ENVIRONMENT OBSERVATION MODULE FOR THE DECI ZEBRO

Obstacle and Motion Detection using a Laser Rangefinder and Optical Flow

THESIS FOR THE DEGREE OF BACHELOR OF SCIENCE IN ELECTRICAL ENGINEERING PRESENTED BY:

Oxana Oosterlee 4374428
Sjors Peterse 4342208

SUPERVISOR:
Chris Verhoeven

DAILY SUPERVISOR:
Daniël Booms

ADVISORS:
Edwin Hakkennes
Ronald Bos

JUNE 19, 2017

DELFT UNIVERSITY OF TECHNOLOGY

FACULTY OF ELECTRICAL ENGINEERING, MATHEMATICS AND COMPUTER SCIENCE

ELECTRICAL ENGINEERING PROGRAMME

Abstract

A Camera Submodule for the Environment Observation Module for a Zebro robot was designed. It is able to collect data about obstacles to ensure Zebro's safety and send this to the main processor of the module using UART. The two main functions that were implemented are a laser rangefinder based on triangulation and a motion detector based on optical flow.

The laser rangefinder calculates the distance to an obstacle by finding its own laser point in a frame. Testing showed that it is especially accurate at short distances (0-4 cm), and the range can differ between 0.30 to 6m due to environmental light and obstacle color.

The motion detector uses dense optical flow data that is provided by the h264 encoder of the Pi Camera. This data is filtered and moving obstacles are grouped together. In the tests, a moving obstacle was detected every time, but 29% also showed false positives.

Contents

1 Introduction **7**
- 1.1 The Zebro Project . . . 7
- 1.2 The Environment Observation Module . . . 7
- 1.3 State of the Art . . . 8
- 1.4 Thesis Structure . . . 9

2 Programme of Requirements **11**
- 2.1 Environment Observation Module Functional Requirements . . . 11
- 2.2 Thesis Requirements . . . 11
- 2.3 System Requirements . . . 11
- 2.4 Camera Submodule Functional Requirements . . . 12

3 General design **13**
- 3.1 Motion Detection . . . 13
- 3.2 Obstacle Detection . . . 13
- 3.3 Processing and Camera Hardware . . . 14
- 3.4 Detection of QR Codes . . . 14

4 Design of the Laser Rangefinder **15**
- 4.1 Setup . . . 15
- 4.2 Physical Parameter Design . . . 16
- 4.3 Laser Type . . . 16
- 4.4 Software . . . 18

5 Design of Motion Detection **23**
- 5.1 Hardware and Libraries . . . 23
- 5.2 Software . . . 23

6 Module Integration **27**
- 6.1 Software Integration . . . 27
- 6.2 Communication . . . 27
- 6.3 Hardware Integration . . . 28
  - 6.3.1 Power Consumption . . . 29
- 6.4 Physical Integration in Module Enclosure . . . 29

7 Test Results **31**
- 7.1 Validation Laser Rangefinder . . . 31
  - 7.1.1 Accuracy . . . 31
  - 7.1.2 Maximum Range . . . 32
  - 7.1.3 Discussion of results . . . 34
- 7.2 Validation QR-code Detector . . . 34
  - 7.2.1 Discussion of Results . . . 34
- 7.3 Validation Motion Detection . . . 35
  - 7.3.1 Setup . . . 35
  - 7.3.2 Results . . . 36
  - 7.3.3 Discussion of results . . . 37
- 7.4 Timing . . . 37

8 Conclusion **39**

9 Recommendations **41**
9.1 Laser Rangefinder 41
9.2 Motion Detection 41
9.3 QR Code Detection 42
9.4 Other Proposed Ideas 42

Appendices **43**

A Important Variables in Submodule Code **45**
A.1 Adjustable Variables in laserclass.py 45
A.2 Relevant Classes, Constants and Variables in MotionAnalysis.py 47

B Bill of Materials **49**

Bibliography **51**

Chapter 1

Introduction

1.1 The Zebro Project

Zebro is a six-legged robot developed by the Zebro Team at TU Delft. The robust design of Zebro makes it possible to conquer rough terrains. Zebro comes in various sizes, like the matchbox-sized Nano Zebro and the shoe box-sized Deci Zebro. Zebro is supposed to be employed as a swarm robot. The first real 'Zebro swarm' will consist out of 100 Deci Zebro's. Currently, Zebro is only able to walk. For their bachelor thesis, three groups of six people have worked on different modules that can be attached to Zebro to increase functionality. These are: The Autonomous Charging module, the Localization & Charging module, and the Environment Observation module. This thesis is part of the Environment Observation Module

1.2 The Environment Observation Module

Zebro needs to be able to walk autonomously, and therefore has to keep itself safe. Possible environments Zebro has to navigate through range from a table with obstacles, the university's campus, to the moon. The 'Environment Observation Module' is designed so that Zebro can keep itself safe using different kinds of sensors. The module is completely self-contained, and collects and processes its own sensor data. This data can be acquired by Zebro's top-level controller via the Zebro bus, which will make the final decision on what to do with the data.

The Environment Observation module was designed by three groups of two students. An overview of the division of functions is shown in Figure 1.1. This thesis is about the 'Camera Submodule', indicated by the green frame. The camera is used for visual information, the laser is used in a laser rangefinder system, and all processing is done on a Raspberry Pi, separate from the rest of the module. Its main functions is obstacle detection, either static or moving. The Camera Submodule will deliver all its information to the main processor of the Environment Observation module, which in turn can deliver it to the Zebro Bus. Information about the other parts of the module can be found in [1] [2], which also include the design of a custom PCB and an enclosure for the module.

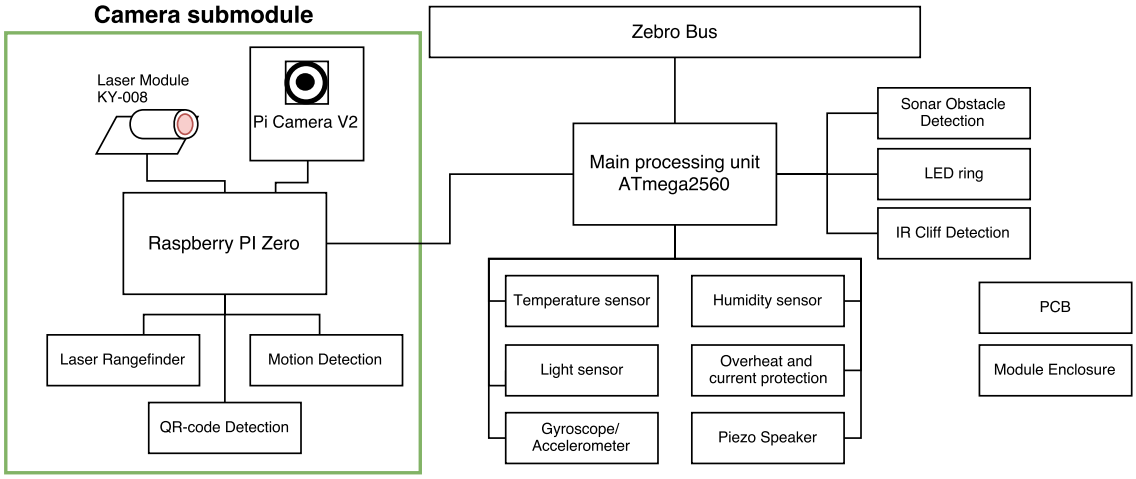


Figure 1.1: *Overview of the complete Environment Observation module. This thesis is about the 'Camera Submodule'.*

1.3 State of the Art

Obstacle detection and avoidance is a widely covered subject in robotics, as it is required to ensure the safety of autonomous moving robots (AMRs). There exists a large amount of obstacle avoiding methods. These can be vision based and non-vision based.

Vision-based methods include stereo vision and monovision. Stereo vision can give a detailed 3D-map of the environment, but suffers very high computational costs [3] [4]. Obstacle detection using monovision can be done by optical flow or color segmentation. Optical flow comes in two flavours: sparse or dense. Sparse optical flow computes motion on specific feature points, whereas dense optical flow uses the entire frame. Sparse optical flow is used by [5], [6] and [7]. In [5] the expected optical flow when driving through a (virtual) narrow tunnel is calculated, and compared with the observed optical flow. Deviations mean that the 'tunnel' is blocked by an obstacle. In [6], a homography-based motion detection is used and in [7], the Time To Contact (TTC) is used to determine the distance to stationary objects. Dense optical flow is used in [8], in which the Focus of Expansion and the TTC is used to determine a balance strategy for obstacle avoidance.

When using color segmentation based methods, known characteristics (color, texture) are used to distinguish obstacles from the background [9] [10]. A good application for this is for example in a greenhouse [11], where green plants, fruits, and the white greenhouse roof can be easily characterized. The process of making this distinction however is complicated, even more when a robot is deployed in unknown areas. Especially distinguishing an object when it is the same color as the background is a challenge when it comes to monovision.

Vision based systems have common drawbacks, namely computational cost and that they need good illumination. Therefore, in AMR's often other types of sensors are deployed next to, or instead of a vision based system. There exist various types of sensors that can be used, ranging from expensive (e.g. LIDAR) to cheap (e.g. ultrasonic). Different situations in which obstacle detection is deployed, call for a different kind of approach. For example, self-driving cars use different (and more) sensors than a Roomba.

Hanumante et al propose a low cost obstacle avoidance robot using IR sensors [12]. This can detect obstacles right in front of the robot, but does not work well outside and on dark-colored obstacles.
A type of sensor that does not suffer from the constraints of light- and vision based types of sensors, are ultrasonic sensors. In [13] ultrasonic sensors are used in a UAV (Unmanned Aerial Vehicle) so it can be used in an environment with gas or smoke. The system had problems with detection soft surfaces as foam

and people wearing clothes, but was able to detect water and glass surfaces, which the (simultaneously tested) IR sensors couldn't. Because the detections of the ultrasonic sensors were not reliable enough, they argue that they should always be used in combination with other sensors.

Fu et al. [14] demonstrate a triangulation laser scanner as a small, simple, low-power and low-cost method of extracting information from the surroundings of a robot. Distances up to 600 mm were measured with high accuracy, with the limiting factor being the environmental light.

1.4 Thesis Structure

This document serves two main purposes: it describes our methods, thoughts and experiments during the project, but is also serves as a reference for the designed camera submodule when it will be used by others. The chapter layout is as follows:

- Chapter 2 lists the requirements of the module, and the camera submodule. This includes the initial requirements as set by the Zebro team, as well as additional requirements for our submodule.
- In Chapter 3, general design considerations are discussed such as the methods used and the hardware employed. Moreover, the use of a camera is justified.
- In Chapter 4, the design of the laser rangefinder including physical parameter design and software, is described.
- In Chapter 5, the design and software of the motion detection system is explained.
- Chapter 6 discusses the entire module integration and how the camera submodule fits in the Environment Observation module.
- Chapter 7 deals with the tests, results and discussion of the submodule.
- In Chapter 8, a summary will be given on the achieved results.
- Finally, additional recommendations and ideas will be given in Chapter 9.

