Pac-King
Placement of IR Sensors on Line Following Robot and Construction of a Gripper and Lift
TESS ANTONSSON
SOFIA JÖNSSON
Pac-King

Placement of IR Sensors on Line Following Robot
and Construction of a Gripper and Lift

TESS ANTONSSON, SOFIA JÖNSSON

Bachelor’s Thesis at ITM
Supervisor: Nihad Subasic
Examiner: Nihad Subasic

TRITA-ITM-EX 2019:25
Abstract

Line following robots are a practical mechatronics solution in a world that is becoming more and more automated. With added gripping and lifting abilities, a very versatile robot can be created. The goal of this thesis was to create a prototype that could navigate a black line using infrared sensors whilst gripping and lifting a package. A working prototype was built and five different sensor placements were evaluated to assess which was optimal for line-following. The results showed that too close placement of the sensors led to a less stable system while more distance between the sensors made a faster and more accurate system.

Keywords: Mechatronics, Line Following Robot, Infrared sensor, Gripper, Lift, Robotics
Referat

Placering av IR Sensorer på Linjeföljarrobot och Konstruktion av ett Grepp och Lift


Nyckelord: Mekatronik, Linjeföljarrobot, Infraröd sensor, Grepp, Lift, Robotik
Acknowledgements

We would like to thank our supervisor Nihad Subasic for the excellent lectures and feedback. Our special gratitude goes to Seshagopalan Thorapalli Muralidharan for all the help, support, and useful information provided throughout this project. We also want to thank Anton Olin for all the help at the prototype centre.
## Contents

1 Introduction ........................................ 1  
  1.1 Background ....................................... 1  
  1.2 Purpose .......................................... 2  
  1.3 Scope ............................................ 2  
  1.4 Method .......................................... 2  

2 Theory .............................................. 3  
  2.1 Optical Theory .................................... 3  
  2.2 Mechanical Theory ............................... 4  
     2.2.1 Gripper ..................................... 4  
     2.2.2 Lifting mechanism ......................... 5  

3 Prototype .......................................... 7  
  3.1 Gripper .......................................... 8  
  3.2 Lift ............................................. 9  
  3.3 Electronics ..................................... 10  
  3.4 Other components ................................ 11  

4 Testing .............................................. 13  
  4.1 Placements of sensors ......................... 13  
  4.2 Tracks .......................................... 15  
  4.3 Gripper and lift ................................ 17  
  4.4 Algorithm ...................................... 18  

5 Results ............................................. 19  

6 Discussion ......................................... 23  
  6.1 Tracks and Sensor Placements ............... 23  
  6.2 Physical design ................................ 24  
  6.3 Electronics .................................... 25  
  6.4 Algorithm ..................................... 25  

7 Conclusion ......................................... 27
# 8 Recommendations and further work

Bibliography

Appendices

## A Data sheets

- A.1 Servo motor SG90
- A.2 Arduino Uno

## B Schematic Diagram

## C Flow charts

- C.1 Case C and D
- C.2 Case E

## D Test Results

- D.1 Base Track
- D.2 Right Curves
- D.3 Left Curves
- D.4 Combined Tracks

## E Code

- E.1 Code case A and B
- E.2 Code case C and D
- E.3 Code case E
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>IR sensor detecting black and light coloured surface [6]</td>
<td>4</td>
</tr>
<tr>
<td>2.2</td>
<td>Graphical representation of how servo-electric grippers work [9]</td>
<td>5</td>
</tr>
<tr>
<td>3.1</td>
<td>Overall CAD view of complete robot with the gripper, lift and Arduino.</td>
<td>7</td>
</tr>
<tr>
<td>3.2</td>
<td>The gripper made in Solid Edge.</td>
<td>8</td>
</tr>
<tr>
<td>3.3</td>
<td>The gripper and lift made in Solid Edge.</td>
<td>9</td>
</tr>
<tr>
<td>3.4</td>
<td>Schematic diagram for the motors. Made in KiCad</td>
<td>10</td>
</tr>
<tr>
<td>3.5</td>
<td>Schematic diagram for the sensors. Made in KiCad</td>
<td>11</td>
</tr>
<tr>
<td>4.1</td>
<td>Test cases A and B for placements of sensors. Made in Solid Edge.</td>
<td>14</td>
</tr>
<tr>
<td>4.2</td>
<td>Test cases C and D for placements of sensors. Made in Solid Edge.</td>
<td>14</td>
</tr>
<tr>
<td>4.3</td>
<td>Test case E for placements of sensors. Made in Solid Edge.</td>
<td>15</td>
</tr>
<tr>
<td>4.4</td>
<td>Reference track. Made in PowerPoint.</td>
<td>15</td>
</tr>
<tr>
<td>4.5</td>
<td>Right turn tracks. Made in PowerPoint.</td>
<td>16</td>
</tr>
<tr>
<td>4.6</td>
<td>Left turn tracks. Made in PowerPoint.</td>
<td>16</td>
</tr>
<tr>
<td>4.7</td>
<td>Combined left and right turn tracks. Made in PowerPoint.</td>
<td>17</td>
</tr>
<tr>
<td>4.8</td>
<td>Flow chart for cases A and B. Made in draw.io</td>
<td>18</td>
</tr>
<tr>
<td>5.1</td>
<td>The finished prototype of Pac-King on track.</td>
<td>19</td>
</tr>
<tr>
<td>5.2</td>
<td>Mean, minimum, and maximum time for all tracks. Made in Excel.</td>
<td>20</td>
</tr>
<tr>
<td>5.3</td>
<td>Mean, minimum, and maximum time for left, right, and base tracks.</td>
<td>21</td>
</tr>
<tr>
<td>B.1</td>
<td>Complete schematic diagram. Made in KiCad.</td>
<td>40</td>
</tr>
<tr>
<td>C.1</td>
<td>Flow chart for cases C and D. Made in draw.io</td>
<td>42</td>
</tr>
<tr>
<td>C.2</td>
<td>Flow chart for cases E. Made in draw.io</td>
<td>43</td>
</tr>
<tr>
<td>D.1</td>
<td>Mean, minimum, and maximum time for base track. Made in Excel.</td>
<td>45</td>
</tr>
<tr>
<td>D.2</td>
<td>Mean, minimum, and maximum time for right curve. Made in Excel.</td>
<td>46</td>
</tr>
<tr>
<td>D.3</td>
<td>Mean, minimum, and maximum time for left curves. Made in Excel.</td>
<td>47</td>
</tr>
<tr>
<td>D.4</td>
<td>Mean, minimum, and maximum time for combined tracks. Made in Excel.</td>
<td>48</td>
</tr>
</tbody>
</table>
## List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC</td>
<td>Direct Current</td>
<td>9</td>
</tr>
<tr>
<td>IR</td>
<td>Infrared</td>
<td>2</td>
</tr>
<tr>
<td>IR LED</td>
<td>Infrared Light Emitting Diode</td>
<td>3</td>
</tr>
<tr>
<td>LFR</td>
<td>Line-following robot</td>
<td>1</td>
</tr>
<tr>
<td>P-I-N</td>
<td>Positive-Intrinsic-Negative</td>
<td>3</td>
</tr>
<tr>
<td>P-N</td>
<td>Positive-Negative</td>
<td>3</td>
</tr>
<tr>
<td>PID</td>
<td>Proportional-Integral-Derivative</td>
<td>26</td>
</tr>
<tr>
<td>PLA</td>
<td>Polylactic acid</td>
<td>8</td>
</tr>
<tr>
<td>PWM</td>
<td>Pulse-Width Modulation</td>
<td>10</td>
</tr>
<tr>
<td>SEK</td>
<td>Swedish Kronor</td>
<td>2</td>
</tr>
<tr>
<td>V</td>
<td>Volts</td>
<td>10</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The use for automation today is increasing rapidly, not only industrially but also in every-day life. One of the simplest automated movements is gripping and lifting. Gripping and lifting objects can be the basis of many operations, especially in assembly lines in industrial settings [1]. Combining this with the ability to recognise and follow a line creates the foundation for a very versatile robot.

