1.2
12	1.2	46.8	0.7
13	0.8	45.5	-0.7
14	1.1	45.7	-1.2
15	-0.6	45.7	0.4
16	0.5	45.7	-0.2
17	1.4	46.8	-4.1
18	1.6	46.6	-2.2
19	-0.6	45.4	0.7
20	-0.4	46.9	1.2

Table 2.1: Measurements taken in the forward run

	X axis (cm)	Y axis(cm)	Orientation(deg)
1	15.2	35.3	-57.4
2	13.3	34.3	-54.4
3	16.5	36.9	-60.8
4	14.2	35.5	-55.7
5	16.0	34.4	-51.2
6	15.0	35.9	-56.7
7	15.2	36.0	-58.7
8	15.3	35.3	-49.4
9	16.1	35.8	-58.9
10	15.4	35.9	-58.5
11	15.2	36.0	-59.8
12	15.0	35.6	-58.2
13	16.4	36.0	-58.7
14	16.7	36.1	-60.1
15	13.2	35.1	-54.5
16	14.1	34.9	-55.4
17	16.0	36.6	-59.6
18	16.0	35.9	-59.3
19	15.1	35.5	-58.0
20	15.5	36.5	-59.9

Table 2.2: Measurements taken in the right run

	X axis (cm)	Y axis(cm)	Orientation(deg)
1	-13.1	37.9	54.8
2	-12.3	35.9	53.3
3	-12.7	36.5	53.6
4	-12.3	36.7	53.5
5	-11.5	37.1	51.9
6	-12.6	37.0	53.3
7	-13.1	36.8	54.5
8	-11.7	36.7	51.9
9	-11.5	37.4	51.1
10	-12.1	36.4	52.9
11	-10.9	36.0	50.7
12	-12.3	37.2	52.8
13	-12.7	37.2	52.9
14	-11.8	37.5	52.1
15	-11.8	36.7	52.0
16	-12.4	38.1	53.8
17	-12.3	37.4	52.2
18	-12.8	37.8	53.5
19	-11.9	37.1	51.8
20	-12	36.8	51.8