Chapter 2

Programme of Requirements

2.1 Environment Observation Module Functional Requirements

From the Zebro Team, the Environment Observation Module as a whole had the following functional requirements (the ones relevant for this thesis are in bold):

[1.1] Detect obstacles

[1.2] Detect cliffs

[1.3] Discern between scalable and dangerous obstacles and cliffs

[1.4] Determine the system's temperature

[1.5] Determine the system's voltage, currents and power flow

[1.6] Implement an Autonomous Module Damage Prevention System (AMDPS)

2.2 Thesis Requirements

For the thesis, the following requirements are expected:

[2.1] Develop a test bench to run and record different parameters of Environment Observation performance

[2.2] Develop and test at least three types of sensor systems

[2.3] Develop the necessary software, hardware, control and their architectures for requirement [1.3]

[2.4] Develop the necessary software, hardware, control and their architectures for requirement [1.6]

2.3 System Requirements

The Environment Observation Module should also meet the following criteria:

[3.1] It should be self-contained

[3.2] It should have very well-defined interfaces with the outside world, that is, the Physical interface, the power interface and the control & status interface.

[3.3] The module must be modular and easily replaceable by another robot.

[3.4] The system must be built cost-effective, as 100 Zebro's are planned to be built.

[3.5] The system must be able to survive in an unknown environment, including rough terrain and public areas with people.

2.4 Camera Submodule Functional Requirements

Requirements [1.1] and [1.2] are carried out by [2] and requirements [1.4], [1.5] and [1.6] are covered by [1]. This thesis will address [1.1] and [1.3].

In order to make requirement [1.3] more concrete, we define *dangerous* as being a moving object, and *scalable* as being an obstacle that can be overcome by Zebro. Furthermore, three additional requirements were drawn up that are specific to the submodule described in this thesis:

[4.1] Detect and read QR-codes.

[4.2] The system must communicate with the rest of the module, which in turn will communicate with the Zebro Bus.

[4.3] The system must work on 5V.

Chapter 3

General design

3.1 Motion Detection

The choice of using a camera came about by viewing Zebro as an animal rather than a robot. Even though the module is officially called the "Environment Observation Module", we are effectively implementing the senses for this animal. When considering the five human senses, vision makes the most sense to implement in robotics. One problem that arises however, is the fact that complete obstacle recognition would not only be extremely difficult but also very computationally expensive, leading to expensive hardware, leading to an unfeasible module.

The way that follows nature most closely is a stereo-vision solution. The DelFly Project [15] is an example of a project that makes use of two cameras and computes a disparity map in order to see depth. A few weeks have been spent trying to use this platform for the Environment Observation Module, because it was believed to be a very cheap option. However, when it turned out that this was not the case (around 80 euros), together with the fact that no documentation was available, this plan was aborted. Other stereo cameras on the market are even more expensive than the DelFly.

After this, using two eyes was no longer considered an option. However, this is not the only way to view depth using vision. Some animals, for example birds or locusts [16], make use of motion parallax by moving their heads in order to see depth. This lead to the idea of using optical flow in order to estimate distances to objects. This has been implemented using the idea of Wilfried Enkelmann [5] and making use of the OpenCV library. Enkelmann implemented an algorithm that was to be used for cars to detect obstacles on the road.

This algorithm makes a virtual tunnel though which the car can drive and calculates the expected flow vectors based on that tunnel. This is then compared to the real flow vectors and if the real vectors flow faster than the ones in the tunnel, this is due to an obstacle blocking the road. Two necessary assumptions are based on the car motion: the velocity is known and the camera only translates in one direction (no rotations). For Zebro, neither of these assumptions hold. Furthermore, calculating optical flow can be computationally expensive (in OpenCV, in which a limited number of features are tracked, took about 0.5 seconds per frame).

Luckily, another solution is available. The Pi Camera has on-board hardware that is capable of outputting motion data directly, practically without delay (without processing this can reach 20 frames per second). This data can be used by the Zebro to detect motion in its surroundings, for example moving people or cars. This is important for Zebro in order to function properly in society. For example, Zebro can notify that it is aware of the presence of humans by making sound or it can lie down when something dangerous is approaching. The design for motion detection is explained in Section 5.

3.2 Obstacle Detection

A method for obstacle detection is also integrated in another part of the Environment Observation Module in the form of an ultrasound sonar system. This gives a good indication of the surroundings of the Zebro,

but not all obstacles can be correctly detected due to reflection and absorption of the ultrasound waves. Another sensor type should be deployed as an addition to the the ultrasound sonar system to make obstacle detection more reliable.

It was therefore decided that the Camera Submodule should have an obstacle detection method based on sensors, that provides the distance to an obstacle. Although movement can be used to become aware of the presence of possible dangers, it is not really functional for absolute distance measurements. Purely visual methods were therefore not considered a reliable option. Finally, a laser rangefinder system based on triangulation was chosen as a cheap, reliable option. This was chosen for the following reasons:

- The laser rangefinder will work alongside the sonar system. The laser rangefinder can therefore be used to give extra critical information for the most important direction: forward. According to literature [14] [17] [18], a laser rangefinder is considered to be very accurate, especially in close range, so it is very suitable for this task.
- The laser rangefinder works on materials that the sonar system does not well on. This includes wave absorbing materials and objects at an angle that reflect the wave the wrong direction. Furthermore it is less prone to false detections (for the sonar system this can be caused by floor reflections for example).
- A laser makes it visible to humans what the robot is looking at, which is good from a human interaction point of view.

3.3 Processing and Camera Hardware

Image processing is not done on the ATmega2560 that is used by the rest of the submodule, as it is not suitable for this task due to insufficient storage and processing power. Moreover, because image processing is very different from the functions that the main processor of the module has, it was also deemed logical that the Camera Submodule should work 'separate' from the rest for the sake of modularity. The Raspberry Pi is suitable for image processing, and it is very user friendly. The latter is important for making it easy to make adjustments to the vision system with a specific task in mind, and also handy for when the Zebro team decides on enhancing the vision system. The type of Raspberry Pi (RPi) used is the Raspberry Pi Zero W. The Pi Zero is the cheapest and smallest available but therefore has also limited processing capabilities. The 'W' variant of the Pi Zero provides wireless LAN which was useful during the design process.

The camera used is the Raspberry Pi Camera V2 [19], with an 8-megapixel camera and HD filming possibilities. The Pi Camera can be connected easily to any Raspberry Pi with a ribbon connector, and contains a lot of build-in features. The Pi Camera is a popular product, and therefore a lot of documentation can be found online making it a relatively easy camera to work with.

3.4 Detection of QR Codes

As a demonstration of the possibilities that having a camera brings, a simple QR Code detector and decoder was implemented. When a QR Code is detected, it outputs a string with the translated data.
As this functionality is out of the project's scope, it was implemented using the ready-to-use 'ZBar' package. ZBar is an open source bar code reader, that can also be used for detecting and translating QR Codes. Some possible usages of this functionality include:

1. Recognize and classify other Zebro-related objects like a charger, other Zebros or other objects it could use.
2. Give commands to Zebro, e.g. when a QR code with the command 'stop' is shown, Zebro stops moving.

Chapter 4

Design of the Laser Rangefinder

To measure distances to obstacles that Zebro is facing, a laser rangefinder based on triangulation was designed. The basic principle of this laser rangefinder is: Zebro shines on an obstacle with a laser, the laser dot is detected by the camera, after which the distance can be calculated using known parameters.

4.1 Setup

The setup of the laser rangefinder is shown in Figure 4.1. The laser is placed at a horizontal distance of 10 cm from the camera's center, at an angle of 80°. The laser is placed next to the camera, instead of above or below, because the design of the complete module enclosure made it impossible to place the laser at a sufficient vertical distance from the camera. The distance of 10 cm is also close to the maximal attainable width on the enclosure.

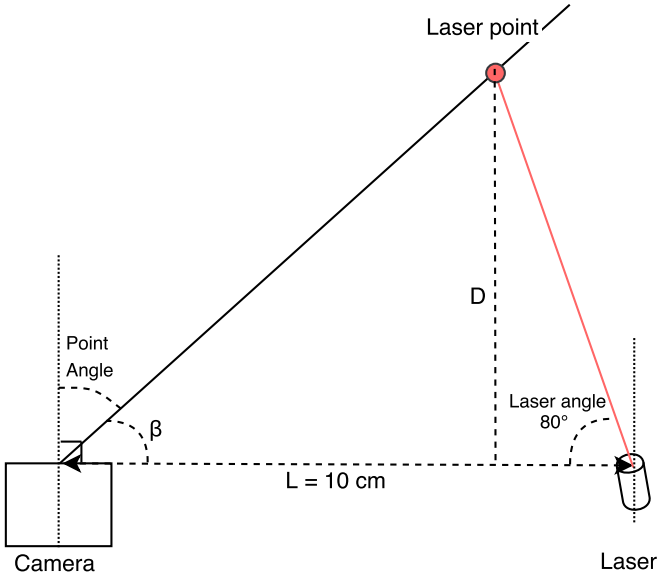


Figure 4.1: *Setup of the laser rangefinder.*

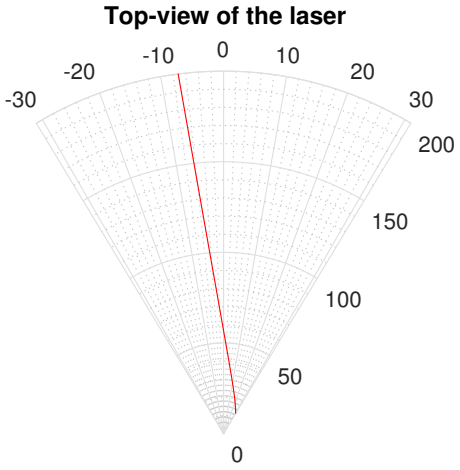
The Pi Camera's field of view is 62.2° [19]. Knowing the width of one camera frame, the angle per pixel can be calculated. With the angle per pixel and the laser point's x-coordinate in the frame, the angle at which the laser point is detected (the *Point angle*), can easily be calculated (Equation 4.1). *Point Angle* can be positive or negative, depending on whether the laser point is in the right or in the left half of the camera's field of view.

$$\text{Point angle} = \frac{\text{Field of view}(^\circ)}{\text{Frame width (pixels)}} \cdot \text{Dist. to laser point from field of view center (pixels)} \tag{4.1}$$

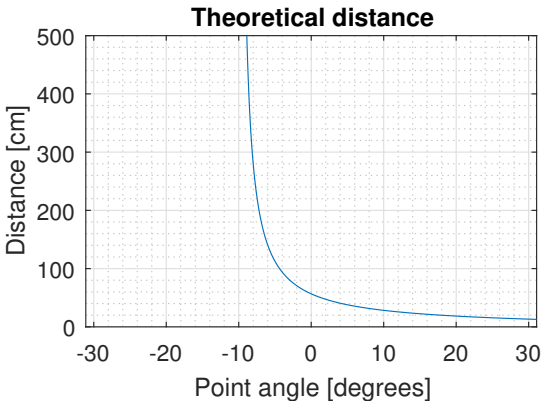
After determining the Point Angle, the distance D to the laser point can be calculated using Equation 4.2. This is the distance Zebro can go forward until the obstacle the laser points to is reached.

$$\mathrm{D} = \mathrm{L} \cdot \frac{\tan(\beta) \cdot \tan(\textit{laser angle})}{\tan(\beta) + \tan(\textit{laser angle})} \tag{4.2}$$

The plot of this equation is shown in Figure 4.2, as a function of the Point Angle instead of β.