1.1 Background

Mechatronics encompasses the study of mechanics, electronics, and programming. Gaining an understanding of the basic movements and components that can be utilised in mechatronics is very important, especially to gain greater insight into the possibilities it creates for the future of automation. Automation involves creating processes that can be self-functioning and independent. Such automation can have an indefinite number of purposes, whether it is to make operations safer, faster, more flexible, or to remove the need for human work [1]. By creating a prototype that can mimic the simple movements used in many automation mechanisms, namely gripping and lifting, this thesis seeks to explore the mechatronics behind such processes to gain a deeper comprehension of them.

Aside from understanding the basics of this engineering field, it is also important to gain an awareness of the implications of it. A simple line following robot (LFR) can be applied in a multitude of real-world scenarios, such as the organisation and effectivisation of industry, health care [2], entertainment [3], as well as domestic and commercial life. In every-day life, a LFR robot could assist with carrying groceries through shopping centres or aiding the elderly or blind by providing support and guidance. The technology could also be applied to self-driving cars that follow lines in the roads, or warehouse robots tasked with gripping and lifting objects.
1.2 Purpose

The objective of this project was to create a robot that could follow a line, as well as grip and lift packages to bring along. The purpose was to gain an understanding of design, programming, and mounting when creating such a robot. This thesis also strives to answer some fundamental research questions:

- What is the most effective way to utilise infrared (IR) sensors to follow a line regarding the number of sensors and their placements?
- How reliant are self-built IR sensors in this application?
- How can a working lift and gripper be created with simplicity?

1.3 Scope

The budget of 1000 Swedish Kronor (SEK) limited the possibilities of construction to be a small prototype with relatively cheap components. The scope did not include optimisation of steering, power supply or aesthetic look. It did include the positioning of sensors as well as the number of sensors used. The scope of the code was to have a functioning code without any obvious unnecessary commands.

The package that was used to test the robot’s grip was simplified to a 3x5x5cm cuboid constructed out of paper. The focus was to test the placement of sensors and create a working grip and lift. The tests were conducted with essentially the same code, only adjusted depending on the number of sensors being used, see Chapter 4 and Appendix C.

The motors were chosen only through considering whether they were functional or not, which was determined by testing them. No calculations were made with regards to the electronics or mechanics since the focus of this project was the IR sensors, grip, and lift.

1.4 Method

The initial phase of this project was to formulate the requirements of the problem, namely to determine the requirements of how the robot should look, what features it should have and what tasks it should be able to perform. Then, initial sketches and computer-aided design (CAD) models were made using Solid Edge [4]. The following phase was to create a prototype that could fulfil the project goals: to follow a line, grip, and lift packages. This involved the creation of a basic physical design and making IR sensors. Once this was achieved, an iterative process of programming, optimising the construction, and developing the prototype started in order to achieve the project’s remaining goals, namely to test the different placements of sensors and control of the motors and servo. Lastly, the tests were conducted and all data was visualised in graphs to be analysed.
Chapter 2

Theory

To achieve this project’s objectives, some theory first needed to be explored. The first and most fundamental theory of the project was the theory behind following a line, which includes IR sensors and their placement. Secondly, the mechanics behind a gripping tool and lift was explored. The basic idea of a LFR is to have a contrast between the line to be followed and the line’s surroundings. Sensors can then be placed over the line to always detect it, or on either side of the line to never detect it. Whenever a sensor detects a change it means that the line turns and the robot has to drive accordingly. In this project the latter was used.

2.1 Optical Theory

The most common way to recognise a line in robotics is by photo diodes. Photo diodes can either be positive-negative (P-N) diodes or positive-intrinsic-negative (P-I-N) diodes. P-N photo diodes consist of P-type and N-type semiconductor materials fused together, creating a P-N junction. If this junction receives light (photons), electrons are excited and move to the N-layer which acts as a cathode. This movement of electrons produces a current flow called the photocurrent and can be measured by a microcontroller. In order to increase the photocurrent, an intrinsic layer, or I-layer, can be added between the P- and N-layers, resulting in a P-I-N photodiode. This intrinsic layer expands the ‘active region’, where photons from the light source excite the electrons in the diode, so that more photocurrent is generated [5].

An IR sensor has both an infrared light emitting diode (IR LED) as a transmitter and a photo diode as a receiver. Together they make up a sensor that allows for the detection of black and white surfaces, see Figure 2.1.
CHAPTER 2. THEORY

Figure 2.1: IR sensor detecting black and light coloured surface [6].

The IR LED emits a light with the same wavelength that the photo diode is most sensitive for. This means that the IR LED functions as the sensor’s own light source, making it relatively stable and unaffected by changes in room lighting. When the sensor is pointing at a light surface, see Figure 2.1 (top), all or almost all light is reflected and enters the photo diode. The light increases the photocurrent and gives a higher reading on the analog input on the microcontroller. If the sensor is pointing at a black surface, see Figure 2.1 (bottom), all light from the IR LED is absorbed by the black surface and no light enters the photo diode. The measured photocurrent is smaller than when the sensor is pointed at a light surface [1].

2.2 Mechanical Theory

The prototype necessary to reach the thesis goals required several mechanically complex components. This section will cover the theory behind these mechanical aspects, namely the gripping and lifting mechanisms.