Table 2.3: Measurements taken in the left run

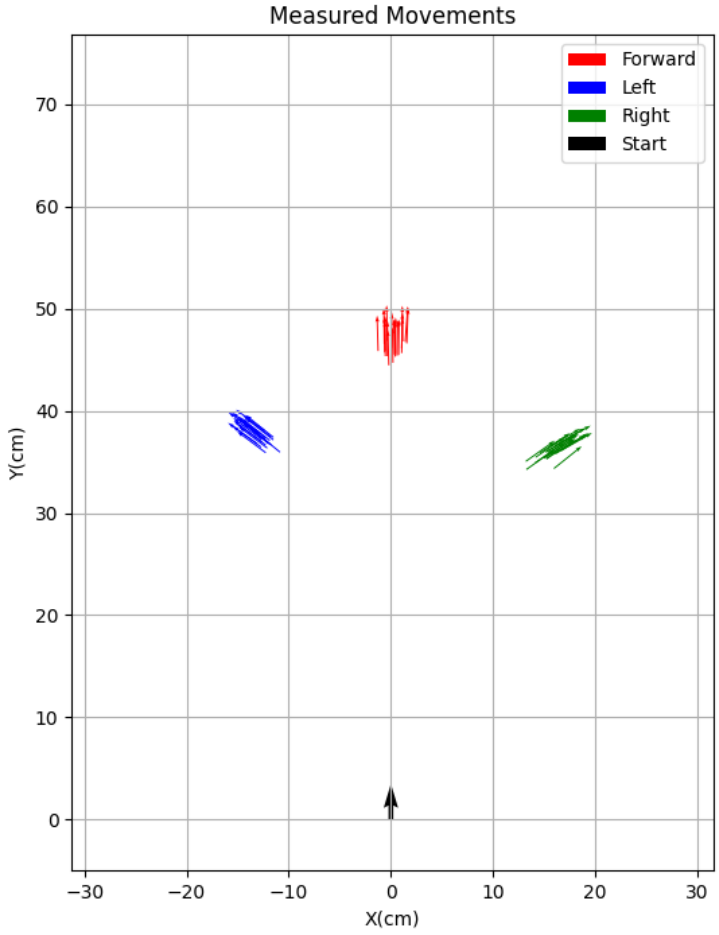


Figure 2.8: Visualising the end poses from the manual measurements

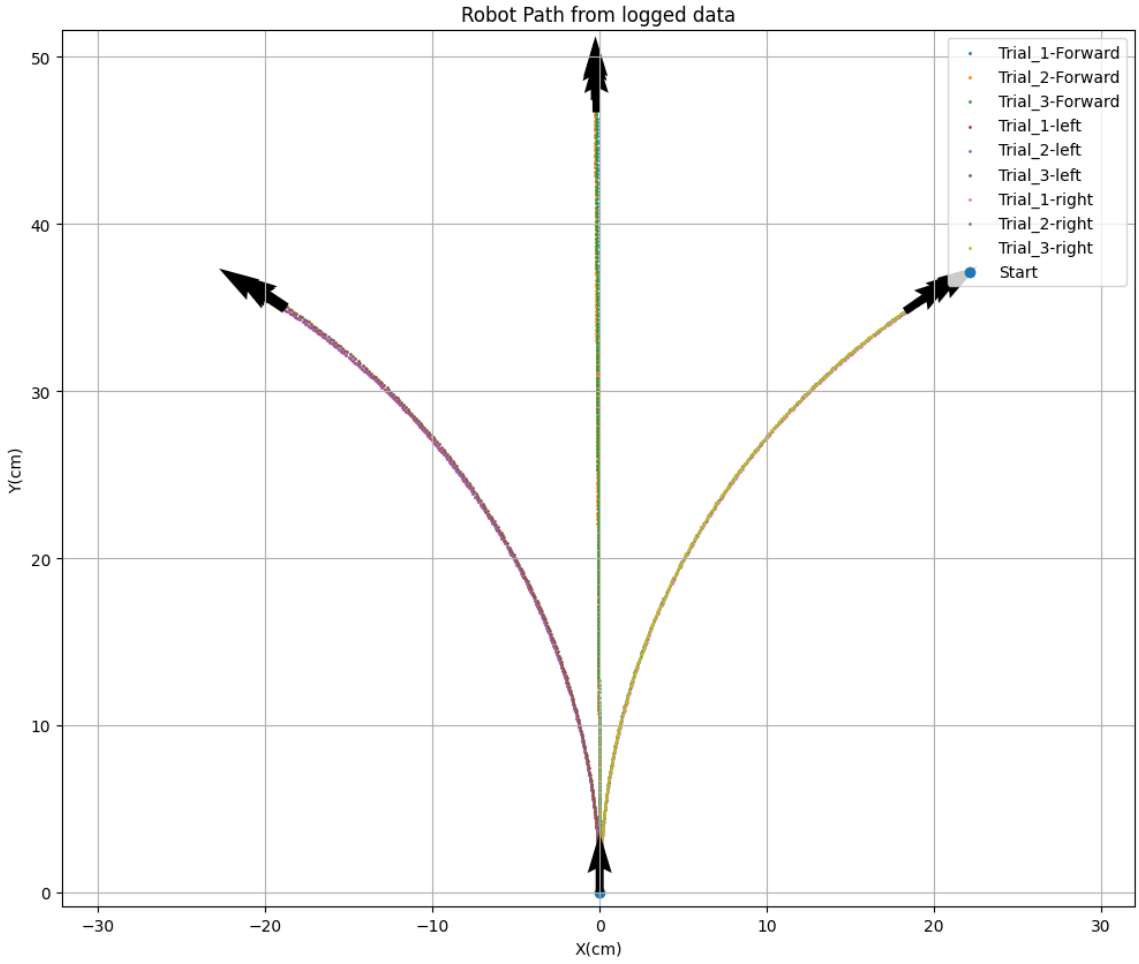


Figure 2.9: Visualising the paths from Encoder Data

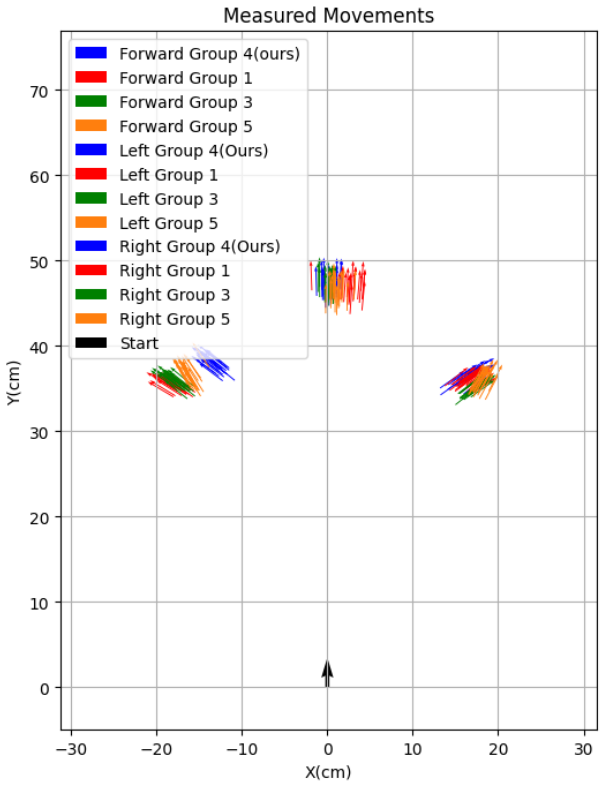


Figure 2.10: Visualising the end poses with manually measured data from all the groups