(a) *Top-view of the laser, as seen from the camera's Field of View. The laser is located on the right side of the camera.*



(b) *A linear plot clearly shows that the distance has an asymptote on -10°, which corresponds to the Laser Angle.*

Figure 4.2: *Angle versus distance plots, with Laser Angle = 80° and separation L = 10 cm.*

4.2 Physical Parameter Design

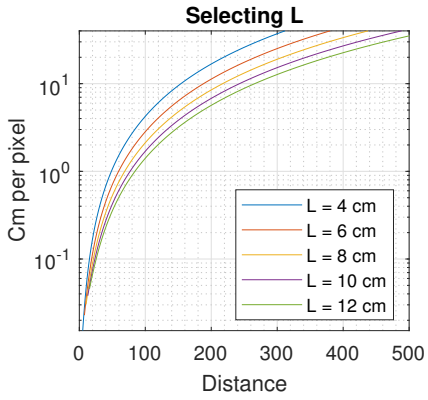
There are two parameters in Figure 4.1 that had to be chosen: The *Laser Angle* and the space between laser and camera, *L*. For choosing these values, four things were considered:

- The Laser Angle cannot be less than 58.9°, as this would mean the laser would go out of the Field of View of the camera at larger distances.
- The Laser Angle should be somewhat close to 90° , as it is important to see what is directly in front of Zebro.
- Increasing L and the Laser Angle increases the resolution at larger distances. See Figure 4.3.
- Decreasing L and the Laser Angle decreases the minimum distance at which something is detected. See Figure 4.4.

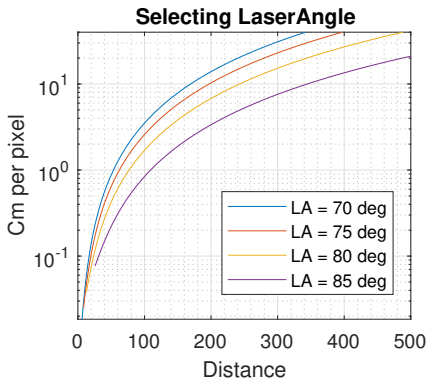
The last two considerations lead to a trade-off between resolution and minimum detectable distance. With all these considerations in mind, the Laser Angle has been set on 80°, and L has been set to 10 cm. This results in a minimum distance that can be detected at 12.8 cm, a resolution of one cm per pixel at 77 cm and a resolution of 10 cm per pixel at 246 cm.

4.3 Laser Type

For the choice of laser the following characteristics were considered: color, type (point or line) and price. For the color of the laser optimally a green laser is used as the camera is more sensitive for green light than for red [20]. However, green lasers are around four times as expensive than red lasers, and therefore a red laser is used (which still gives acceptable results). An IR laser was considered but deemed inappropriate

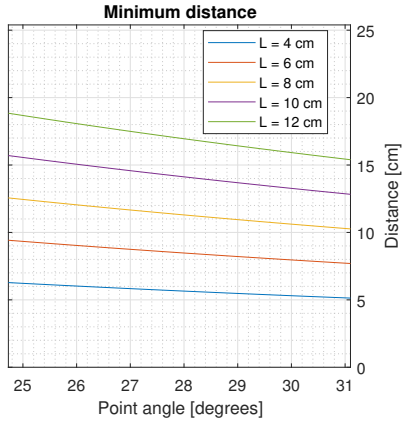


(a) A larger L results in less cm per pixel, so a better resolution.

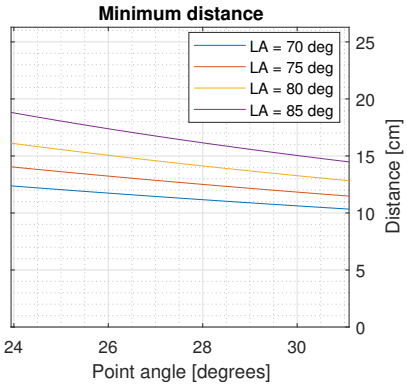


(b) A larger Laser Angle results in less cm per pixel, so a better resolution.

Figure 4.3: *For better resolution, L and the Laser angle should be as large as possible.*



(a) A smaller L results in a smaller minimum distance



(b) A smaller Laser Angle results in a smaller minimum distance.

Figure 4.4: *The minimum distance at which the laser can be detected is when the laser is at the right side of the FoV, which is at 31.1°.*

for the Zebro. For applications in which Zebro walks around public places, the risk of someone looking into the laser unknowingly and risking eye damage is too high.

Second, a choice was made between a line laser and a point laser. A laser line provides information about the whole horizontal or vertical field of view at a certain point, a laser point only provides information about the point it shines at. Ultimately a point laser was chosen. The reason for this was that one of the perceived uses of Zebro is that it can walk over campus. In that case, by humans, it can be perceived as 'annoying' when there are laser lines everywhere. A point laser is less obtrusive, and therefore more desirable from a human interaction point of view. Furthermore, the laser works as an enhancement for the sonar system, which will already provide initial information about the whole field of view. The laser will only provide extra information about the most critical direction for the robot: what it is moving towards.

The laser used is the KY-008 Laser module, which transmits 650 nm (red) light. This laser module is cheap (<€2), and has a fixed resistor on it. It is controlled by one of the RPi's GPIO pins, which outputs 3.3V. The laser requires 5V, therefore the laser is connected using a transistor as switch as shown in the schematic of Figure 4.5.

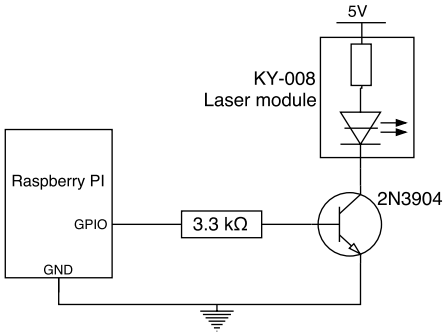


Figure 4.5: *Schematic of the laser module connected to the Raspberry Pi GPIO pin.*

4.4 Software

The software for the laser rangefinder needs:

1. To detect a laser spot
2. Calculate the distance using the detected spot
3. Distinguish between its own laser spot and possible others.

Two methods were tested for detecting the laser in a frame. For one method, the 'difference method' the laser was turned on and off with each frame. A frame with laser was then compared to the previous frame without laser. The area that differed could then be determined as the laser spot. However, it turned out that the frame rate was not high enough and therefore the difference between two subsequent frames was more than just the laser. This method was then deemed unusable for this application.

The method that was implemented uses the visual characteristics of the laser to detect it in a frame. Every time the camera captures a frame, Zebro will look for its own laser in the frame by searching for red, bright areas. The general process for this is illustrated in Figure 4.6. For the laser detection the OpenCV library was used.

The code for the laser rangefinder contains a lot of variables that can be used to optimize the code for different situations and Zebro's behaviour (e.g. speed and average amount of Zebros in a given area). For future reference, these variables can be found together with an explanation in Appendix A.1.

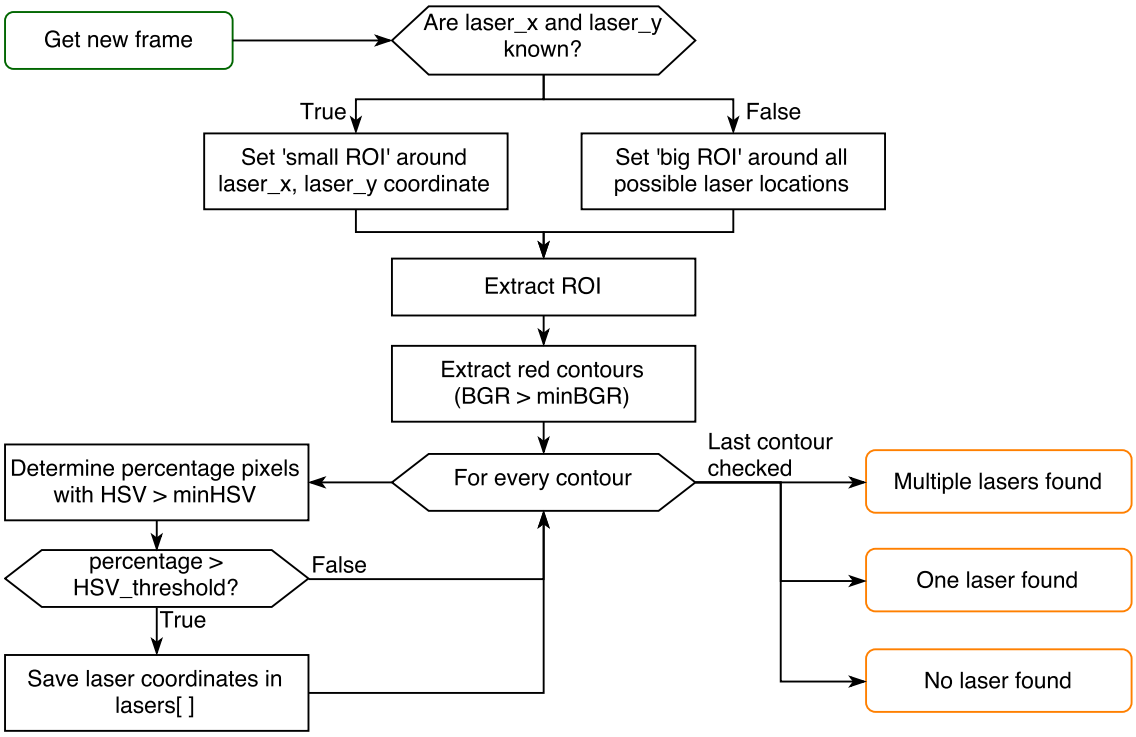
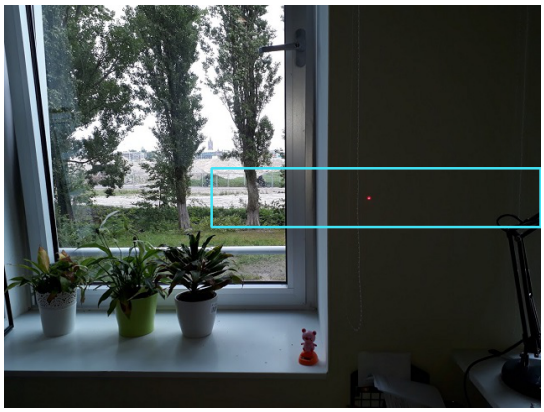


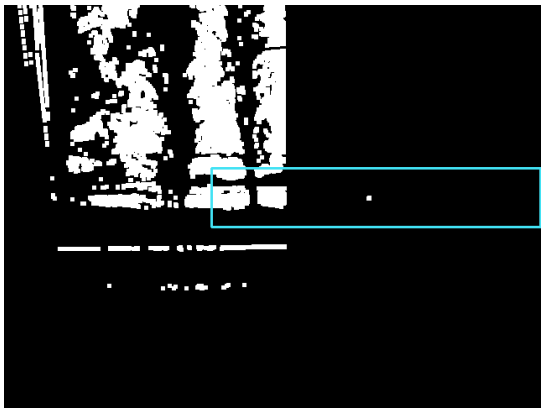
Figure 4.6: *Algorithm for the detection of laser(s) in a frame.*

After loading a frame, Zebro does not check the whole frame for a laser but only a Region Of Interest (ROI). When a laser was detected in a previous frame, the Zebro only checks a small area, *smallROI*, around that point. When Zebro doesn't know the laser's previous location, it checks the *bigROI*, that entails all possible locations of the laser. Because of the laser rangefinder's setup, this is only a small part of the whole frame. The laser point will always be around a fixed height, and due to the angle of the laser, cannot be in the far left of the frame (see Fig 4.2a).