2.2.1 Gripper

A variety of gripper constructions exist for a range of mechanical purposes, such as vacuum, pneumatic, hydraulic, and servo-electric grippers [7]. A robotic gripper imitates the movement of a human hand by gripping and releasing objects using motors, gears, and links instead of muscles and joints [8]. In a servo-electric gripper, this is achieved through a servo-electric motor that controls the gripper’s movements. An input signal from a control unit (such as an Arduino Uno) is sent to the servo. In response to this command the motor shaft will provide a force, velocity, or a rotation to a specific location, depending on the signal. This mechanical output is transferred to the gripper’s gears which in turn transform the rotational movements into linear movements in the grippers ‘fingers’, opening or closing the grip [9]. An illustration of this process can be seen in Figure 2.2.
2.2. MECHANICAL THEORY

Within the scope of servo-electric grippers there are still a range of appearances and functions to decide between. Parallel grippers have a motion parallel to the grippers body and can provide greater accuracy compared to angular grippers. The fingers of angular grippers pivot around a single point, which can be more ideal for compact spaces. Grippers can also have different numbers of fingers depending on how much specialisation is needed and how much contact the gripper needs to have with the object it is handling [8].

2.2.2 Lifting mechanism

Lifting mechanisms in robotics are designed to move objects vertically. This can be done through different systems such as rotating joints, elevators, or linkages, all with varying degrees of freedom. Elevators in particular only have one degree of freedom, or one direction of motion: up and down. This allows for the object to maintain its orientation in space while varying the distance from the robot body [10].

A rack and pinion system is one way to achieve this vertical motion by converting rotational movement into linear. A motor is attached to a gear, or pinion. As the motor turns the pinion, the flat gear with straight teeth (rack) slides vertically [11]. By connecting this type of linear actuator to the robot’s chassis, a lifting mechanism can be created.
Chapter 3

Prototype

The prototype was made with flexibility in mind. The tests required the photo diodes to be placed in a variety of locations. Therefore, the chassis was designed to make the placement changes easy. The chassis also had several holes in various shapes and sizes to accommodate the wiring between components. Most of the components were 3D-printed out of plastic materials or laser-cut out of acrylic. The back wheels were controlled by one DC-motor each, connected to the Arduino via H-bridges. An overall CAD view of the complete robot is shown in Figure 3.1.

Figure 3.1: Overall CAD view of complete robot with the gripper, lift and Arduino. Made in Solid Edge.
3.1 Gripper

An angular-style gripper was chosen, as seen in Figure 3.2. Considering the simplicity of the packages there was no need for a more complicated gripper. The angular gripper was relatively simple to produce and satisfied the needs for this project. Because of the packages’ simple shape only two jaws were needed [8].

The servo motor connected to the first, smallest gear through an attachment that had a key, transferring momentum. The servo’s shaft turned the small gear, which in turn rotated the two larger gears, opening the grip. The attachment at the bottom of the gripper was fastened to the lift’s carrier with screws.

A servo motor was chosen for the gripper as it provided precise angular control. All gripper components were laser-cut from acrylic, except for attachments for the lift, which were 3D-printed from polylactic acid (PLA) material.

Figure 3.2: The gripper made in Solid Edge.
3.2 Lift

The lift, seen in Figure 3.3, transformed a rotating motion from the direct current (DC) motor to a linear lifting motion. This was achieved through a set of parallel pinions and a rack. The mounting holes of the gears had a key that allowed the motor’s shaft to effectively transfer momentum. The rack gear attached to a bottom plate that was screwed onto the chassis.

Figure 3.3: The gripper and lift made in Solid Edge.

The back of the rack was extended to create ‘wings’ that fit into the tracks of a carrier so that the carrier could slide up and down the rack. The carrier’s purpose was twofold, it created a support so that the DC motor could be fastened with screws and also allowed the gripper to be fixed on the back.

A DC motor was chosen for the lift to provide sufficient torque to carry the weight of the motor itself, the carrier, and the gripper. All lift components were 3D-printed using PLA material, except the gears which were laser-cut from acrylic.
3.3 Electronics

The electronic components that were used were three DC motors, three H-bridges, a power supply and a servo motor. An Arduino Uno was used as a microcontroller and was programmed through a computer. Four IR sensors were made as well using resistors, photo diodes and IR LEDs.

The servo motor was of type SG90 and was used to control the gripper. It had a turning radius of $120^\circ$ that allowed for the gripping and releasing of packages (Appendix 7.1). The DC motors operated on 9 Volts (V) and the servo on 5 V. The H-bridges were of type L9997 and allowed a power supply of 9 V to the DC motors whilst controlling them with pulse-width modulation (PWM) of maximum 5 V from the microcontroller. The schematic diagram for the motors is shown in Figure 3.4.

![Figure 3.4: Schematic diagram for the motors. Made in KiCad.](image)
The sensors were made using one photo diode, one IR LED, one 150 Ohm resistor and one 10 000 Ohm resistor. Four sensors where then connected in parallel to +5 V and ground. The photo diodes that were used had an operating voltage of 1,2-1,3 V and their maximum sensitivity wavelength was 940 nanometer. The IR LEDs’ operating voltage was 1,5-1,6 V and had a maximum current of 120 milliampere. The IR LEDs emitted light of wavelength 940 nanometers [12]. The schematic diagram for the sensors is shown in Figure 3.5.

![Figure 3.5: Schematic diagram for the sensors. Made in KiCad](image)

### 3.4 Other components

All components were mounted on the chassis. The chassis was laser-cut from acrylic sheets. The back wheels were 3D-printed from PolyFlex material and attached directly to the DC motors for optimal torque transferal. The DC motors were fastened with holders that screwed on to the bottom plate of the chassis and pressed the motors in place. Since no motors were needed at the front of the prototype, a ball transfer unit was used instead of standard wheels since this allowed for easy omnidirectional movement. This unit was also affixed to the chassis through an attachment that could be screwed on. The bottom and top plates of the chassis were held up through six 3D printed support structures with threaded rods through. The L9997 H-bridges and Arduino Uno were also attached through screws.

By using screws to attach all components the prototype could easily be assembled and disassembled so that elements could be replaced or improved upon.
Chapter 4

Testing

Tests were conducted to evaluate this report’s thesis. Throughout construction, simple function tests were administered to ensure that fundamental features were functional, such as opening and closing the grip, as well as control of the lift and motors. The most important tests included the placement of the diodes. To be able to determine if the robot was on track at least two sensors were needed [13]. Five different placements of four sensors together with 15 different tracks were used to evaluate the thesis. Each configuration was tested ten times.

4.1 Placements of sensors

Five different arrangements of four transmitter and receiver pairs were tested with placements according to Figure 4.1-4.3. In the figures, the coloured boxes (green, yellow, blue, and orange) represent one transmitter and receiver, namely one sensor. The sensors themselves were not coloured, the names were only used to differentiate between the sensors. The effects were evaluated by time and performance through a series of tracks, see Chapter 4.2.

Cases A and B each had only two sensors. In case A the sensors were placed further apart than in case B, see Figure 4.1. Cases C and D, Figure 4.2, tested four sensors simultaneously with two on a front row and two on a back row. Similarly to cases A and B, the sensors are placed nearer in case C and further apart in case D. Case E tests four sensors placed on the same front row, see Figure 4.3.
CHAPTER 4. TESTING

(a) Case A  
(b) Case B

Figure 4.1: Test cases A and B for placements of sensors. Made in Solid Edge.