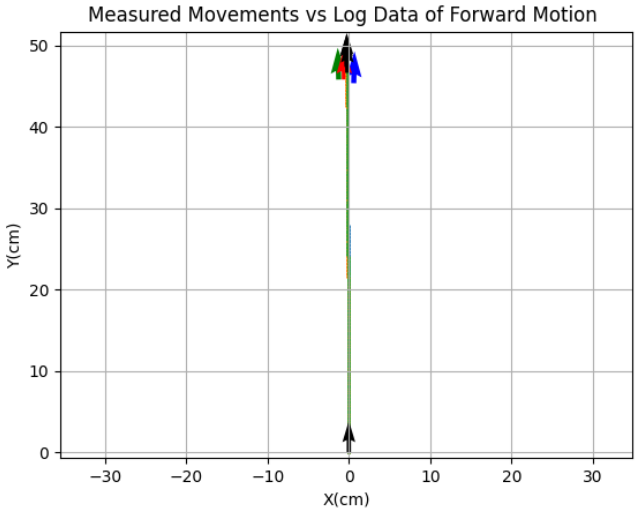


Figure 2.11: Manual Measurement v/s Encoder Data for Forward Run

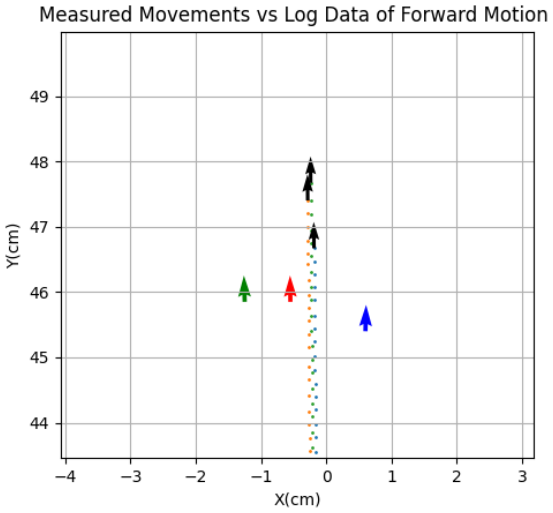


Figure 2.12: Manual Measurement v/s Encoder Data for Forward Run - zooming in

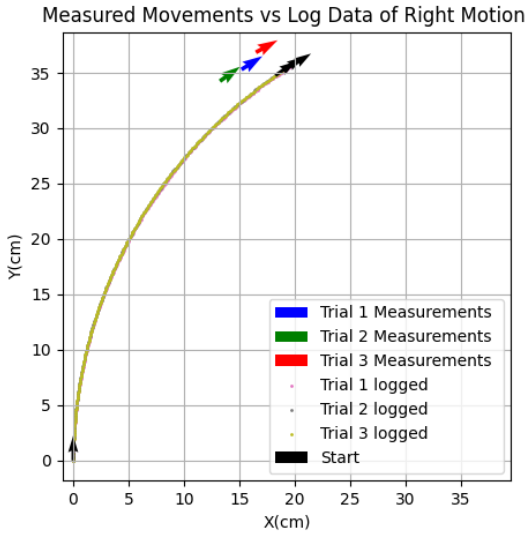


Figure 2.13: Manual Measurement v/s Encoder Data for Right Run

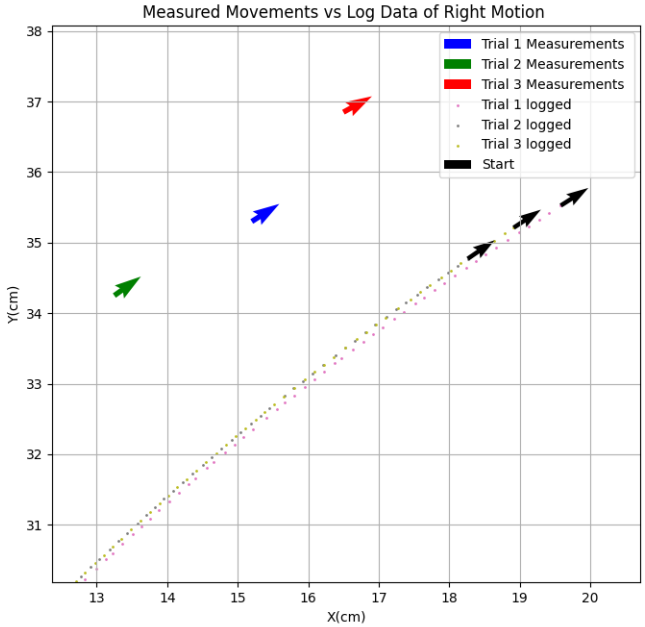


Figure 2.14: Manual Measurement v/s Encoder Data for Right Run - zoomed in

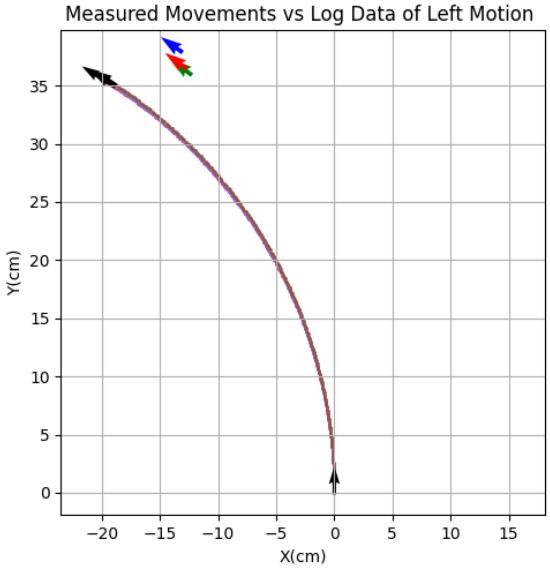


Figure 2.15: Manual Measurement v/s Encoder Data for Left Run

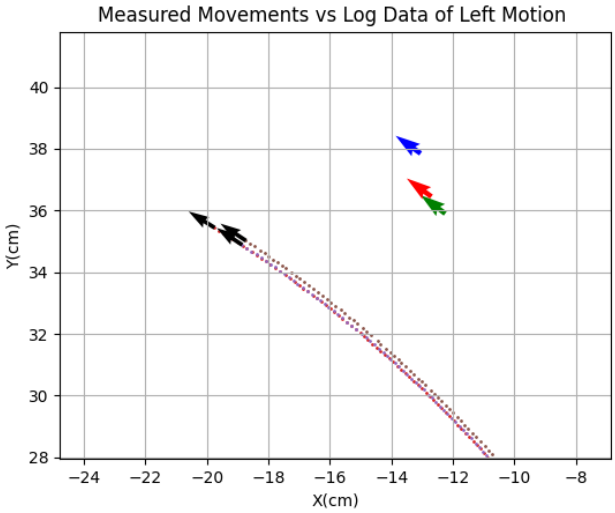


Figure 2.16: Manual Measurement v/s Encoder Data for Left Run - zooming in

3

Task 2.0

3.1 Deliverables 2

Update your previous week's report and add a description of the test for a match with a and the computed motion and the actual and expected accuracy and precision. Include appropriate figures, diagrams, and images backing up any claims you make. The report must be self-contained and provide enough details to support any statement you make. If applicable, include a section on problems encountered. Your update should cover:

1. Any possible pre-processing of your data, like outlier detection and removal.
2. Fit of a Gaussian, either two individual ones in the x and y directions for each of the three cases or a proper two-dimensional distribution per case.
3. Check whether the data are actually distributed according to a Gaussian distribution.
4. List of used software, including source of any function you wrote for performing your analysis.
5. An answer to the following question: When analysing the data with respect to the executed motions, which characteristic of the data do you establish here: the accuracy, the precision, or both?
6. For presenting some of the statistical parameters that characterize the observed robot behaviour, in your report, as an example, you can use the structure defined by table 4 (section A.2).

3.2 Preprocessing of Data