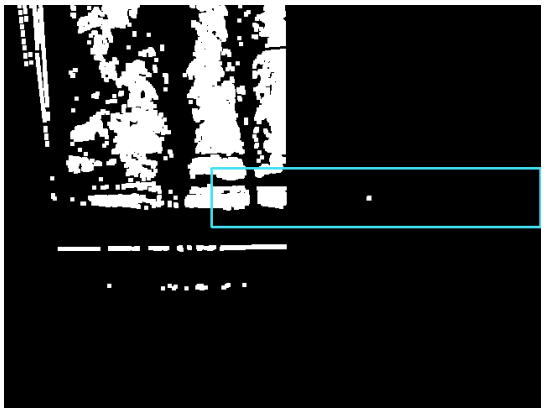
The code starts using *bigROI*, visible as a light-blue frame in Figure 4.7a. Next, it checks for lasers in the selected ROI. As a first step, the code searches for contours with a high red value. This is visualised in Figure 4.7. In (a), the retrieved frame is shown, with the laser point visible in the middle of *bigROI*. Using OpenCV's *inRange()* function, pixels with a BGR (Blue, Green, Red) value of which Red is higher than the set threshold, are set white (b). Then in (c), OpenCV's *dilate()* function is used to enlarge small contours (which the laser often is), and erase some of the noise in larger white areas. This decreases the number of contours found, which increases the speed of the code. Using OpenCV's *findContours()* function, contours of white areas are found. In (d) the contours found after this process are lined purple.



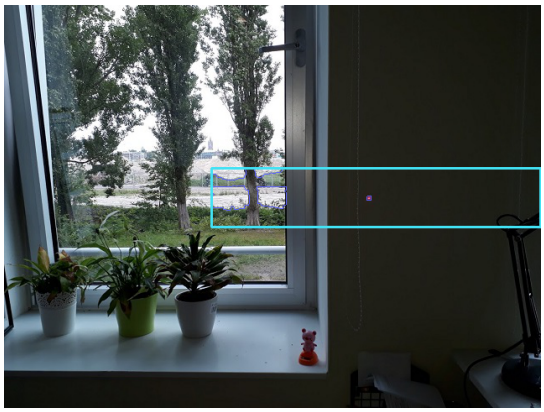
(a) Retreived frame with the laser visible in the middle of bigROI



(b) A BGR threshold (minBGR) is applied using inRange(), areas with a high R value remain



(c) Areas are enlarged using dilate()

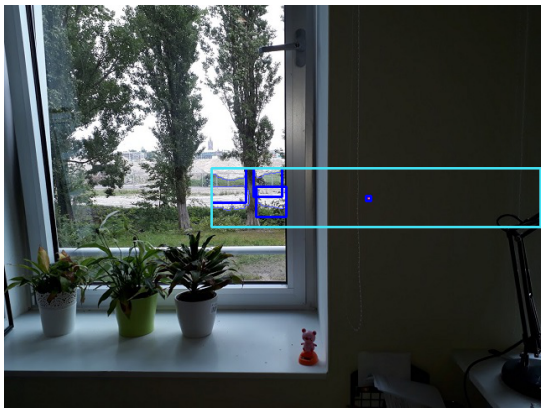


(d) Contours found with findContours() are lined purple

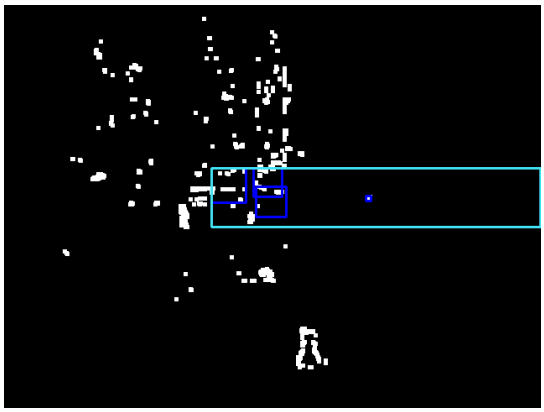
Figure 4.7: *The process of finding red contours in a frame.*

Following the extraction of red contours, every contour is checked whether it is a laser or not. The characteristic used to do this is the laser's brightness. Each contour is converted to the HSV space. HSV (Hue, Saturation, Value/Brightness) is a cylindrical-coordinate representation of points of the RGB color model. By trial the minimal values of the HSV range (minHSV) the laser often falls into were determined.

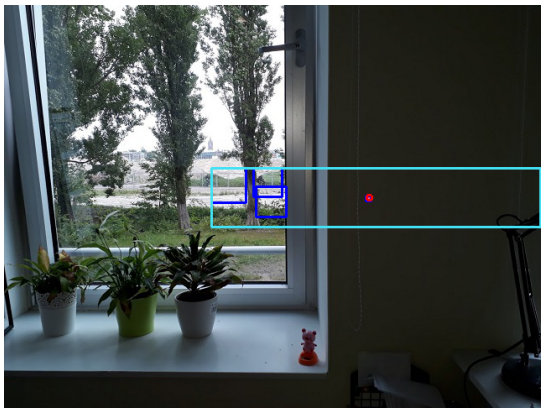
The process starts by drawing a minimal enclosing rectangle around each previously detected contour that is small enough to be a laser (Figure 4.8a). These contours are then converted to the HSV space. By using the *inRange()* function for every contour the percentage of pixels with a HSV value larger than *minHSV* is determined (Figure 4.8b). When the percentage in the minimal enclosing rectangle is at least 85%, the contour is determined to be a laser spot. In Figure 4.8c the contour determined to be a laser spot is indicated with a red circle, and is indeed the laser. In this case, in the next frame the code will only search around *smallROI* shown in Figure 4.8d.



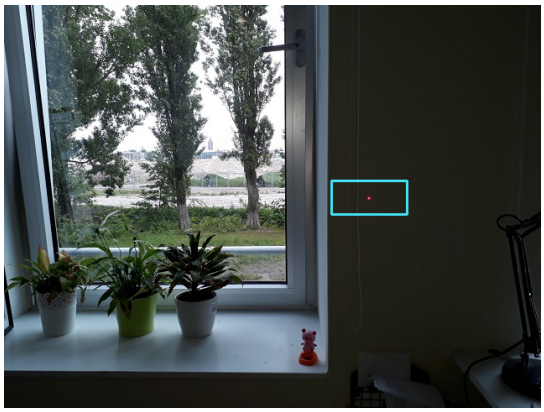
(a) *The contours that will be checked on whether it is a laser or not are marked by dark blue squares.*



(b) *The contours are converted to the HSV space and the minHSV threshold is set using inRange() (for illustration purposes the whole frame is converted). The contour of the laser is fully white.*



(c) *The contour with a high enough percentage bright pixels is marked as laser. A minimal enclosing circle (red) is drawn around the contour to determine the laser's x- and y-coordinates.*



(d) *smallROI is drawn around the found laser point.*

Figure 4.8: *The process of determining which red contour(s) is/are a laser spot.*

After the process of detecting laser spots in the frame, there are three situations possible: There is no laser found, there is only one laser found, or there are multiple lasers found.

The case in which one laser is found is the most straightforward, as shown in Figure 4.9. The coordinates of the laser's center are determined by drawing a 'smallest enclosing circle' around the laser contour. The laser's coordinates are equal to the center of that circle. The value for *minHSV* is decreased a bit so the laser can be detected easier in the next frame, as the probability of a false detection is lower for the *smallROI*.

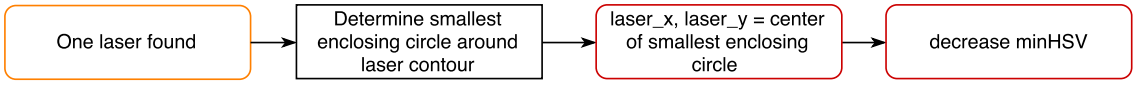


Figure 4.9: *Algorithm for when only one laser has been found.*

When no laser is found, the code continues as shown in Figure 4.10. Every time no laser is found, the *notfound* variable is increased by 1. When the laser is not found too many times in a row, the laser spot is deemed unknown and the Zebro will start searching in the *bigROI* again with increased *minHSV* (to avoid

false detections). Every time a laser is found, *notfound* is reset to 0.

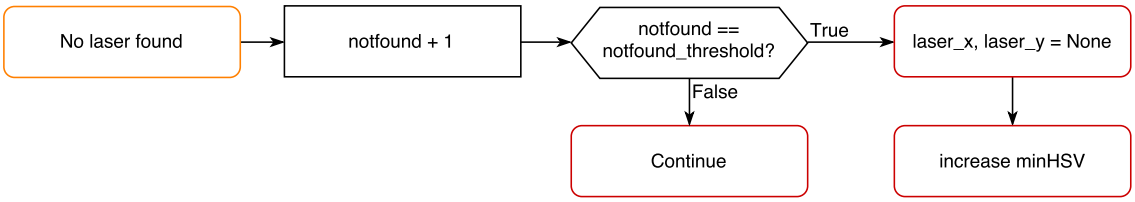


Figure 4.10: *Algorithm for when no laser is found.*

The case in which Zebro detects multiple lasers in one frame, is somewhat more complicated. A flowchart of the algorithm is shown in Figure 4.11. When multiple lasers are found, *multilaser_max* is set, which is the threshold of how much frames in a row multiple lasers have to be found before taking action. This is set randomly every time, so it will be different for different Zebros. *Multilaser_max* is reset when either no or 1 laser is found, or when its value has been reached and action is taken.

When *multilaser_max* is reached, Zebro turns its own laser off. Every area (small ROI) around detected laser locations, stored in *lasers[]*, is then checked for the presence of a laser using the same method as for the initial laser detection described previously. The area in which no laser can be found anymore, is set to be Zebro's own laser location. Because of the possibility of something going wrong in this stage, the Zebro's laser coordinates are only changed when there is really only one area in which no laser can be found anymore.

