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

Experiment 01

1.1 Deliverables

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 using the **Lego Mindstorms EV3**(evolution 3) kit. Figure 1.1 shows our final design, with the measurement facility for the robot intact.
- The following are the materials used for building Rosa:
    1. Lego EV3 kit
    2. Focusable Laser Module (2 units)
    3. Jumpers for powering the laser modules
- The robot is three-wheeled with a differential drive configuration. In this configuration (Figure 1.3), two of the wheels placed in front of the robot on either sides, have independent actuators. The 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.

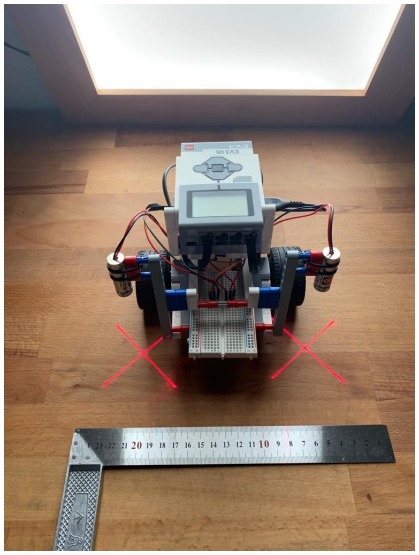

Figure 1.1: Rosa - Front View

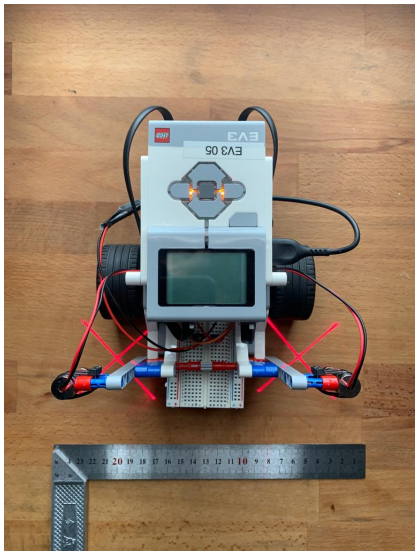

Figure 1.2: Rosa - Top View

- The measurements of the bot's motions are made on top a A1 sized grid sheet (grid size $\approx$ 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 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

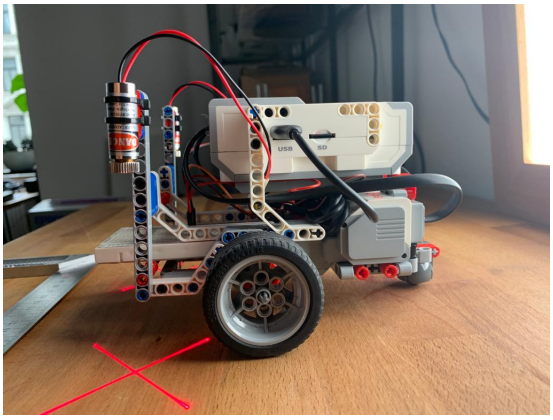

Figure 1.3: Rosa - Side View

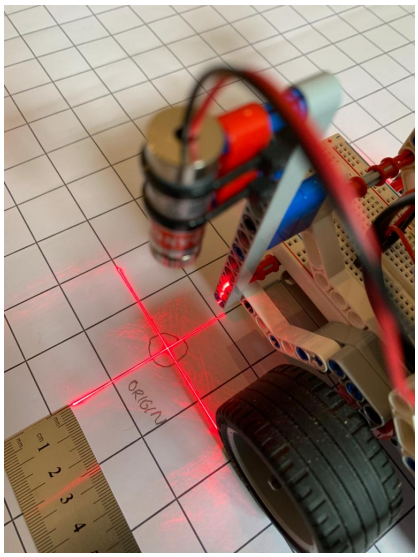

Figure 1.4: Rosa on the grid sheet

makes any additional batteries unnecessary.

- The **pose** of the robot described by the vector $[x, y, \theta]^T$ is the **measurand**. Here, (x, y) represents the **translation** of the robot which is measured in **millimeters** and θ represents it's **orientation** which is the **measurement result** obtained from the measured translation, in **degrees**.

- The Figure 1.5 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**:

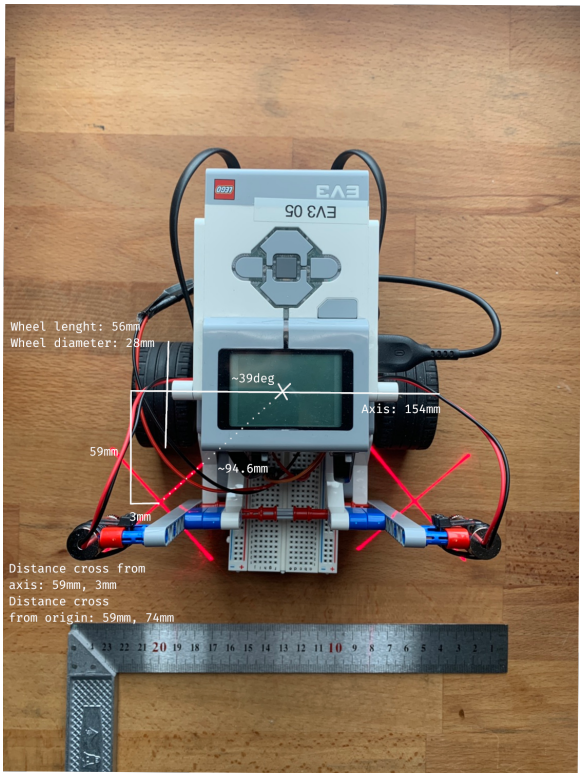

Figure 1.5: Rosa with the measurement facility

1. Securing the A1 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**:

1. The stop positions are marked with the help of the laser crosses, rulers and pencils on the grid.

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. (see attached files)
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. Each position is measured using a coordinate system with the start position of the left laser as point of origin.