- Carrying out the experiment in Task 1, we obtained a total of 9 set of measurements -
  - X axis coordinates (in cm) for the forward, left and right motions
  - Y axis coordinates (in cm) for the forward, left and right motions
  - Theta or the orientation (in degrees) for the forward, left and right motions
- Combining the measurements obtained during the experiment in Task 1 from all 6 groups, we got a total of 120 data points in each of the above mentioned sets.

Motion	Random Variable	Original Data Points	Outlier Count
Forward	X (cm)	40	0
	Y (cm)	40	0
	Orientation (degrees)	40	0
Left	X (cm)	40	0
	Y (cm)	40	0
	Orientation (degrees)	40	0
Right	X (cm)	40	0
	Y (cm)	40	1
	Orientation (degrees)	40	1

Table 3.1: Outlier Detection

- Of these, it was observed that only 1 of the groups (Group 1) had a measurement process consistent to ours. Hence, we perform the statistical analysis on a total of 40 data points.
- The first pre-processing step was to remove the outliers.
  - Chebyshev Theorem (Eq 3.1 and 3.2.) was used to remove the outliers. It states that "only a certain amount of data points in a probability distribution can be present from a particular distance from the mean of the distribution".

$$P(\mid X - \mu \mid \leq k\sigma) \geq (1 - \frac{1}{k^2}) \tag{3.1}$$

$$P(\mid X - \mu \mid \geq k\sigma) \leq \frac{1}{k^2} \tag{3.2}$$

  - The number of outliers detected in each set is represented in the Table 3.1

- **3.3 Fitting a Gaussian for each measurement**
  - The Figures 3.1a, 3.1b, 3.1c, 3.2a, 3.2b, 3.2c, 3.3a, 3.3b & 3.3c represent the fitting of a Gaussian to the data points after the removal of the outliers.
  - The table 3.2 shows the various statistical measures computed for the manually measured data.
  - Chi square test is performed in order to evaluate whether the data fits the Gaussian distribution.
  - The significance level taken for the test is 0.01. It is observed that majority of the data does not fit the guassian distribution.