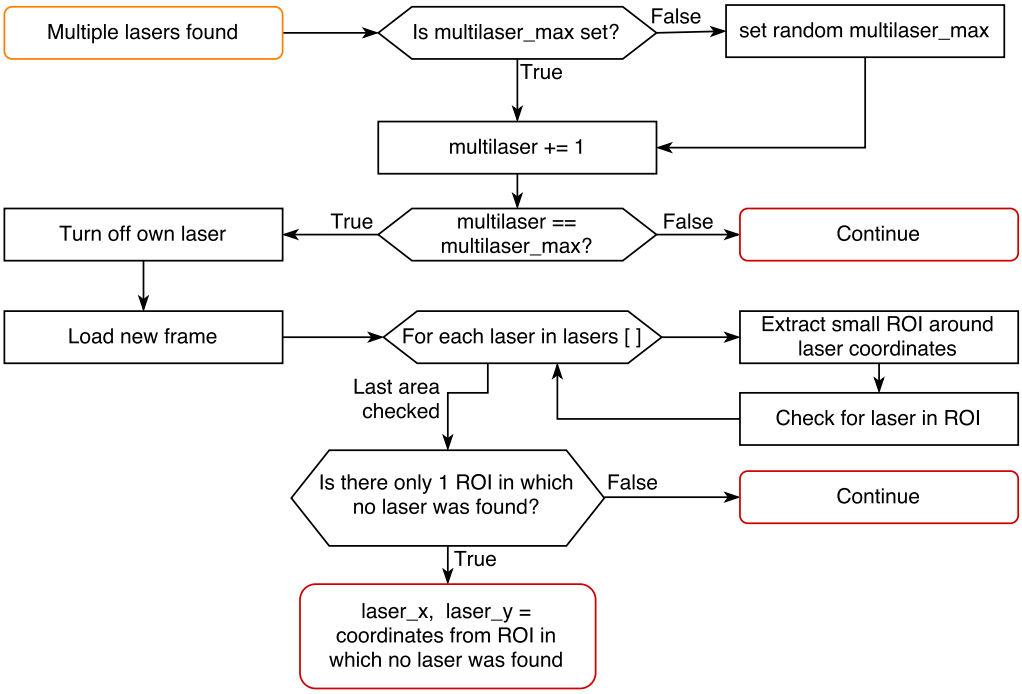


Figure 4.11: *Algorithm for when multiple lasers are found in one frame.*

When the location of Zebro's own laser is known after finishing one of the three processes (one laser, no laser or multiple lasers), the distance to the obstacle the laser shines on is calculated using Equation 4.2. This formula is implemented in the function *calcDistance()*. After that, the next frame is grabbed and the whole process will start over from the beginning again.

Chapter 5

Design of Motion Detection

5.1 Hardware and Libraries

The motion estimation is done by the h264-encoder on the Pi Camera [21]. This is done using a block-matching algorithm. In short, this divides the screen in blocks of 16x16 pixels and tries to find out where this block went in the next frame, using a search window. This process has been encapsulated by the Pi Camera API [22]. After that, it is ready for analysis. The result is shown in Figure 5.1.

5.2 Software

For a frame that is 640x480 (experiments showed that this was the best trade-off between resolution and processing cost), the resulting motion array provided by the encoder is 41x30. The first step is that all the lengths of the motion vectors are computed, and the lower section of the screen is discarded. This is because very often, the motion data about the surface Zebro is walking on is very chaotic, so that this information has to be discarded. See Figure 5.1. The threshold for ignoring data is defined by a constant called *ROAD_THRESH* and can be found in Camera_Module/Autoboot/MotionAnalysis.py. Another way to ignore the chaos is based on the *'SAD'* (Sum of Absolute Difference)-value that is also provided by the h264-encoder. Sadly, finding a robust threshold value can be difficult, because if the limit is too low it might also ignore actual moving objects. This would defeat the purpose of doing motion detection.

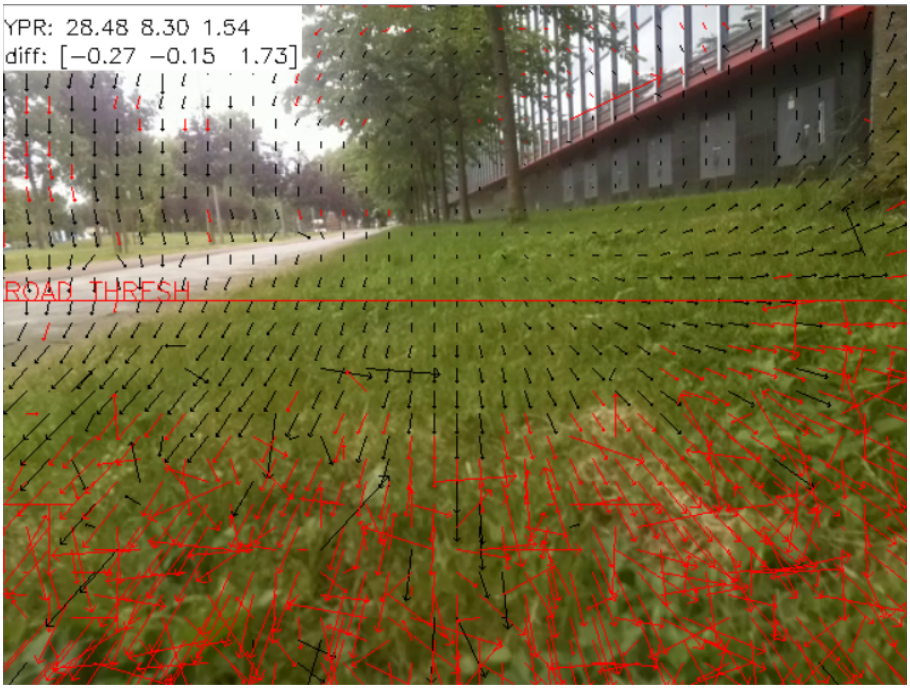


Figure 5.1: *Motion Data when the Zebro is moving forward, without moving obstacles. At the bottom of the figure, the data is very unreliable. Motion vectors with a 'SAD' value higher than 400 are colored red. The data in the top-left of the figure is the data of the gyroscope (Yaw, Pitch, Roll) in degrees. "Diff" is the difference in degrees with respect to the previous frame.*

In the second step, the Motion Matrix is formed. A Motion class instance represents one motion vector and is created for vectors that are sufficiently large to be considered relevant. "Sufficiently large" is defined by the parameter *MOTION_THRESH* and can be found in the same file. The Motion Matrix is a 2D-list of Motion instances. In the third step, the motions are grouped together to form a Danger instance. A Danger can be a person, a car or any other moving object. The Danger class includes a *get_severity()* method that determines the total motion in a Danger instance. Figure 5.2 visualizes the Motion and Danger classes.

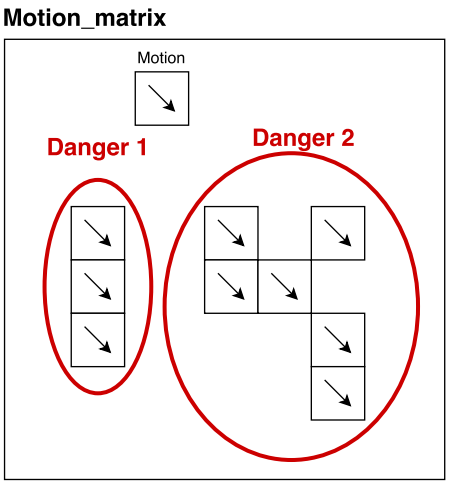


Figure 5.2: *The Motion_matrix, Motion and Danger visualized.*

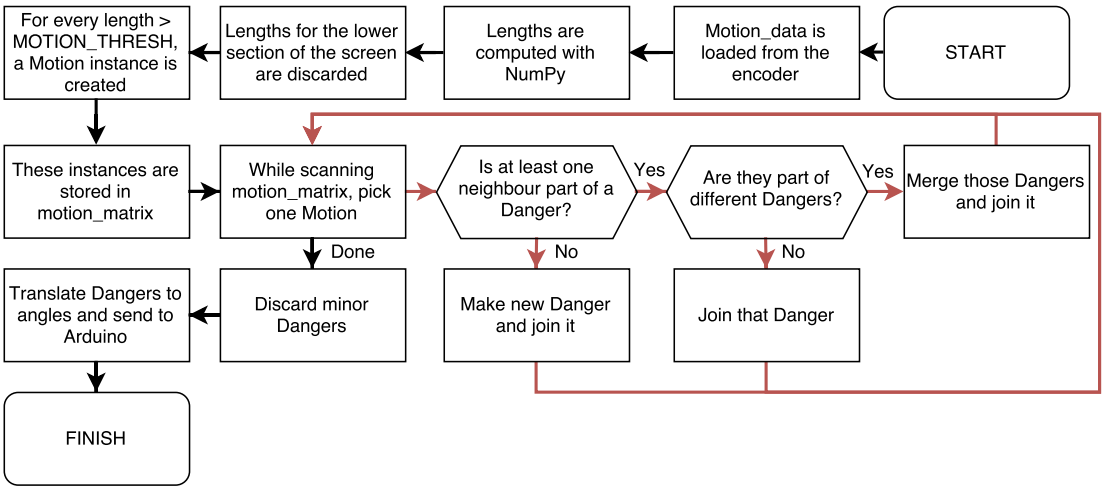


Figure 5.3: *The entire process of motion detection. The arrows in red indicate a loop.*

The Motion instances are grouped together in the following way: the Motion Matrix is scanned from left to right, top to bottom. When it encounters a Motion instance, this is linked to a Danger instance. First, it checks the neighbouring locations in the Motion Matrix that it has already processed (that is, one position to the left, top-left, top and top-right). If any of those are Motions (and thus already linked to a Danger), join that Danger. If two neighbours are part of different Dangers, merge the Dangers to one as they are apparently connected. If there are no neighbouring Motions, a new Danger is instantiated. The result is if two people are moving within a frame, both can be treated as separate entities.

Then, "minor" Dangers are discarded . These are dangers that may be due to moving objects in the far distance, or due to motion noise. A Danger is considered "Dangerous" if its *severity()* (simply the *length* of all the Motions summed) is larger than *MIN_SEVERITY* and the number of different Motions is larger than *MIN_MOTIONS*.

Finally, the angles that correspond to the location of the moving object are calculated based on the leftmost and rightmost motion vector of each Danger. This, together with the severity of each Danger, is sent to the PCB. The entire process is shown in Figure 5.3, and all the relevant classes, constants and variables are summarized in Table A.2. The parameters mentioned in this table can be adjusted in order to make Zebro more curious, or more shy.

Figure 5.4: *Final result of the motion detection. A person is walking and his motion is detected. At the moment of the capture, only his right leg is moving. The red arrows show small movement due to the camera motion. The black arrows represent motion larger than MOTION_THRESH.*

Chapter 6

Module Integration

6.1 Software Integration

In Chapters 3, 4 and 5, three different functionalities have been described:

- The detection of QR-codes
- Distance measurements using the laser rangefinder
- Motion detection

These functions are carried out by classes called *AnalyzeQR*, *AnalyzeLaser* and *AnalyzeMotion* respectively, and are instantiated in *main.py*. This main python script starts automatically when the Raspberry Pi is booted. The three classes all have an *analyze* method that is called by the *picamera* package [22] whenever a new frame is available. The Raspberry Pi Camera has four possible video streams, two of which are used for distance measurement and QR-code detection, and one is used for streaming motion data. The fourth stream is reserved for future use.

6.2 Communication

For communication between the Raspberry Pi and the ATMega controller on the PCB, two options were considered: UART and I2C. When using I2C, the Raspberry Pi can only act as the master [23], but the ATMega in this case has to act as master to communicate with attached sensors. Therefore, UART was chosen.
The ATMega's pins work on 5V, whereas the RPi's pins work on 3.3V. To prevent damage to the RPi due to over-voltage, a voltage divider is needed between the ATMega's TX and the RPi's RX. Because 3.3V is still seen as a logic high by the ATMega, no level converter is needed for the other line. The connections for communication are schematically shown in Figure 6.1.