- 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 1.6

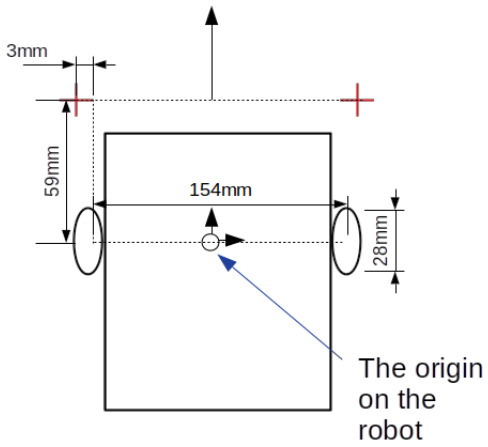

Figure 1.6: Dimensions of the robot

- 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.
- Since the magnitude change of x and y (Δx & Δy) will be the same for for markers and the origin of the robot, we just have to account for the offset. The change in orientation will be the same and requires no additional calculations. See Figure 1.7 for details.

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. Parallax error when marking the laser cross = $\pm 2mm$
2. Parallax error when measuring the laser cross with the ruler = $\pm 1mm$
3. Pressing the button pushes the robot from one position and can put extra weight on one side of the wheel = $\pm 2mm$

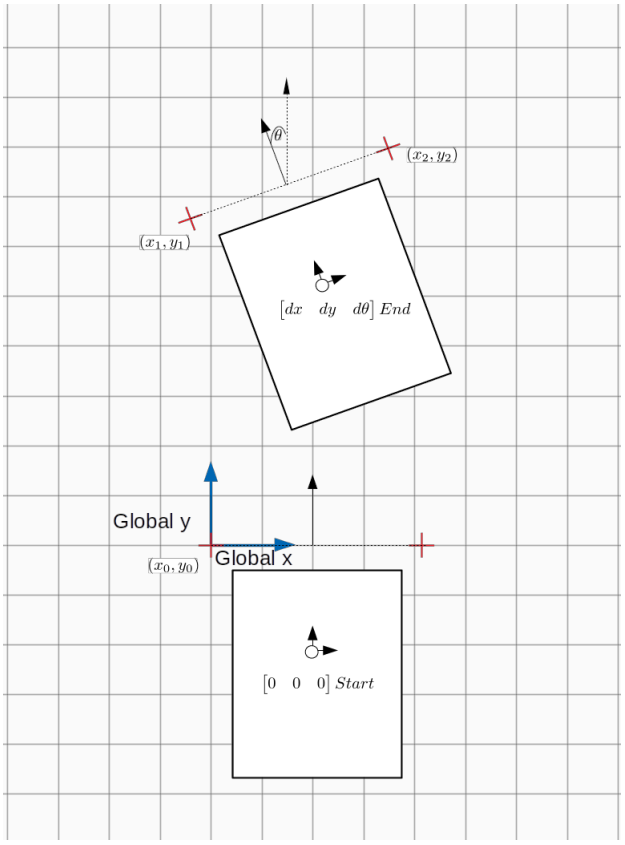

Figure 1.7: Robot orientation

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. = $\pm 2mm$

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 1.7 for naming convention), the error associated to each of them contribute to the propagated error.
- The orientation is given by the equation 1.1. Due to the alignment of our lasers we find $\frac{\pi}{2} - \theta$ instead of directly finding θ:

$$\frac{\pi}{2} - \theta = arctan(\frac{y_2 - y_1}{x_2 - x_1}) \tag{1.1}$$

- 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 were defined according to Figure 1.8 . 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.

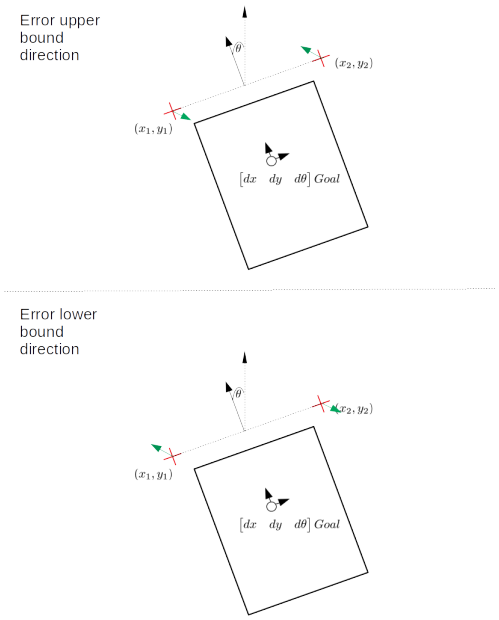

Figure 1.8: Error Bounds

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

- 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 0.01285 radians
    b. The error lower bound for straight motion is -0.01285 radians
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 0.01893 radians
    b. The error lower bound for left motion is -0.01893 radians
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 -0.00425 radians
    b. The error lower bound for right motion is 0.00425 radians

```
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
eq1 = sp.atan2((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()
```

References