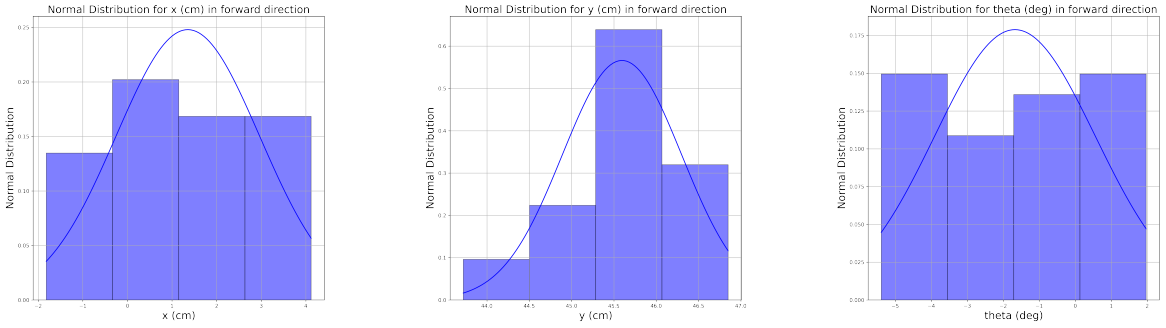
- The accuracy is computed by comparing the mean value in each set to the true value (from the encoder logs). Further details regarding the accuracy and precision of the measurements are discussed in section 3.6.

```
def detect_outliers(data, pp1 = 0.01, pp2 = 0.001) -> (int, np.array([])):
        '''
        Detect outliers based on Chebychev Theorem

        Returns
        ---------
        outlier_data_indices:  Indices of outliers detected
        final_data: Filtered data
        '''

        outlier_data_indices = []

        mu1, sigma1 = mean_variance(data)
        k = 1 / np.sqrt(pp1)
        odv1u = mu1 + k * sigma1
        odv1l = mu1 - k * sigma1

        new_data = data[np.where(data <= odv1u)[0]]
        outlier_data_indices.append(list(np.where(data >= odv1u)[0]))

        new_data = new_data[np.where(new_data >= odv1l)[0]]
        outlier_data_indices.append(list(np.where(new_data <= odv1l)[0]))

        mu2, sigma2 = mean_variance(new_data)
        k = 1 / np.sqrt(pp2)
        odv2u = mu2 + k * sigma2
        odv2l = mu2 - k * sigma2
        final_data = new_data[np.where(new_data <= odv2u)[0]]
        outlier_data_indices.append(list(np.where(final_data >= odv2u)[0]))

        final_data = new_data[np.where(final_data >= odv2l)[0]]
        outlier_data_indices.append(list(np.where(final_data <= odv2l)[0]))

        return outlier_data_indices, final_data

def chi_squared_test(data):
```

```
        np.random.seed(1)
        actual_data = data
        actual_freq,_ = np.histogram(actual_data, bins=4)
        mean = np.mean(actual_data)
        std = np.std(actual_data)
        true_data = np.random.normal(mean,std,len(data))
        true_freq,_ = np.histogram(true_data, bins=len(actual_freq))
        return (np.round(scipy.stats.chisquare(actual_freq, true_freq)[0], 2),
                np.round(scipy.stats.chisquare(actual_freq, true_freq)[1], 2))

data_filtered = [forward_new, left_new, right_new]
for direction in data_filtered:
    for i in range(3):
        if chi_squared_test(direction.T[i])[1] < 0.01:
            print('Reject', chi_squared_test(direction.T[i]))
        else:
            print('Accept', chi_squared_test(direction.T[i]))
```

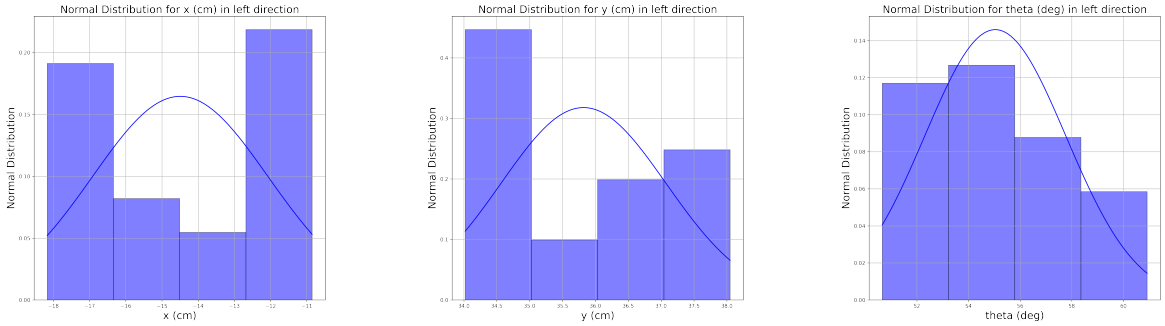
Motion	Random Variable	Mean (cm)	Variance (cm)	Accuracy (cm)	Chi Value	P-value	Null Hypothesis: Data Fits the Guassian Distribution (Accept?)
Forward	X (cm)	1.35	1.61	1.0	19.56	0.0	No
	Y (cm)	45.6	0.70	0.5	13.75	0.0	No
	Orientation (degrees)	-1.68	2.23	1.2	45.46	0.0	No
Left	X (cm)	-14.49	2.42	1.5	87.17	0.0	No
	Y (cm)	35.82	1.25	0.9	138.76	0.0	No
	Orientation (degrees)	55.04	2.73	7.3	52.9	0.0	No
Right	X (cm)	15.35	0.90	1.1	7.72	0.05	Yes
	Y (cm)	35.37	0.62	0.7	9.8	0.02	Yes
	Orientation (degrees)	-55.02	2.64	8.3	44.57	0.0	No