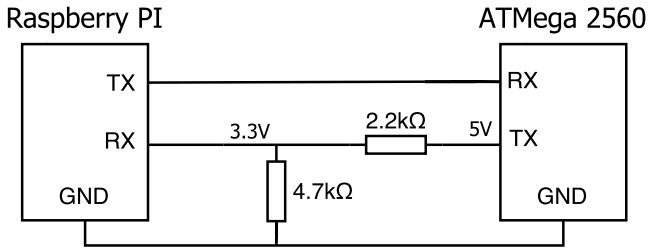


Figure 6.1: *Connection for serial communication between the RPi and the ATMega2560.*

In *main.py*, a separate thread is dedicated for communication. The ATMega can control what functions will run by sending commands, see Tables 6.1 and 6.2. If *get data* is called, the RPi checks if new data is available for every function. If so, it is loaded in DATA. If no new information is available since the last call, DATA is "Same". If no QR-codes or laser is found, they will return None.

Table 6.1: Communication protocol between the Raspberry Pi and the PCB

RPi Receives	RPi Sends	RPi action
“laser on”	“laser is on”	Turn laser on
“laser off”	“laser is off”	Turn laser off
“qr on”	“qr is on”	QR search function is called every frame
“qr off”	“qr is off”	QR search function is not called anymore
“get data”	“QR = DATA/None” “Laser = DATA/None” “Motion = DATA”	

Table 6.2: DATA format

Function	Format
QR	Decoded QR string
Laser	”Obstacle: [Angle Distance]” Angle in degrees (pos or neg), distance in cm, all numbers are integers.
Motion	“Danger 1: [startAngle endAngle severity]” “Danger 2: ...” Angles are in degrees, all numbers are integers.

6.3 Hardware Integration

The connections of the Camera Submodule with the rest of the Environment Observation Module are shown in Figures 6.2 and 6.3 with the relevant pin connection in Tables 6.3 and 6.4. The Bill of Materials for the complete module can be found in Appendix B.

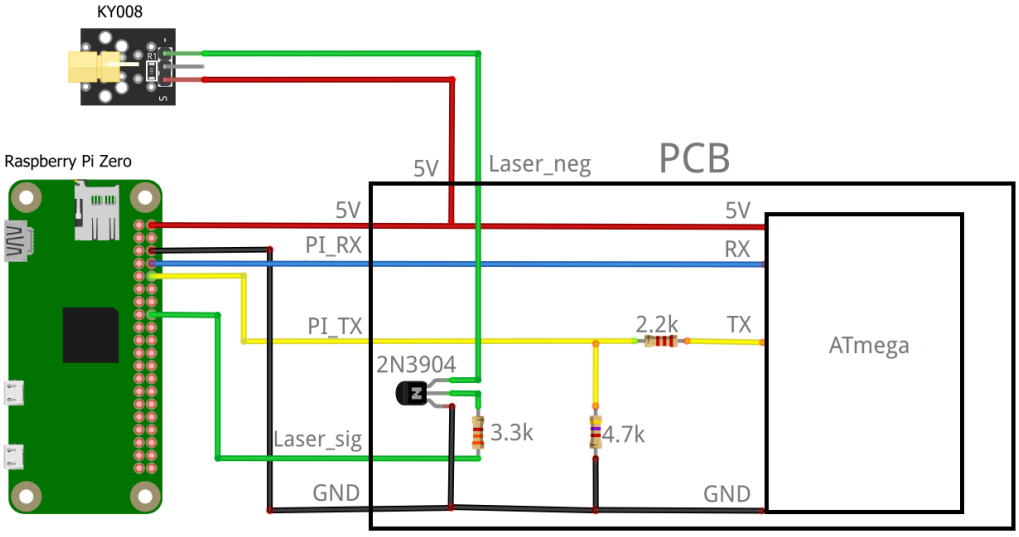


Figure 6.2: Overview of the connections of the Raspberry Pi to the PCB, and the relevant components on the PCB. The camera is directly connected to the Raspberry Pi with a flex cable.

Table 6.3: *Raspberry Pi physical connections*

Pi Pin	Pi Name	PCB Pin (J12)	PCB Name
02	5V	1	5V
06	Ground	2	GND
08	TXD0	4	PI_RX
10	RXD0	3	PI_TX
16	GPIO23	5	Laser_sig

Table 6.4: *Laser physcial connections*

Laser Pin	PCB Pin (J23)	PCB Name
Sig	1	5V
Gnd	2	Laser_neg

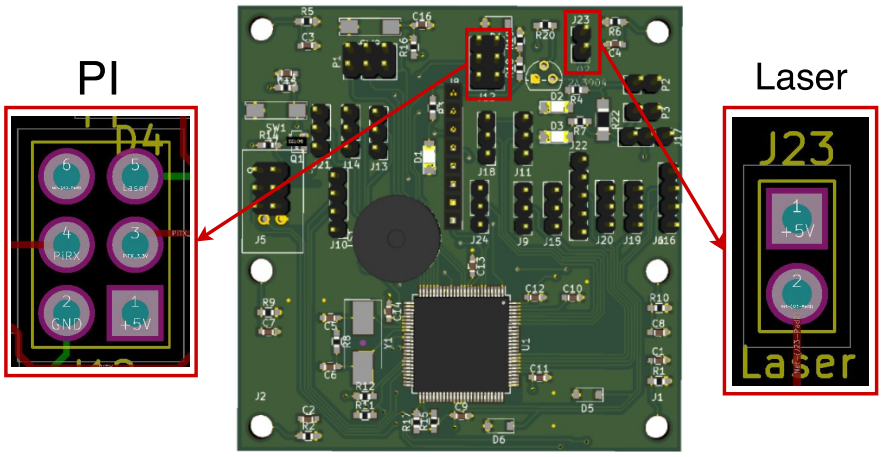


Figure 6.3: *The PCB that has been designed by [2]. The insets show the connections relevant for the Camera module.*

6.3.1 Power Consumption

The Raspberry Pi and the laser will be powered by the rest of the submodule, which also includes overheat and over-current protection [1]. Both the laser and the RPi will be connected to a 5V source. The current usage was calculated as in Table 6.5.

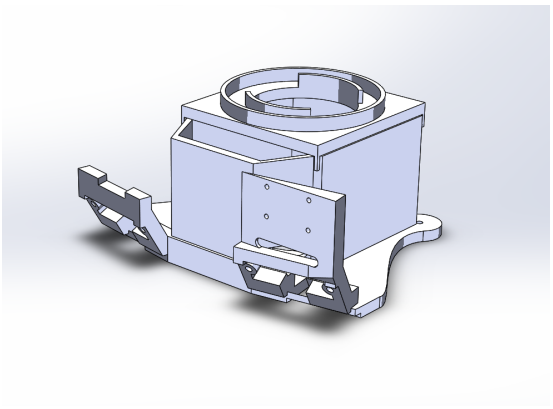
Table 6.5: *Calculated current usage by the Camera Submodule*

Part	Avg. current consumption (mA)
Raspberry Pi	150
Pi Camera	250
Laser circuit	30.55
Serial communication	0.77
Total	**431.32**

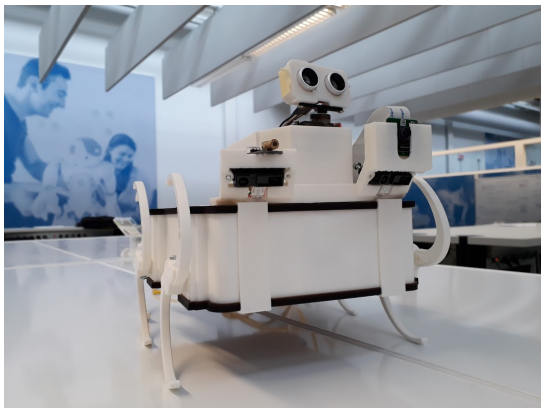
With a current measurement the total average current was measured to be around 450 mA, with occasional peaks to 500 mA maximum.

6.4 Physical Integration in Module Enclosure

The complete hardware for the Environment Observation module is confined in one enclosure. An impression of the enclosure can be seen in Figure 6.4a and Figure 6.4b. A detailed description of the design process of the enclosure can be found in [2].



(a) 3D-model of the module enclosure (not yet finalized).



(b) The enclosure attached to a Zebro. This was one version previous to the 3D model shown in (a).

Figure 6.4: *Visuals of the module enclosure. The laser will be placed on the left protrusion, the camera will be attached to the right protrusion. The Raspberry Pi will be placed in the middle of the closed area, above the PCB.*

The laser should be placed at the exact horizontal distance of 10 cm from the camera and at an angle of 80°, as specified before (see Sec. 4.1). Especially when the angle differs slightly, this can have a big effect on the accuracy of the distance measurement. In the case that the angle is not correct, the *Laser angle* needs to be adjusted accordingly in the code (see Appendix A.1).

The height at which the camera is placed does not affect the distance measurement. For the Zebro to be able to distinguish between scalable and non-scalable objects, it is advised to place the laser at the maximum scalable height. This way, the laser ranger doesn't detect anything when the obstacle is scalable, and non-scalable obstacles are detected.

Chapter 7

Test Results

7.1 Validation Laser Rangefinder

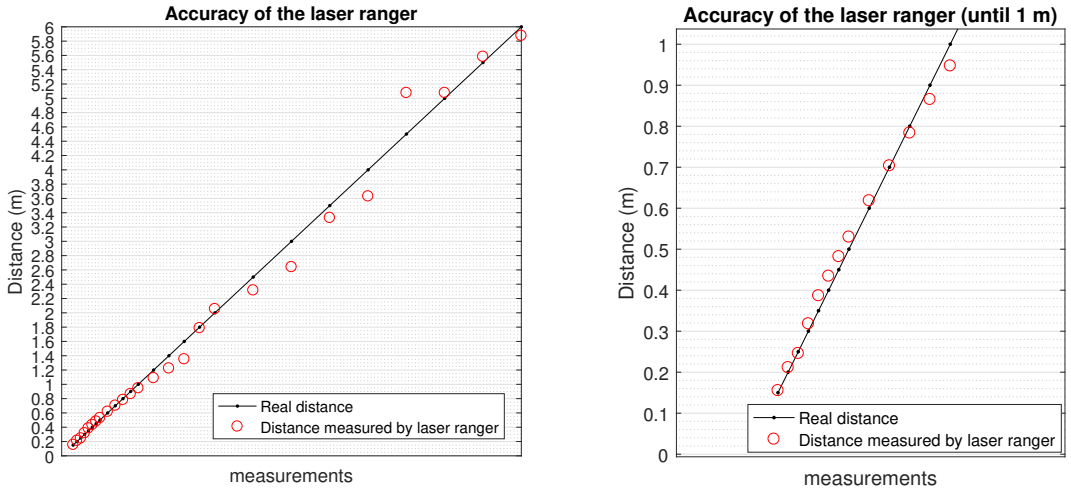
For testing the laser rangefinder, the setup (as in Figure 4.1) was made. Data from the RPi was retrieved using serial communication with an Arduino and a serial monitor on a PC.

For testing the result of the laser rangefinder, the following tests were done:

1. Testing the accuracy of the distance measurement
2. Testing the range in which the laser can be detected for different situations

7.1.1 Accuracy

The first test was done in an indoor environment on an object on which the laser is easy to detect. The real distance to the object was measured using a measuring tape, and compared to the result given by the laser rangefinder. This was done until a distance of 6 meters. The result of the measurements can be seen in Figure 7.1.



