Computer Project I – Final Report

1st Asger Høøck Song Poulsen
Undergraduates at Aarhus University,
Dept. of electrical and computer engineering
Study number:
AU-id:

2nd Andreas Kaag Thomsen
Undergraduates at Aarhus University,
Dept. of electrical and computer engineering
Study number: 202105844
AU-id: au691667

***Abstract*—This is our abstract.**

I. Introduction

Overall, the project can be divided into four parts with certain aim and objectives:

1) Understanding the basic concepts of robot programming by working with an Arduino - that is, the *See-think-act cycle*. Reading and analyzing data from Range sensor and RGB sensor.
2) ROS Programming on Raspberry PI. Learning the structure that ROS provides and going through the beginner's tutorial.
3) Programming the robot to avoid obstacles. Then optimize the robot's performance both with respect to linear speed and collision avoidance.
4) Finalizing the code and testing the robot on an obstacle course.

II. Specifications

We have used the following equipment throughout the course.

- Arduino PRO MICRO - 5V/16MHZ
- Ultrasonic Range Finder (LV-MAXSONAR-EZ0)
- RGB Light Sensor ISL29125
- LED's, cables etc.
- Turtlebot3 Burger Robot equipped with i.a. a Raspberry Pi 3 and a 360°LiDAR sensor.

For coding we have used the language C for programming the Arduino on the Arduino software. For programming the Turtlebot we have been using Python and the command prompt with the built in nano-editor.

III. Design and implementation

Design and implementation of the system.

A. Part 1

We were to consider the *See-think-act* cycle as can be seen in figure 1 below:

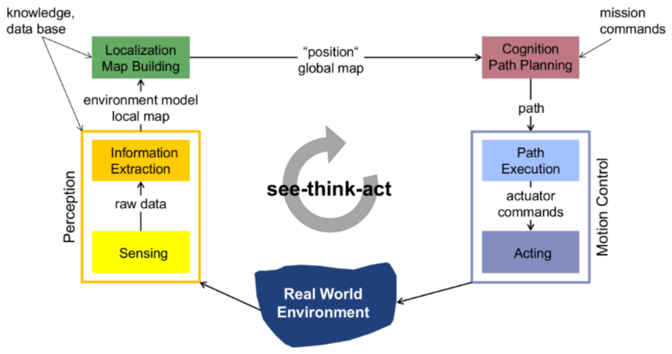

Fig. 1. See-think-act cycle

On our breadboard we connected the Arduino to which we connected the ultrasonic range finders and som LED's. The cycle was then implemented as follows: the range finder sensors **saw** any obstacle we put in front of it, and sent some data back to Arduino. This data was computed - that is the **think** step of the cycle. Then the **act** step was performed, which in this case was the LED blinking.
Afterwards we extended the circuit to include three range finder sensors plus the RGB light sensor and four LED's. This thus acted as a simulation of the real Turtlebot, which we should work with in part 3.

B. Part 2

In this part of the course we worked on a Virtual Machine (VM) on which we had installed Ubuntu. We went through the ROS beginner's tutorial and learned the structure of a ROS system.

C. Part 3

As described, the Turtlebot is equipped with a LiDAR 360°laser sensor, which measures the distance. It continuosly returns an array:

$$\texttt{dist} = [d_0, d_1, \ldots, d_{359}]$$

where d_i is the distance to the nearest object at angle i. We decided to look at a span of 120 degrees, which we divided into three distinct parts:

$$\begin{aligned}\texttt{left} &= [d_{15}, d_{16}, \ldots, d_{60}], \texttt{N=45}\\ \texttt{front} &= [d_0, d_1, \ldots, d_{14}, d_{345}, d_{346}, \ldots, d_{359}], \texttt{N=30}\\ \texttt{right} &= [d_{300}, d_{301}, \ldots, d_{344}], \texttt{N=45}\end{aligned}$$

We decided to do this so the robot more often would *turn* instead of just driving *backwards*, since this supposedly would achieve an overall higher linear speed.

For the different cases of obstacles we have made several cases of `if-else` statements. The cases and the decisions in each case can be seen in the figure below.

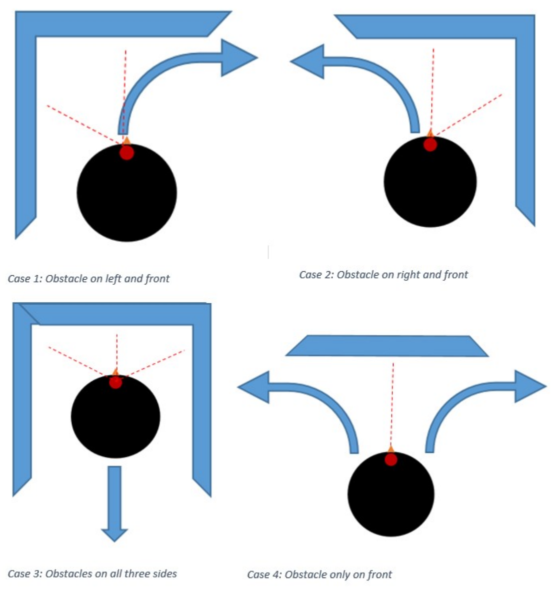
Case 1: Obstacle on left and front
Case 2: Obstacle on right and front
Case 3: Obstacles on all three sides
Case 4: Obstacle only on front

Fig. 2. Obstacle cases

D. Part 4

In the last part of the course we should test the robot on an arbitrary course of obstacles. We did, however, test the robot continuosly in part 3, so we focused this part on optimizing the existing code. This will be elaborated in the next section.