Table 3.2: Statistical parameters for manual measurements



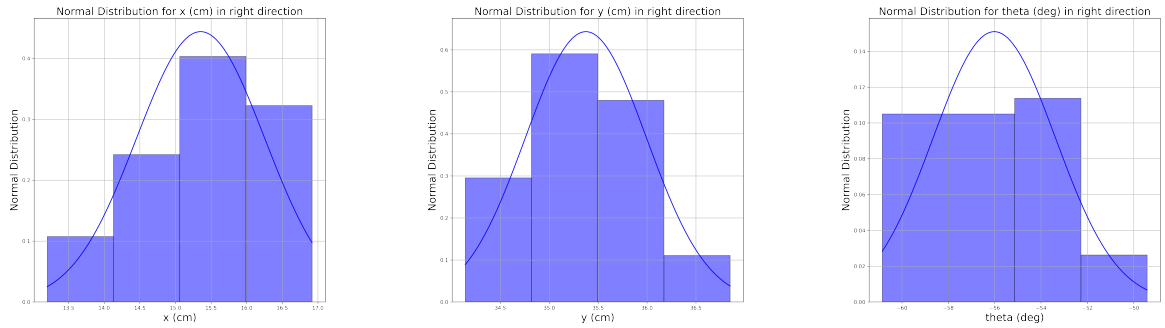
(a) X-axis coordinates (b) Y-axis coordinates (c) Orientation

Figure 3.1: Manually measured x, y and orientation in the forward direction



(a) X-axis coordinates (b) Y-axis coordinates (c) Orientation

Figure 3.2: Manually measured x, y and orientation in the left direction



(a) X-axis coordinates (b) Y-axis coordinates (c) Orientation

Figure 3.3: Manually measured x, y and orientation in the right direction

3.4 Uncertainity Ellipses after PCA

- After removing the outliers, PCA is used to further reduce noise in the data through dimensionality reduction.
- For this, data is projected into k dimensions by using an orthogonal linear transformation. Here, k is represented by the dimensions with the highest variance.
- The eigenvectors of the covariance matrix of the data is taken as the target dimensions and they are ranked according to their eigenvalues. Eigenvalues which have the highest values show the highest variance.
- The code snippet used for PCA is given below:

```
def get_PCA(data, components = 3):
    X = data
```

```
pca = sklearn.decomposition.PCA(n_components=components)
pca.fit(X)
return pca.transform(X)
```

Figures 3.4a, 3.4b & 3.4c represent the uncertainity ellipses after PCA.

3.5 List of software used

- All pre-processing and visualisation of of data was carried out using Python. The following libraries were used:
  1. numpy
  2. pandas
  3. scipy.stats
  4. matplotlib
  5. sklearn

3.6 Observations regarding manual measurements and encoder logs

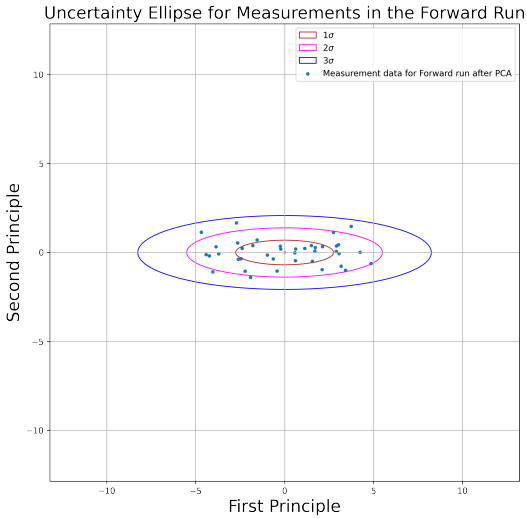
During the process of visualising the evaluated data, we quickly discovered that the encoded data from the EV3 and the measured data did not fit together as good as we would have hoped. For each of the three movement cases we noticed mainly two scenarios in the data delivered by the encoder -

1. The encoder delivers data, which fits to the manually measured measurements.
2. The encoder delivers data which is way off, the mean of the data points hover around 166 for both x and y axis.

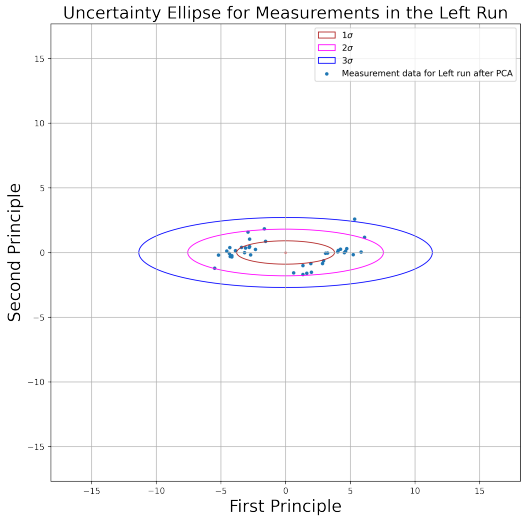
When removing around have the data, which is in the second mode, we can compare standard error between the two different contributions.

1. Forward case: the standard error is much smaller along the x-axis (encoded: 4.9, measured: 31). Along the y-axis, the standard error is the same (around 28 for both)
2. Left case: The error along the x-axis is the same for both methods (24), along the y-axis the error is smaller (encoded 17, measured 25)
3. Right case: Along the x-axis the encoded has an error of 38 and measured around 42. Along the y-axis its 24 and 29.