(a) *Accuracy from 0.15m to 6m.*

(b) *Accuracy from 0.15m to 1m.*

Figure 7.1: *The accuracy of the laser ranger distance measurement at different distances. In close range the accuracy is very good, but starts decreasing after 1 m.*

From the Figure, it is clear that the laser rangefinder is very accurate at close range. Until about 1 m, deviations are between 0 - 4cm. These small variations can also be partially caused by measurement errors. At further distances, the accuracy decreases, leading to variations from 10 to 60 cm. This is caused by the

resolution (cm/pixel) that decreases at larger distances, which was explained in Section 4.2. The minimal detectable distances lies around 15 cm. At smaller distances, the laser is not in the camera's field of view.

7.1.2 Maximum Range

During the design process, it became clear that different situations and obstacles can affect how well the laser rangefinder is able to detect the laser spot. Therefore, the maximum distance at which the laser can be detected was determined in the following situations:

- Indoor medium-lit environment, with a grey, white, and black obstacle.
- Outside in the shade, with a grey, white, and black obstacle.
- Outside in the sun, with a grey, white, and black obstacle.
- A red brick wall, in direct sunlight and in the shade

A few of these situations are shown in Figure 7.2.

(a) *Outdoors in sunlight, grey object*

(b) *Indoors, black object*

(c) *Outdoors in shade, white object*

(d) *Brick wall in direct sunlight*

Figure 7.2: *The laser pointing at different colored objects in different situations. The laser is often hard to detect by the human eye in the pictures*

The grey obstacle is considered easy to detect a laser spot on. The choice for the black and white obstacles comes from that these are the two 'extremes' on which it is the hardest to detect a laser on. Black objects absorb a lot of light, whereas white objects reflect a lot of light.

Table 7.1: The maximum distance at which the laser is detected for various situations and object colors

Situation	Object color	Max. detectable distance (m)	Notes
Indoors	Grey	6	Max. measured distance
Indoors	White	6	Max. measured distance
Indoors	Black	1.20	R too low
Outdoors shadow	Grey	2.2	R too high
Outdoors shadow	White	1.3	R too low
Outdoors shadow	Black	0.7	R too high
Outdoors sunlight	Grey	0.65	R too high
Outdoors sunlight	White	0.65	R too low
Outdoors sunlight	Black	0.55	R too high
Outdoors shadow	Red brick wall	1.30	R too high, especially when Zebro itself is in the sun
Outdoors sunlight	Red brick wall	0.30	R too high

From the results shown in Table 7.1, it becomes clear that the performance varies greatly between different settings. It is harder to detect the laser in bright areas, than in darker areas. Especially direct sunlight on an object makes it hard to detect, as the contrast between the laser and the surrounding area is not as clear anymore.

A black obstacle absorbs a lot of the laser's light, making it hard to detect a red contour on the surface when the minBGR value is set too high. A white object has the problem that the minBGR value is sometimes not high enough, such that the laser cannot be distinguished from the white area surrounding it (which also has a high R value). Because both are influenced by an increase/decrease of the minBGR value there exists a trade-off between the different situations.

Furthermore, sunlight on a white object also increases the amount of false detections. On white often contours are found that are not lasers, and in direct sunlight the white will become so 'bright' that the HSV values are also higher than minHSV, which leads to contours falsely being judged as laser. In this case, Zebro can determine its own laser by setting it off and on (as described in 4.4), but this takes some extra time. An increase of the minHSV's saturation (S) value could overcome this problem as in this case the laser spot's saturation value is usually higher than its surroundings, but this can also decrease the performance in other situations in which this is not the case.

Another important thing to note is that the Pi Camera's automatic white balance also sometimes decreases the detection range. Figure 7.3 shows the same wall in the shade. In (a) the camera is in the shade too, but in (b) the camera is in the sun. Because of the camera being in the sun, the camera's white balance changes making the wall and it surroundings appear darker. This makes it harder to detect the laser on the wall (because of R being too high).

(a) *100 cm*

(b) *110 cm*

Figure 7.3: *The same brick wall at 100 and 110 cm. Because the camera is in the sun in (b), the camera adjusts its white balance, making the wall appear darker.*

7.1.3 Discussion of results

According to requirement [1.1], the Zebro should be able to detect obstacles. Even in the case of direct sunlight making the laser detection very difficult, Zebro was still able to detect an obstacle at about 30 cm distance. This can still be considered a well-enough distance for Zebro to react on this information in time. Therefore, requirement [1.1] is met. Furthermore, the laser rangefinder is very accurate at distances until one meter, which makes the system reliable such that obstacles can be approached closely without bumping into them. Distance measurements further away are less accurate, but since high accuracy at large distances is not a strict requirement for Zebro's functionalities, this can be considered acceptable.

7.2 Validation QR-code Detector

For testing the QR-code detector, the QR-code shown in Figure 7.4 was printed out in different sizes and held in front of the camera. For each size, the maximum distance at which the QR-code was detected was measured. These results are shown in Table 7.2.

Table 7.2: *Max. detectable distance for different QR code sizes*

QR-code side length (cm)	**Max. detectable distance (cm)**
4.0	26
5.5	46
6.75	76
8.8	98

Figure 7.4: *"Hello Zebro!" QR-code*

7.2.1 Discussion of Results

Currently, the QR-detection does not perform very well. For example, a QR code put on a Zebro would probably be of side length 5.5 cm at maximum. A maximal detectable distance of 46 cm would then require another Zebro to approach it very closely before it can read it. For large QR-codes a range of around 1 m

can be considered reasonable, but it all depends on the application.

The current QR-code detector can be viewed as a proof of concept that probably needs improvement for certain applications. Finally, what should be noted is that the QR-code detector proves that it is possible to add another vision-based functionality next to the laser rangefinder and the motion detector.

7.3 Validation Motion Detection

7.3.1 Setup

In order to analyze the performance of the motion detection, an RC car has been used to model the movements of a Zebro. Several scenarios have been tested in different locations. The locations that were picked as representatives are:

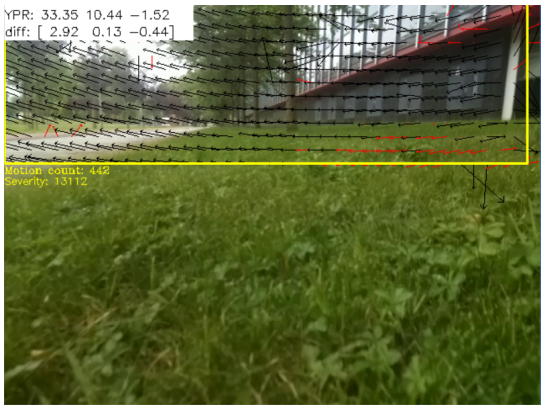
- Inside, on a table.
- On the campus, on grass.
- On a biking trail.
- On a bus lane.

For most of these locations, a number of scenarios has been tested.

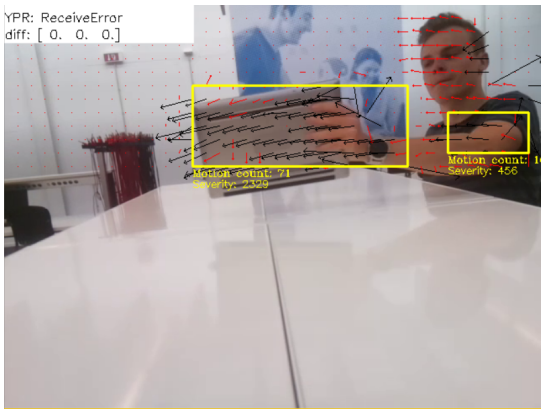
1. Driving without obstacles.
2. Driving to a stationary obstacle.
3. Driving to an obstacle that moves perpendicular to the motion of the car.
4. Driving to an obstacle that moves towards the car.
5. Standing still, obstacle moves perpendicular.
6. Standing still, obstacle moves towards the car.

For practical and technical reasons, not all possible combinations of location and scenario have been tested, but the results should still reflect the overall performance. Figure 7.5 gives an idea of the testing environments.

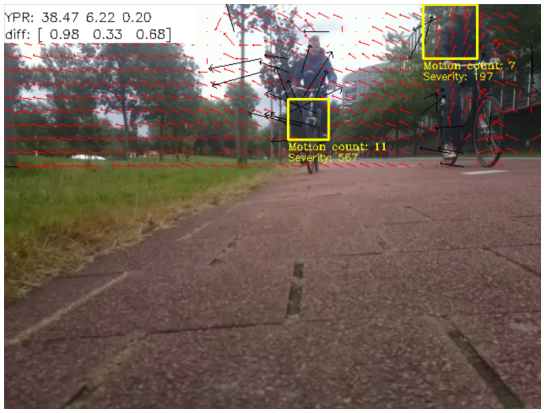
(a) *Sometimes, stationary objects are detected.*



(b) *Camera egomotion still leads to false positives, see Section 9.2.*



(c) *Perpendicular motion works best.*



(d) *Detecting approaching bicyclists.*

Figure 7.5: *Different locations in which the motion detection has been tested. An image of the bus lane is shown in Figure 5.4. The upper-left corner shows the current gyroscope data in Yaw, Pitch, Roll, and the difference with the previous frame*

7.3.2 Results

The results of the tests that are described in Section 7.3.1 are shown in Table 7.3. Each testing video clip has been assessed on two criteria: whether the obstacle was detected and if there were false positives.

Table 7.3: *Test results for the motion detection. The test number definitions can be found in Section 7.3.1. All these results can be verified and analyzed by running "Interpret_motion_data.py", found in Camera_Submodule/Motion_Detection/Testing.*

Test	Table	Grass	Cycling road	Bus lane
1	✓ No false positives	✗ No false positives		✗ No false positives
2	✓ Detected obstacle ✓ No false positives	✓ Detected obstacle ✗ No false positives		✗ Detected obstacle ✗ No false positives
3		✓ Detected obstacle ✗ No false positives		✓ Detected obstacle ✓ No false positives
4	✓ Detected obstacle ✓ No false positives		✓ Detected obstacle ✓ No false positives	✓ Detected obstacle ✓ No false positives
5	✓ Detected obstacle ✓ No false positives		✓ Detected obstacle ✓ No false positives	✓ Detected obstacle ✓ No false positives
6	✓ Detected obstacle ✓ No false positives	✓ Detected obstacle ✓ No false positives		✓ Detected obstacle ✓ No false positives

7.3.3 Discussion of results

Test 1 - driving without obstacles - appears to do worse than with obstacles. This is due to the fact that the car was driving faster than in the other tests. What is most notable from the results is that there are still many false positives. There are two main reasons for this:

- Apparent motion due to the adjusting of the white balance at the start of a video. This is only a problem from the testing videos, since the camera will be on all the time when in operation.
- The car ego-motion. The situations in which this causes false positives were when the car drove faster and with sharper turns. This will be discussed in Section 9.2.

The obstacle was nearly always detected (the moving ones were always detected), although with the stationary object it was not able to detect it until just before collision. However, since Zebro has two other sensors for detecting stationary objects (ultrasonic, laser), this is not a problem. By considering the moving objects as 'dangerous', requirement [1.3] is met.

7.4 Timing

In order to determine the total frame rate, the total run times of all the functionalities that are currently implemented have been measured. See Table 7.4. For completeness: all computations are done on a Raspberry Pi Zero W.