(a) Case C  
(b) Case D

Figure 4.2: Test cases C and D for placements of sensors. Made in Solid Edge.
4.2 Tracks

The robot’s line following abilities were tested through a variety of tracks with varying turn angles. Cases A-E were tested by timing them ten times each on every track for improved result accuracy and reliability. A straight reference track, seen in Figure 4.4 was also created with a length of 100 cm. The same total length was used on all other tracks. This was used to created a baseline for all cases.

There were three general tracks tested: right turns, left turns, and combined left and right turns. In each of these categories the same angles of turns were tested, namely 22.5°, 45°, 67.5°, and 90°.

The tracks with right turns, seen in Figure 4.5, first had 50 cm of straight track, then the turn, and another 50 cm straight track before stopping.
The left turn tracks in Figure 4.6 had the same dimensions as the right turn tracks.

The combined left and right turn tracks started with 20 cm straight track, then a right turn for 20 cm, straight track for 20 cm, left turn for 20 cm, and a final straight track for 20 cm before the robot stopped. These tracks can be seen in Figure 4.7.
4.3. GRIPPER AND LIFT

Testing the gripper and lift was straightforward. The robot was required to grip a package, lift it up, bring it, and set it down in a controlled manner. The package’s dimensions were 3x5x5 cm. It was constructed out of paper and thus lightweight.

For all performed track tests the robot not only followed the line, but also gripped and lifted a single package. This tested whether the prototype could achieve the thesis goals regardless of sensor placement. The thicker perpendicular line at the top of the tracks represent the track’s end where the robot stopped, lowered the package and released it.
4.4 Algorithm

The same basic algorithm was used for all cases. The robot turned left if any of the left hand side sensors detected black and it turned right if any of the right hand side sensors detected black. Depending on which case was tested, the motors were controlled to drive at different speeds [14]. The flowchart for cases A and B is shown in Figure 4.8. Flowcharts for cases C-E can be found in Appendix C.

![Flowchart for cases A and B](image)

Figure 4.8: Flow chart for cases A and B. Made in draw.io.
Chapter 5

Results

The project succeeded in creating a robot that follows a line and has gripping and lifting abilities. The final prototype can be seen in Figure 5.1.

Figure 5.1: The finished prototype of Pac-King on track.
CHAPTER 5. RESULTS

The outcomes from the tests are presented in Average-Maximum-Minimum charts, with the cases presented on the x-axis and time in seconds on the y-axis. The orange line represents the maximum time value of the tests and the blue line represents the minimum. The black rhombus shows the mean time of the tests. The test results in their entirety are presented in Appendix D.

Figure 5.2 below shows the total results from all performed tests on all tracks. This includes the base track as well as the left, right, and combined tracks with 22.5°, 45°, 67.5°, and 90° turns.

![Collected Data All Tracks](image)

Figure 5.2: Mean, minimum, and maximum time for all tracks. Made in Excel.

It can be seen here that case A had the fastest minimum time overall (although Case C had a very similar result), case C had the fastest average time, case E had the slowest average time, and case D the slowest maximum time. Case B had the smallest distribution and case D had the largest.

It can be noted from the results, Appendix D, that the combined left and right tracks had significantly slower times than the other tracks. Because of this, the graph in Figure 5.3 was made to exclude the combined tracks and include only the base track, left and right tracks with varying angles. This was done to create results without extreme outliers.
In this representation of results, differences in distribution are less extreme, which is especially evident for case D. Here case A has the fastest minimum time, and case C was a very close second. Case C had the fastest average time and case A had again a very similar time to case C. Case E had the slowest maximum time.

On each track the gripper and lift were tested on the prototype. At the beginning of the track the robot had to grip the package, lift it up, run through the track, and lower the package and release it as it reached the stop line. The prototype was successful in doing this on each track.
Chapter 6

Discussion

This chapter will discuss the thesis results, physical design of the prototype, the electronics that were used, and the robot’s algorithm.

6.1 Tracks and Sensor Placements

The different cases had varying results in all of the performed tests, see Appendix D, however the compiled data in Figures 5.2 and 5.3 make it easier to decipher which cases were favourable.

From the results, it seemed that case D or E had worse sensor placement. In Figure 5.2, case D has significantly larger distribution compared to the other cases, indicating inconsistencies in its performance. Both cases D and E had the slowest maximum, minimum, and average times overall. With the combined tracks excluded, as seen in Figure 5.3, it can be noted that cases D and E again had the slowest times, although their distributions improved slightly. From what could be seen during testing, these cases were unstable whilst driving and would often miss turns completely and therefore needed to be retested. Similarly, it was also observed that case B was unstable during testing and often missed the track completely or had ‘choppy’ runs. A possible explanation for this could be that due to the sensors in cases B, D, and E being placed in close proximity to each other interference was created. Through visual inspection, it seemed that cases A and C were more stable overall, most likely due to the sensors being placed further apart. When looking at the overall results, Figure 5.2, case B had the smallest and most reliable distribution, in spite of the visual observations of it being more erratic. Case C had the overall fastest average time and Case A and C had the fastest average minimum time. When excluding the combined tracks in Figure 5.3, cases A and C have very similar results: they both had the fastest minimum, maximum, and average times.

All cases experienced difficulties in the tighter turns, which is evident in the results for 67.5° and 90° turns, see Appendix D. This could be due to the prototype being too front-heavy or the algorithm being too slow, see Sections 6.2 and 6.3 for further discussion about this. For the prototype to be able to execute these curves,
additional black lines had to be added to the track to make the turn smoother. For the prototype to complete the 90° combined track, the algorithm had to be changed to reduce both the forward and turning power. Since this was the most complicated route for the prototype to complete, the robot had to go slower regardless of which cases were being tested in order to register the turn.

6.2 Physical design

The physical design of the prototype had both advantages and disadvantages. The gripper and lifting system were objectively well-designed and functioning, however they made the prototype very forward-heavy. This resulted in the robot struggling to complete tight turns, such as 67.5° and 90°, as well as combined tracks. With a front-heavy construction, the back of the prototype would become lighter, resulting in the steering back wheels not gripping onto the surface sufficiently enough to turn in time. In some instances, the prototype required human assistance to complete these tracks, for example with a finger pushing down on the back of the chassis to balance it. This could have resulted in interference and inconsistencies in the results, since uneven pressure was administered to the robot. The weight of the DC motor on the lift also skewed the gripper’s angle, resulting in it not being consistently perpendicular to the chassis. Although this did not affect the robot’s ability to pick up packages, it was not optimal out of a design perspective. It could also have had an impact on right and left turns, as the motor applied more weight on the right hand side of the prototype, making the centre of mass slightly acentric. This might be the reason for the faster results in right curves compared to left curves, see Appendix D.