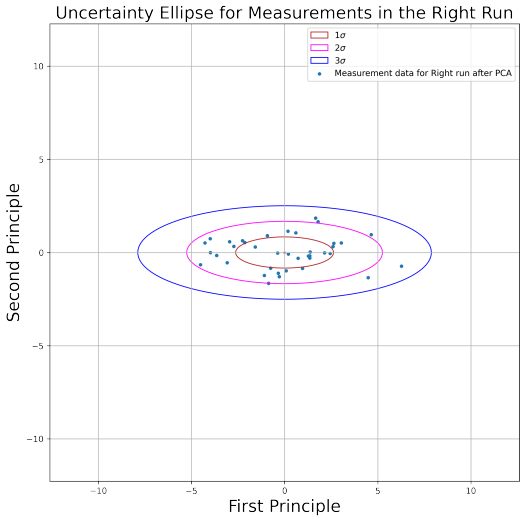
We see that the difference between the single measurements in the encoded methods are much smaller than for the manually measurements (Figures - 3.5, 3.6 & 3.7) Since the data points are so few now, we can infer that the smaller standard error for the encoded data comes from a much lower resolution - this makes the data points much more similar.



(a) Forward Direction Measurements



(b) Left Direction Measurements



(c) Right Direction Measurements

Figure 3.4: Uncertanity Ellipses for the measurements after PCA

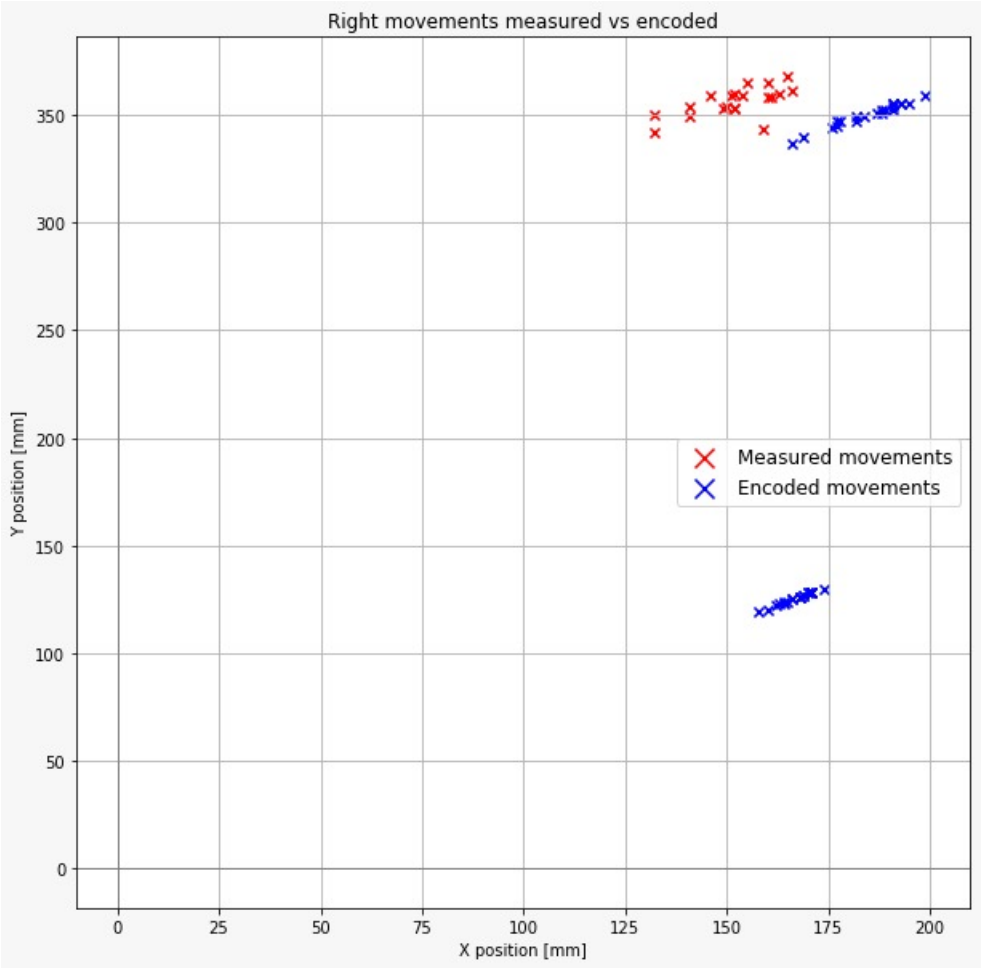


Figure 3.5: Visualising the manually measured right poses with encoder logs

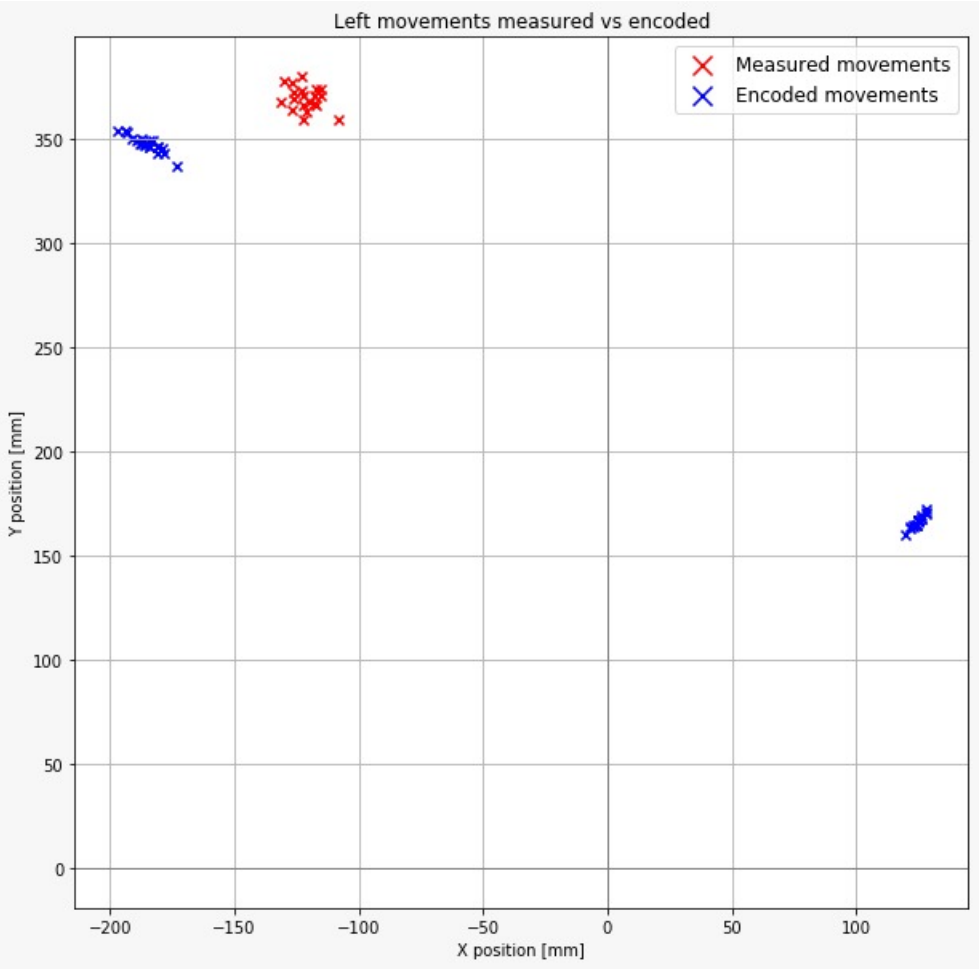


Figure 3.6: Visualising the manually measured left poses with encoder logs

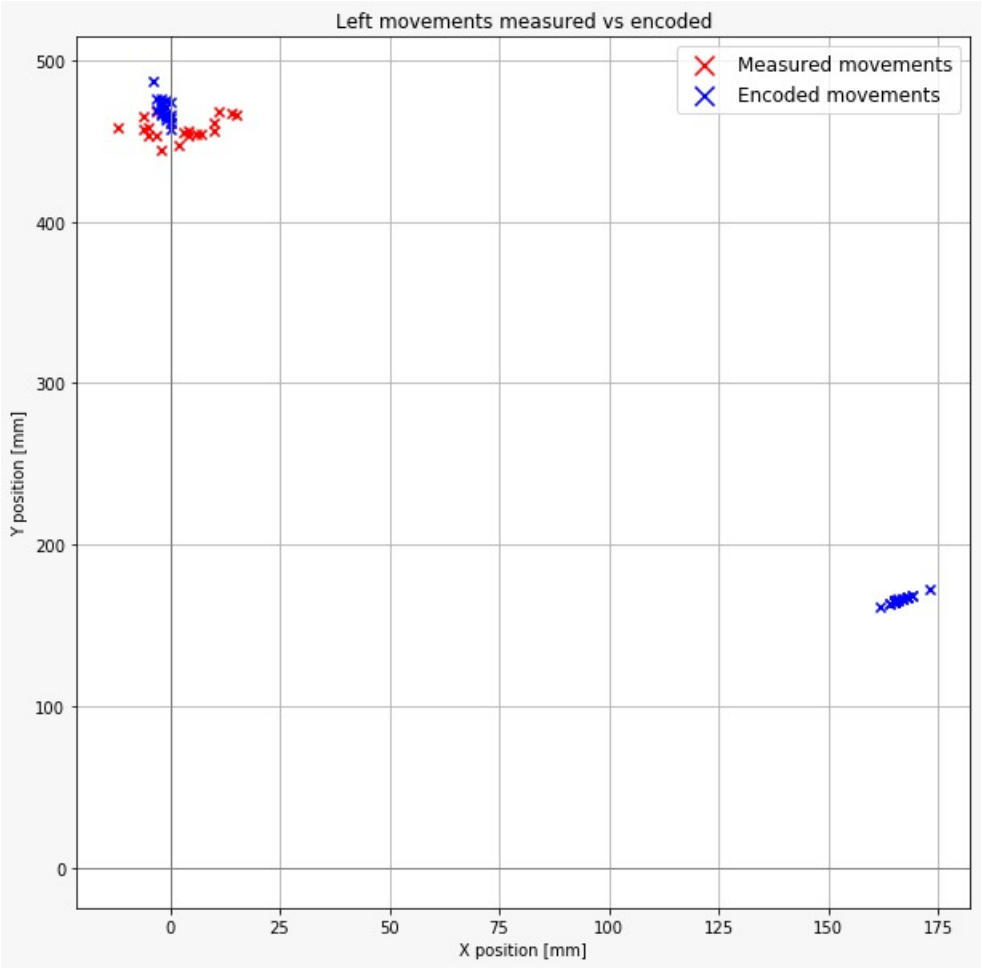


Figure 3.7: Visualising the manually measured forward poses with encoder logs

References