IV. Experiment Setup and Results

A. Part 1

We connected the Range sensor the Arduino and intialized it as follows:

```
int SENSOR = A0; //range sensor connected to port A0
double range_input = 0; //variable for storing input
    values
void setup()
{
  pinMode(SENSOR,INPUT); //sensor declarared as an
    input
}
```

Listing 1. Initialization of variables

In the control loop we updated `range_input` with the value read from the sensor with using the `analogRead` function. Through a series of `if-else` statements we made the LED blink with different intervals for different distances:

```
if (range_input+margin < 20 && range_input+margin >
    0) {
    TXLED0;
    delay(50);
    TXLED1;
    delay(50);
}

else if (range_input+margin < 30 && range_input+
    margin > 25) {
    TXLED0;
    delay(333);
    TXLED1;
    delay(333);
}

else if(range_input+margin < 25 && range_input+
    margin >20) {
    TXLED0;
    delay(1000);
    TXLED1;
    delay(1000);
}
TXLED1; //LED turned off by default
```

Listing 2. `else-if` statements for Arduino

Afterwards we extended the circuit to include three range sensors in the same way as above. We discovered that our measurements at first were wrong since we did not convert the input values to distances. Looking at the equipment specifications we found out that the sensor uses a scaling factor of $6.4\frac{mV}{inch}$. By dividing the analog input with this scaling factor, a measurement in inches is calculated. Afterwards, this is multiplied by $2.54\frac{cm}{inch}$.

```
front_input_cm = (front_input/6.4)*2.54;
left_input_cm = (left_input/6.4)*2.54;
right_input_cm = (right_input/6.4)*2.54;
```

Listing 3. Calculations of ranges

Now we had the correct measurements which we also manually confirmed with a ruler.
Lastly we also implemented the RGB light sensor on the breadboard, which was intialized in the same way as the range sensors. The RGB sensor returned one value measured in lux for the luminance. If the luminance was less than 30, one LED was turned on. With everything connected as described above, our final circuit is depicted in figure 3 below.

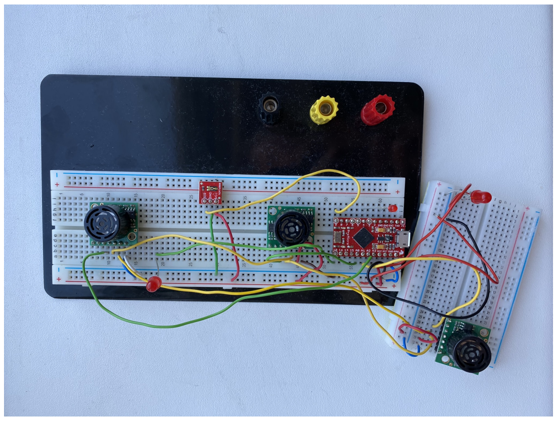
Fig. 3. Final Arduino circuit

B. Part 2

As mentioned we followed the ROS beginner's tutorial in which we installed ROS using several commands. We got stuck in part 2 of the tutorial, so we decided to go back to the beginning and reinstall everything. After doing this, we were able to complete all the steps from 1 through 17. We got a deeper understanding of how the ROS system is structured and learned several ROS-related commands. We will explain the most essential for our further work with the Turtlebot below[1]:

1) **roscore**: roscore is a collection of nodes and programs that are pre-requisites of a ROS-based system. You must have a roscore running in order for ROS nodes to communicate.
2) **rosrun**: rosrun allows you to run an executable in an arbitrary package from anywhere without having to give its full path.
3) **source**: When opening a new terminal your environment is reset and you are obliged to use the `source` command to re-source the .bashrc file.
4) **nano**: nano is a text editor that is used to edit files. We used it to edit our code files directly in the terminal.

C. Part 3

In this part we are going to describe our initial implementation. The optimization we did, will be described in part 4.

```
import rospy as rp
else:
    left_lidar_samples_ranges = 60
    right_lidar_samples_ranges = 345
    front_lidar_samples_ranges = 15
```

Listing 4. My example

D. Part 4

V. Discussion

In this section of the report we will discuss some of the difficulties we encountered during the project. We are also going to discuss whether or not we reached our goal. Again, we have divided it into the four main parts of the project described earlier.

A. Part 1

1) **Preparation to main project**

B. Part 2

1) **Understanding the ROS structure**

C. Part 3

1) **Getting wrong sensor measurements**
We experienced quite a difficulty in getting the right measurements from the sensor. We knew that it would return an array of size 360 with one distance for each angle. However, we couldn't figure out how they were indexed and our robot thus behaved incorrectly and sometimes turned left when it should turn right.
Solution
We solved the problem by manually printing out different test angles and then physically measure that angle also. For instance printing `scan_ranges[45]` which gave some change in output distance for some specific angle, which we measured. In turned out that it our code should be reversed.

D. Part 4

1) **Structuring the `else-if` statements**

VI. References

- Course material from Brightspace.
- ROS beginner's tutorial: *http://wiki.ros.org/ROS/Tutorials*
- Data sheet and specifications for Ultrasonic Range Finder: *shorturl.at/cBHN1*
- Data sheet and specifications for RGB light sensor: *shorturl.at/htEP8*

[1]These definitions are taken from the ROS beginner's tutorial, which is linked in section VI