Further problems arose with the prototype’s wheels. The ball transfer unit was placed centrally on the prototype and since the black lines on the track were composed of electrical tape, the frontal wheel often hooked on the tape’s edges. This resulted in some of the runs becoming rough. The back wheels that connected to the DC motors also had to be redesigned several times. At first, manufactured tires were used, which can be seen in Figure 3.1. The axle hole in the tire’s centre was too large to fit the motor’s shaft rigidly, resulting in a non-centred and flawed torque transferal from the motor. Thus, better-fitting wheels were made by 3D-printing with PolyFlex material. These can be seen in Figure 5.1. The 3D-printed wheels were also made smaller than the manufactured, both the diameter and width, to counter the front heavy issue.

Due to the sensors needing to be moved around for testing, they were simply attached to the chassis with electrical tape. This allowed for easy change of placement but did not provide a sustainable solution for the further use of the prototype. Instead, more durable holders could be made to fasten the sensors to the chassis. Even if the tape served the purpose for these tests, something more reliable and precise would have been preferable. The tape had a tendency to stretch during tests which might have resulted in different focus for the sensors.
Altogether however, the physical design of the prototype was satisfactory for this project. All components were durable enough to last through several weeks of testing and improvements, as well as flexible enough for the prototype to be disassembled when it needed to be.

6.3 Electronics

The sensors that were made worked reliably only for short intervals at a time. At first, all sensors had the same threshold between black and white reading. After a while it seemed like they started to take in more interference and noise as the threshold values started to change. The green sensor especially could ‘run off’, going up in readings rapidly and without any obvious reason. Due to this the threshold values in the code had to be changed during the tests. When the readings became too unstable all power supply had to be removed from the robot and some time had to be taken to allow all current to run out of the system. This reset the sensors and allowed for more testing. The issue with unstable sensors might have had a significant impact on the tests, as all the tests were dependent on the sensors. The source of the interference was not found, however there is a possibility that it came from the other sensors, the motors and servo, or both. Because of the increase of disturbance it could have come from inductance or capacitance in the circuit, which would also explain why it disappeared after a reset. To solve this issue one could use manufactured sensors instead. Manufactured sensors usually have a more protected circuit and connections which would protect them from possible interference from inductance and capacitance. The circuits of the constructed sensors were completely open, allowing for sensitivity regarding air humidity, dust, and other contaminants.

The motors that were used for the wheels were the same type but did not output the exact same torque and speed. Due to this, the robot did not drive exactly straight when it was supposed to. However, because of the feedback from the sensors the robot was able to adjust itself to keep on track. This could have been taken into account in the code but because of the feedback it was not necessary. The motors were sufficient but it would have been preferable to have more torque and slower rotation. This would allow the robot to go slower, giving more time for the sensors to read the ground and the code to execute, making sure that the robot did not miss a turn. A lower forward power was tested, but it was too low for the motors to rotate.

6.4 Algorithm

For the simple right and left curve tracks the algorithm worked well. Sometimes the robot missed a curve and continued straight. This could be because of the sensors’ misreadings, too high speed, or too slow code. When the robot missed a curve it helped to put a little pressure on the back of the chassis, making it go slower and putting more weight over the back wheels. This also increased the time interval for
the robot to react on the ground, allowing it to make the turn. Whether it was the slow code or too high motor speed that was the issue is hard to know since they go hand in hand. A better balance between motor speed and algorithm should be striven for. The algorithm was sufficient for testing and its simplicity made it easy to troubleshoot. However, the drive was fitful due to the distinct jumps of motor power: turning the motor off and on multiple times instead of smoothly adjusting the power. This also resulted in unstable behaviour since if one motor was on for a little bit too long the robot would oscillate between trying to counter the first fitful turn, sometimes resulting in it missing the line completely. This could be solved with a more efficient and thorough algorithm. For example, a proportional-integral-derivative (PID) controller could be used. This would remove the fitful drive and possibly make the tests more reliant.
Chapter 7

Conclusion

This project’s objectives to create a LFR, lift, and grip has been achieved. It has provided insight in the ways of designing, programming and mounting a LFR, lift and grip. This report has shown that the best placement for IR sensors is case A or C, see Figure 4.1 (a) and 4.2 (a), both indicating that too near placement is worse for reliability and time. The IR sensors created were sensitive to disturbance and are not recommended to be used in further work without any adjustment. However, they did work with some precautions. Overall the project has succeeded to answer the thesis questions.
Chapter 8

Recommendations and further work

For further work on this thesis more reliable tests should be conducted. To achieve this the following should be adjusted:

- Change the front wheel to two front wheels to remove the issue with the tape.
- Adjust the centre of mass to remove the prototype’s front-heaviness and differences in right and left curves.
- Use manufactured sensors, or make the sensors more resilient to interference.
- Change the sensor mounting principle from tape to something more reliant, for example a 3D-printed attachment.
- Improve the algorithm with a model and perhaps use a PID controller.
- Change the wheel motors to motors with more torque and slower speed.
Bibliography


Appendix A

Data sheets

A.1 Servo motor SG90

Tiny and lightweight with high output power. Servo can rotate approximately 180 degrees (90 in each direction), and works just like the standard kinds but smaller. You can use any servo code, hardware or library to control these servos. Good for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. It comes with a 3 horns (arms) and hardware.

<table>
<thead>
<tr>
<th>Dimensions &amp; Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>A (mm) : 32</td>
</tr>
<tr>
<td>B (mm) : 23</td>
</tr>
<tr>
<td>C (mm) : 28.5</td>
</tr>
<tr>
<td>D (mm) : 12</td>
</tr>
<tr>
<td>E (mm) : 32</td>
</tr>
<tr>
<td>F (mm) : 19.5</td>
</tr>
<tr>
<td>Speed (sec) : 0.1</td>
</tr>
<tr>
<td>Torque (kg-cm) : 2.5</td>
</tr>
<tr>
<td>Weight (g) : 14.7</td>
</tr>
<tr>
<td>Voltage : 4.8 - 6</td>
</tr>
</tbody>
</table>

Position "0" (1.5 ms pulse) is middle, "90" (~2ms pulse) is middle, is all the way to the right, "-90" (~1ms pulse) is all the way to the left.
A.2 Arduino Uno

Overview

The Arduino Uno is a microcontroller board based on the ATmega328 (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega16U2 (Atmega8U2 up to version R2) programmed as a USB-to-serial converter.

Revision 2 of the Uno board has a resistor pulling the 8U2 HWB line to ground, making it easier to put into DFU mode.

Revision 3 of the board has the following new features:

- 1.0 pinout: added SDA and SCL pins that are near to the AREF pin and two other new pins placed near to the RESET pin, the IOREF that allow the shields to adapt to the voltage provided from the board. In future, shields will be compatible both with the board that use the AVR, which operate with 5V and with the Arduino Due that operate with 3.3V. The second one is a not connected pin, that is reserved for future purposes.
- Stronger RESET circuit.
- Atmega 16U2 replace the 8U2.

"Uno" means one in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduino, moving forward. The Uno is the latest in a series of USB Arduino boards, and the reference model for the Arduino platform; for a comparison with previous versions, see the index of Arduino boards.

Summary

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATmega328</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating Voltage</td>
<td>5V</td>
</tr>
<tr>
<td>Input Voltage (recommended)</td>
<td>7-12V</td>
</tr>
</tbody>
</table>
A.2. ARDUINO UNO

Input Voltage (limits) 6-20V
Digital I/O Pins 14 (of which 6 provide PWM output)
Analog Input Pins 6
DC Current per I/O Pin 40 mA
DC Current for 3.3V Pin 50 mA
Flash Memory 32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM 2 KB (ATmega328)
EEPROM 1 KB (ATmega328)
Clock Speed 16 MHz

Schematic & Reference Design

EAGLE files: arduino-uno-Rev3-reference-design.zip (NOTE: works with Eagle 6.0 and newer)
Schematic: arduino-uno-Rev3-schematic.pdf
Note: The Arduino reference design can use an Atmega8, 168, or 328, Current models use an ATmega328, but an Atmega8 is shown in the schematic for reference. The pin configuration is identical on all three processors.

Power

The Arduino Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically.
External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.
The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.
The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it’s using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V.** This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7-12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. We don’t advise it.
- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND.** Ground pins.

Memory

The ATmega328 has 32 KB (with 0.5 KB used for the bootloader). It also has 2 KB of SRAM and 1 KB of EEPROM (which can be read and written with the EEPROM library).

Input and Output

Each of the 14 digital pins on the Uno can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- **External Interrupts: 2 and 3.** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- **PWM: 3, 5, 6, 9, 10, and 11.** Provide 8-bit PWM output with the analogWrite() function.
SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication using the SPI library.

LED: 13. There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

The Uno has 6 analog inputs, labeled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though it is possible to change the upper end of their range using the AREF pin and the analogReference() function. Additionally, some pins have specialized functionality:

- TWI: A4 or SDA pin and A5 or SCL pin. Support TWI communication using the Wire library.

There are a couple of other pins on the board:

- AREF. Reference voltage for the analog inputs. Used with analogReference().
- Reset. Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

See also the mapping between Arduino pins and ATmega328 ports. The mapping for the Atmega8, 168, and 328 is identical.

Communication

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega16U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The '16U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, on Windows, a .inf file is required. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A SoftwareSerial library allows for serial communication on any of the Uno's digital pins. The ATmega328 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation for details. For SPI communication, use the SPI library.

Programming

The Arduino Uno can be programmed with the Arduino software (download). Select "Arduino Uno from the Tools > Board menu (according to the microcontroller on your board). For details, see the reference and tutorials.

The ATmega328 on the Arduino Uno comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files).

You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see these instructions for details.

The ATmega16U2 (or 8U2 in the rev1 and rev2 boards) firmware source code is available. The ATmega16U2/8U2 is loaded with a DFU bootloader, which can be activated by:

- On Rev1 boards: connecting the solder jumper on the back of the board (near the map of Italy) and then resetting the 8U2.
- On Rev2 or later boards: there is a resistor that pulling the 8U2/16U2 HWB line to ground, making it easier to put into DFU mode.

You can then use Atmel's FLIP software (Windows) or the DFU programmer (Mac OS X and Linux) to load a new firmware. Or you can use the ISP header with an external programmer (overwriting the DFU bootloader). See this user-contributed tutorial for more information.

Automatic (Software) Reset
A.2. ARDUINO UNO

Rather than requiring a physical press of the reset button before an upload, the Arduino Uno is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2/16U2 is connected to the reset line of the ATmega328 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload. This setup has other implications. When the Uno is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the Uno. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.

The Uno contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It’s labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see this forum thread for details.

**USB Overcurrent Protection**

The Arduino Uno has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

**Physical Characteristics**

The maximum length and width of the Uno PCB are 2.7 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Four screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.
Appendix B

Schematic Diagram
Figure B.1: Complete schematic diagram. Made in KiCad.
Appendix C

Flow charts

C.1 Case C and D

Figure C.1: Flow chart for cases C and D. Made in draw.io
C.2 Case E

Figure C.2: Flow chart for cases E. Made in draw.io
Appendix D

Test Results

D.1 Base Track

Figure D.1: Mean, minimum, and maximum time for base track. Made in Excel.
D.2 Right Curves

(a) Right Curve 22.5°

(b) Right Curve 45°

(c) Right Curve 67.5°

(d) Right Curve 90°

Figure D.2: Mean, minimum, and maximum time for right curve. Made in Excel.
D.3  LEFT CURVES

D.3  Left Curves

Figure D.3: Mean, minimum, and maximum time for left curves. Made in Excel.
D.4 Combined Tracks

Figure D.4: Mean, minimum, and maximum time for combined tracks. Made in Excel.
Appendix E

Code

There are three different codes, one for each flow chart. All three uses the same eight functions: LEFT, RIGHT, STOP, GRIP, RELEASE, LIFTUP, LIFTDOWN and STRAIGHT. The difference between the codes is the logic that determines when to turn which is visualised in Appendix C and Figure 4.8.
APPENDIX E. CODE

E.1 Code case A and B

```c
/* Pac King
   * Author: Sofia Jönsson
   * Course: MF133X
   * School: KTH
   * Date: 2019-06-07
   * Programming code for bachelor degree project
   * in Mechatronics spring 2019
   * by Sofia Jönsson and Tess Antonsson
   * Case A and B
   */

/*************************************/
// Include the libraries for the servo.
/*************************************/
#include "Servo.h"
#define pinServo 3

/*************************************/
// Initiates the variables
/*************************************/
int hogerfram=5;
int hogerbak=9;
int vansterfram=6;
int vansterbak=10;
int liftupp = 11;
int liftner = 12;
it IRpinGul = A2;
it IRpinGron = A1;
it IRgul;
it IRgron;
int fwpwr = 80;
int turnpwr = 100;
int backpwr = 0;
int thresh = 15;
int threshgreen = 16;
int pathing;
Servo servo;

/*************************************/
// Initiates the functions.
/*************************************/
void LEFT (void);
void RIGHT (void);
void STOP (void);
void GRIP (void);
void RELEASE (void);
void LIFTUP (void);
void LIFTDOWN (void);
void STRAIGHT (void);

/*************************************/
// Initial setup for the servo and print to serial monitor
/*************************************/
void setup()
{
  servo.attach(pinServo);
  Serial.begin(9600);
}
```
//Loop for the robot to execute.
//Pac-King follows the line until it encounters a stop.
//A stop changes the variable pathing to 0 (false).

void loop()
{
    pathing = 1;
    RELEASE();
    delay(3000);
    GRIP();
    LIFTUP();
    while(pathing)
    {
        IRgron = analogRead(IRpinGron);
        IRgul = analogRead(IRpinGul);
        if((IRgron <= threshgreen) && (IRgul > thresh))
        {
            LEFT();
        } else if((IRgron > threshgreen) && (IRgul <= thresh))
        {
            RIGHT();
        } else if((IRgron <= threshgreen) && (IRgul <= thresh))
        {
            pathing = 0;
            STOP();
            break;
        } else
        {
            STRAIGHT();
        }
    }
    STOP();
    delay(2000);
    LIFTDOWN();
    RELEASE();
}

//Function for driving straight.

void STRAIGHT (void)
{
    analogWrite(vansterbak, 0);
    analogWrite(vansterfram, fwdpwr);
    analogWrite(hogerbak, 0);
    analogWrite(hogerfram, fwdpwr);
}

//Function for turning left.

void LEFT (void)
{
    while(IRgron <= thresh)
    {
        IRgron = analogRead(IRpinGron);
IRgul = analogRead(IRpinGul);
analogWrite(vansterbak, backpwr);
analogWrite(vansterfram, 0);
analogWrite(hogerbak, 0);
analogWrite(hogerfram, turnpwr);
}
delay(50);

void RIGHT (void)
{
while(IRgul <= thresh)
{
  IRgron = analogRead(IRpinGron);
  IRgul = analogRead(IRpinGul);
  analogWrite(vansterbak, 0);
  analogWrite(vansterfram, turnpwr);
  analogWrite(hogerbak, backpwr);
  analogWrite(hogerfram, 0);
}
delay(50);

void GRIP (void)
{
servo.write(120);
delay(500);
}

void RELEASE (void)
{
servo.write(-110);
delay(500);
}

void LIFTUP (void)
{
analogWrite(liftupp, 255);
delay(350);
analogWrite(liftupp, 50);
delay(100);
}

void LIFTDOWN (void)
{
analogWrite(liftupp, 0);
digitalWrite(liftner, HIGH);
delay(350);
digitalWrite(liftner, LOW);
delay(1000);
E.1. CODE CASE A AND B

```c
void STOP (void)
{
    analogWrite(hogerbak, 0);
    analogWrite(hogerfram, 0);
    analogWrite(vansterbak, 0);
    analogWrite(vansterfram, 0);
}
```
APPENDIX E. CODE

E.2 Code case C and D

```c
/* Pac King
 * Author: Sofia Jönsson
 * Course: MF133X
 * School: KTH
 * Date: 2019-06-07
 * Programming code for bachelor degree project
 * in Mechatronics spring 2019
 * by Sofia Jönsson and Tess Antonsson
 * Case A and B
 * *
#include "Servo.h"
define pinServo 3

/******************************/
// Includes the libraries for the servo.
/******************************/
#include <Servo.h>
define pinServo 3

/******************************/
// Initiates the global variables
/******************************/
int hogerfram=5;
int hogerbak=9;
int vansterfram=6;
int vansterbak=10;
int liftupp = 11;
int liftner = 12;
int IRpinGul = A2;
int IRpinGron = A1;
int IRpinOrange = A3;
int IRpinBla = A0;
int IRgul;
int IRgron;
int IRbla;
int IRorange;
int fwdpwr = 80;
int turnpwr = 100;
int backpwr = 0;
int thresh = 15;
int threshor = 14;
int threshgreen = 16;
int pathing;

Servo servo;

/******************************/
// Initiates the functions.
/******************************/
void LEFT (void);
void RIGHT (void);
void STOP (void);
void GRIP (void);
void RELEASE (void);
void LIFTUP (void);
void LIFTDOWN (void);
void STRAIGHT (void);

/******************************/
// Initial setup for the servo and print to serial monitor
/******************************/
void setup()
```

54
E.2. CODE CASE C AND D

```c
61  {
62     servo.attach(pinServo);
63     Serial.begin(9600);
64  }
65  //************************************/
66  //Loop for the robot to execute.
67  //Pac-King follows the line until it encounters a stop.
68  //A stop changes the variable pathing to 0 (false).
69  //************************************/
70  void loop()
71  {
72      pathing = 1;
73      RELEASE();
74      delay(3000);  
75      GRIP();
76      LIFTUP();
77      while(pathing)
78      {
79          IRgron = analogRead(IRpinGron);
80          IRgul = analogRead(IRpinGul);
81          IRbla = analogRead(IRpinBla);
82          IRorange = analogRead(IRpinOrange);
83          if((IRgron <=
84              threshgreen)&&(IRbla>thresh)&&(IRgul>thresh)&&(IRorange>threshor) )
85              {
86                  turnpwr = 100;
87                  backpwr = 0;
88                  LEFT();
89              } else if( (IRgron<=threshgreen)&&(IRbla<=thresh) &&
90                  (IRgul>thresh)&&(IRorange>threshor) )
91              {
92                  turnpwr = 150;
93                  backpwr = 0;
94                  LEFT();
95              } else if((IRgron>threshgreen)&&(IRbla<=thresh) &&
96                  (IRgul>thresh)&&(IRorange>threshor) )
97              {
98                  turnpwr = 150;
99                  backpwr = 0;
100                 LEFT();
101               } else if((IRgul <= thresh) && (IRbla>thresh) && (IRorange>threshor) && (IRgron>threshgreen) )
102               {
103                   turnpwr = 100;
104                   backpwr = 0;
105                     RIGHT();
106                 } else if((IRgul<=thresh) && (IRorange<= threshor) &&
107                     (IRgul>threshgreen) && (IRbla > thresh))
108                 {
109                     turnpwr = 150;
110                     backpwr = 0;
111                        RIGHT();
112                    } else if((IRgul<=thresh) && (IRorange<= threshor) && (IRgul>thresh)
113                          && (IRbla > thresh))
114                 {
115                     turnpwr = 150;
116                     backpwr = 0;
117                        RIGHT();
118                    }
```

55
APPENDIX E. CODE

```c
{ pathing = 0;
  STOP();
  break;
}
else
{
  STRAIGHT();
}
STOP();
delay(2000);
LIFTDOWN();
RELEASE();

}                          //Function for driving straight.
void STRAIGHT (void)
{
  analogWrite(vansterbak, 0);
  analogWrite(vansterfram, fwdpwr);
  analogWrite(hogerbak, 0);
  analogWrite(hogerfram, fwdpwr);
}
                        //Function for turning left.
void LEFT (void)
{
  while(IRgron <= thresh)
  {
    IRgron = analogRead(IRpinGron);
    IRgul = analogRead(IRpinGul);
    analogWrite(vansterbak, backpwr);
    analogWrite(vansterfram, 0);
    analogWrite(hogerbak, 0);
    analogWrite(hogerfram, turnpwr);
  }
  delay(50);
}
                        //Function for turning right.
void RIGHT (void)
{
  while(IRgul <= thresh)
  {
    IRgron = analogRead(IRpinGron);
    IRgul = analogRead(IRpinGul);
    analogWrite(vansterbak, 0);
    analogWrite(vansterfram, turnpwr);
    analogWrite(hogerbak, backpwr);
    analogWrite(hogerfram, 0);
  }
  delay(50);
}
                        //Function for closing the grip.
void closeGrip (void)
{
  analogWrite(vansterbak, 0);
  analogWrite(vansterfram, 0);
  analogWrite(hogerbak, 0);
  analogWrite(hogerfram, 0);
}
```
E.2. CODE CASE C AND D

```c
175   void GRIP (void)
176   {
177       servo.write(120);
178       delay(500);
179   }
180   (){****************************/
181   //Function for releasing the grip.
182   (){******************************/
183   void RELEASE (void)
184   {
185       servo.write(-110);
186       delay(500);
187   }
188   (){******************************/
189   //Function for driving the lift up and hold it.
190   (){******************************/
191   void LIFTUP (void)
192   {
193       analogWrite(liftupp, 255);
194       delay(350);
195       analogWrite(liftupp, 50);
196       delay(100);
197   }
198   (){******************************/
199   //Function for driving the lift down.
200   (){******************************/
201   void LIFTDOWN (void)
202   {
203       analogWrite(liftupp, 0);
204       digitalWrite(liftner, HIGH);
205       delay(350);
206       digitalWrite(liftner, LOW);
207       delay(1000);
208   }
209   (){******************************/
210   //Function for stop.
211   (){******************************/
212   void STOP (void)
213   {
214       analogWrite(hogerbak, 0);
215       analogWrite(hogerfram, 0);
216       analogWrite(vansterbak, 0);
217       analogWrite(vansterfram, 0);
218   }
```
E.3 Code case E

```c
/* Pac King
 * Author: Sofia Jönsson
 * Course: MF133X
 * School: KTH
 * Date: 2019-06-07
 * Programming code for bachelor degree project
 * in Mechatronics spring 2019
 * by Sofia Jönsson and Tess Antonsson
 * Case A and B
 */

#include "Servo.h"
#define pinServo 3

int hogerfram=5;
int hogerbak=9;
int vansterfram=6;
int vansterbak=10;
int liftupp = 11;
int liftner = 12;
int IRpinGul = A2;
int IRpinGron = A1;
int IRpinOrange = A3;
int IRpinBla = A0;
int IRgul;
int IRgron;
int IRbla;
int IRorange;
int fwdpwr = 70;
int turnpwr = 80;
int backpwr = 0;
int thresh = 12;
int pathing;

Servo servo;

void LEFT (void);
void RIGHT (void);
void STOP (void);
void GRIP (void);
void RELEASE (void);
void LIFTUP (void);
void LIFTDOWN (void);
void STRAIGHT (void);

void setup() {
  servo.attach(pinServo);
}
```

APPENDIX E. CODE
E.3. CODE CASE E

```c
Serial.begin(9600);

// Loop for the robot to execute.
// Pac-King follows the line until it encounters a stop.
// A stop changes the variable pathing to 0 (false).

void loop()
{
    pathing = 1;
    RELEASE();
    delay(3000);
    GRIP();
    LIFTUP();
    while(pathing)
    {
        IRgron = analogRead(IRpinGron);
        IRgul = analogRead(IRpinGul);
        IRbla = analogRead(IRpinBla);
        IRorange = analogRead(IRpinOrange);
        if( ((IRbla<=thresh)||(IRgron<=thresh)) && (IRgul>thresh)&& (IRorange > thresh) )
        {
            LEFT();
        }
        else if( (IRgron>thresh)&&(IRbla>thresh) && ((IRgul<=thresh) || (IRorange <= thresh) ))
        {
            RIGHT();
        }
        else if( (IRbla<= thresh) && (IRorange <= thresh) )
        {
            pathing = 0;
            STOP();
            break;
        }
        else
        {
            STRAIGHT();
        }
    }
    STOP();
    delay(2000);
    LIFTDOWN();
    RELEASE();
}

//Function for driving straight.

void STRAIGHT (void)
{
    analogWrite(vansterbak, 0);
    analogWrite(vansterfram, fwdpwr);
    analogWrite(hogerbak, 0);
    analogWrite(hogerfram, fwdpwr);
}

//Function for turning left.

void LEFT (void)
{
    while((IRgron<=thresh)||(IRbla <= thresh))
    {
    }
```
APPENDIX E. CODE

```
119   IRgron = analogRead(IRpinGron);
120   IRgul = analogRead(IRpinGul);
121   IRbla = analogRead(IRpinBla);
122   IRorange = analogRead(IRpinOrange);
123       analogWrite(vansterbak, backpwr);
124       analogWrite(vansterfram, 0);
125       analogWrite(hogerbak, 0);
126       analogWrite(hogerfram, turnpwr);
127   }
128   delay(50);
129 }
130 //********************************************************************************
131 //Function for turning right.
132 //********************************************************************************
133 void RIGHT (void)
134 {
135   while((IRorange<=thresh) || (IRgul <= thresh))
136   {
137     IRgron = analogRead(IRpinGron);
138     IRgul = analogRead(IRpinGul);
139     IRbla = analogRead(IRpinBla);
140     IRorange = analogRead(IRpinOrange);
141       analogWrite(vansterbak, 0);
142       analogWrite(vansterfram, turnpwr);
143       analogWrite(hogerbak, backpwr);
144       analogWrite(hogerfram, 0);
145   }
146   delay(50);
147 }
148 //********************************************************************************
149 //Function for closing the grip.
150 //********************************************************************************
151 void GRIP (void)
152 {
153   servo.write(120);
154   delay(500);
155 }
156 //********************************************************************************
157 //Function for releasing the grip.
158 //********************************************************************************
159 void RELEASE (void)
160 {
161   servo.write(-120);
162   delay(500);
163 }
164 //********************************************************************************
165 //Function for driving the lift up and hold it.
166 //********************************************************************************
167 void LIFTUP (void)
168 {
169   analogWrite(liftupp, 255);
170   delay(350);
171   analogWrite(liftupp, 50);
172   delay(100);
173 }
174 //********************************************************************************
175 //Function for driving the lift down.
176 //********************************************************************************
```

E.3. CODE CASE E

void LIFTDOWN (void)
{
    analogWrite(liftupp, 0);
    digitalWrite(liftner, HIGH);
    delay(350);
    digitalWrite(liftner, LOW);
    delay(1000);
}

//Function for stop.
void STOP (void)
{
    analogWrite(hogerbak, 0);
    analogWrite(hogerfram, 0);
    analogWrite(vansterbak, 0);
    analogWrite(vansterfram, 0);
}