Table 7.4: *Run times of the separate functions. In reality, these functions run in separate threads so that the QR-code scanner does not stall the more important functionalities. When it all works together, a frame-rate between 7 and 11 frames per second is achieved.*

Function	Min. time (ms)	Max. time (ms)	Avg. time (ms)
Find laser in *smallROI* and calc. distance	37.7	60.9	45.1
Find laser in *bigROI* and calc. distance	41.9	180.7	152.5
Motion Detection (little motion)	37.2.6	90.3	65.9
Motion Detection (much motion)	57.4	170.5	108.6
Scanning QR codes (found)	562.5	593.1	570.2
Scanning QR codes (not found)	409.6	431.8	419.2

Chapter 8

Conclusion

For the Zebro Environment Observation Module, a Camera Submodule was designed. This submodule helps keeping Zebro safe by detecting obstacles, either static or moving. Furthermore, it was integrated with the rest of the Environment Observation Module.

Using a laser rangefinder based on triangulation, Zebro can detect and measure the distance to an object in front of it. By placing the laser at the height of a maximum scalable object, scalable and non-scalable objects can be distinguished. At distances shorter than 1 m the distance is accurate to a couple centimeters, and at larger distances the system still gives good enough results for Zebro's perceived uses. The maximal detectable range is 6 m or larger, but in practice this is often decreased due to environmental light and object color. In worst-case scenario the range can be reduced to 30 cm, but this is still considered a large enough distance for Zebro to avoid an obstacle.

Moving obstacles also pose a potential threat to Zebro. Motion detection is done by analyzing all the motion vectors in every frame and grouping possible threats together. By tuning the detection parameters, Zebro can become either more curious or more shy. Detected motion can be used by Zebro to act accordingly, this can be making sound and/or lying down. During tests, obstacles were detected in 16 out of the 17 test cases. However, in 5 cases, there were also false positives due to the camera motion. Solutions are proposed in the Recommendations.

To demonstrate the possibilities of the camera, a simple QR-detector was also implemented. This was not in the initial requirements, but it turned out this feature would be appreciated by the Zebro team. It also showed how easy it is to add additional features to the Camera submodule.

The laser and camera are placed at Zebro's front on the module enclosure. The main processing unit of the Environment Observation Module is a ATmega2650. The Camera Submodule, that runs on a Raspberry Pi Zero, can successfully send its own acquired information to the main processor using serial communication. This is information about at what angle and distance an object is detected, how much motion is detected between which angles and possible detected QR-codes.

The main function requirements of the submodule have been met. With the Camera Submodule, Zebro is able to detect obstacles (Requirement [1.1]). Low obstacles are not detected by the laser and moving objects are considered extra dangerous, so Zebro is able to discern between scalable and dangerous obstacles (Requirement [1.3]). It is able to detect and read QR-codes (Requirement[4.1]) and it can communicate with the rest of the module using UART (Requirement [4.2]).

Chapter 9

Recommendations

9.1 Laser Rangefinder

For the laser ranger some ideas could not yet be implemented and tested. There are several ways the laser ranger can be improved when a higher detection rate and range is desired:

- Replacing the laser with a stronger one, which is the easiest method. A strong laser is also more 'dangerous', making this not the best option.
- Making the minBGR value dynamic and adjusting it to the situation. For example, the minBGR can be determined each frame using the average BGR value in the ROI. If this works, it could increase the detection range for very dark and very light objects for which now the R value is sometimes too high or too low.
- Instead of testing BGR and HSV separately on set values, a threshold could be set for the sum of the HSV and BGR values. In some situations the laser can be easily located using the HSV threshold, but not using the BGR threshold (or the other way around). Therefore, setting a combined threshold (e.g. $V + R > 300$) could possibly make detections better.

9.2 Motion Detection

From the results in Section 7.3.2, the biggest cause for error is the lack of ego-motion compensation. Specifically when the Zebro is making a sharp turn, a lot of motion is observed. There are several solutions to this problem:

- One option is the use of the built-in video_stabilization property [22]. This sounds promising, but can only compensate for vertical and horizontal motion, not rotation.
- If this does not work properly, another option is to find the motion vectors that are most common, and subtract these from all vectors. This, too, does not take rotation into account.
- There is another option that does have the potential of compensating for rotation. This is through the use of the gyroscope that is connected to the Arduino. This approach is used in [7] and the hardware has been implemented, but has not yielded satisfactory results so far. This was partly due to timing constraints. To experiment with this solution, the Zebro team is encouraged to have a look at the testing data, by running "Interpret_motion_data.py", found in Camera_Submodule/ Motion_Detection/Testing. These tests already include per-frame synchronized gyroscope data, as shown in the upper left corner of Figures 7.5.

Another interesting idea is to discern between obstacles that are coming towards the Zebro and those that are passing by. With the Danger class, this can be done by calculating the divergence. If the obstacle is passing by, the divergence is approximately zero.

9.3 QR Code Detection

The QR code detection can be improved by using another method than using the ZBar package. A method that seems promising can be found in [24]. Using this method we developed a code for detecting of the QR code's corners, but not further than that. The reason was that it was too time consuming for a functionality that strictly fell out of the scope of the project.

9.4 Other Proposed Ideas

Some other functionalities for the camera submodule were explored but not implemented due to the short duration of the project. We would still like to mention some of them, as it may serve as an inspiration for future projects within the Zebro team.

- The camera module can serve as the 'Black Box' of a Zebro. The picamera package offers the functionality of recording to a circular stream [25], like surveillance cameras. Whenever Zebro encounters something of interest (errors included), this can be recorded, including the fragment before the moment of interest. Naturally, it is also possible to take pictures. This data can be helpful for debugging purposes.
- There is a lot of documentation to be found on using HAAR-cascades with OpenCV. HAAR-cascades can be used for object recognition in an image. Using a lot of pictures, it is possible to train your own classifier that is able to recognize a specific object of choice [26]. This can for example be a Zebro.
- OpenCV already provides some HAAR-classifiers, like for face- and eye recognition [27]. This can be used so Zebro can know whether it is noticed by a person (because their eyes/face point at the Zebro). This information can be used to influence Zebro's behaviour.
- Water is still a potential hazard for Zebro, especially when the water is not substantially lower than the surface Zebro walks and it cannot be detected as a 'cliff'. Visual water detection can be a solution, but requires a complex texture segmentation algorithm like in [28].

Appendices

Appendix A

Important Variables in Submodule Code

A.1 Adjustable Variables in laserclass.py

Variable	Type	Original value	Description	Change when
minHSV1	tuple	(125,10,100)	HSV threshold for bigROI	Adjust when red contours are not classified correctly as laser/no laser
minHSV2	tuple	(100,0,75)	HSV threshold for smallROI	Adjust when red contours are not classified correctly as laser/no laser
minBGR	tuple	(0,0,220)	BGR thrsehold for finding contours	Change when the laser contour is not found too often
xstart_big, xend_big	int	148, 640	Left- and right x-coordinates of bigROI	Adjust when laser angle is changed, and the laser can have an x-coordinate outside of this range.
ystart_big, yend_big	int	260, 360	Upper- and lower coordinates of bigROI. Can be small because the laser will always be in the same height of the frame when it is placed to shine straight forward.	Adjust when laser height is changed, or when it does not shine directly straight forward
smallroi_width, smallroi_height	int	90, 40	Width and height of smallROI	Should be increased when the laser is 'lost' too often (can be caused by low framerate)
perc_thresh	int	0.85	Percentage of pixels within HSV range for the contour to be classified as laser	Adjust when red contours are not classified correctly as laser / no laser
laser_angle	int	80	Angle at which the laser is placed	This should match the real exact laser angle. When the laser is not placed at exactly this angle, this can have big consequences for the calculation of the distance, especially in far-range
notfoundmax	int	10	ROI changes to bigROI after this amount of subsequent frames in which the laser is not found	Should be lowered when the laser is 'lost' too often (e.g. because of low framerate or high Zebro walking speed)

A.2 Relevant Classes, Constants and Variables in MotionAnalysis.py

Name	Description	Attributes
Class: *Motion*	Represents one vector such as shown in figure 5.1.	**coords:** Tuple containing x and y coordinates in the *motion_matrix*. **length**: The length of the vector. **of_danger**: Reference to the Danger it is part of.
Class: *Danger*	Represents a group Motions that form one connected entity.	**motion_list**: List of Motion instances that make up the Danger. **xmin, xmax, ymin, ymax**: Scope of the Danger.
Constant: *ROAD_THRESH*	Contant that determines which part of the screen to ignore. The higher this value, the closer it is to the bottom of the screen.	Current value: 13
Constant: *MOTION_THRESH*	Threshold value for making a Motion object. Any vectors with smaller lengths will be ignored.	Current value: 20
Variable: *motion_data*	Three-dimensional NumPy array that is provided by the encoder.	**'x', 'y'**: Same coordinates as in the *Motion* class. **'sad'**: Quantity of how well the motion vector fits the two video frames.
Variable: *lengths*	Two-dimensional NumPy array containing all the lengths of *motion_data*.	
Variable: *motion_matrix*	Two-dimensional List that contains all the Motion instances of the frame.	

Appendix B

Bill of Materials

€/ 10 → price for one when ordering 10 at once

Component	**Part number**	#	**€/ 10**	**€/ 100**	**Reseller NL**	**€/ 100**	**Reseller World**
Buck Converter	173010578	1	8.32	6.70	Farnell	0.50	AliExpress
Temperature sensor	MCP9701-E/TO	4	0.94	0.712	Farnell	0.712	Farnell
Gyro/accelero	MPU-6050	1	7.24	6.97	Farnell	2.30	DX.com
Light intensity sensor	LDR	1	0.871	0.743	Farnell	0.045	AliExpress
Humidity sensor	SHT21	1	4.10	3.74	Farnell	1.86	AlieExpress
Speaker	ABT-414-RC	1	1.81	0.896	Farnell	0.09	AliExpress
Distance sensor	HC-SR04	1	3.75	3.55	Antratek	0.79	AliExpress
Cliff sensor	GP2Y0A21YK0F	2	8.66	7.18	Farnell	3.45	AliExpress
Laser	KY-008	1	1,95	1,95	hobby-electronica	0.67	AliExpress
Pi Camera		1	17,97	17,97	Farnell	17,78	AliExpress
Raspberry Pi Zero W		1	11	11	Kiwi Electronic	11	Kiwi Electronic
SD Card for RPi (8 GB)		1	7,40	7,40	Kiwi Electronic	5	AliExpress
Pi Camera (long) flex cable		1	4,43	3,93	Kiwi Electronic	1,70	AliExpress
Microcontroller	ATmega2560	1	12.67	10.51	Farnell	3.80	AliExpress
Loose components PCB	Various	1	5.82	4.55	Farnell	4.55	Farnell
PCB Manufacturing		1	8.82	2.89	Euro-Circuits	0.66	ALLPCB
Plastic Enclosure		1	13.14	11.68	3Dhubs	11.10	3Dhubs (China)
Total			**130.37**	**111.68**		**71.59**	

Bibliography

[1] S. van Leeuwen and P. Goris, "Safety and sensor submodule," Bachelor thesis, TU Delft, June 2017.

[2] L. de Herdt and J. M. Buis, "Obstacle and cliff detection for robotics applications using miniaturized sonar and ir distance triangluation," Bachelor thesis, TU Delft, June 2017.

[3] C. Ttofis, C. Kyrkou, and T. Theocharides, "A low-cost real-time embedded stereo vision system for accurate disparity estimation based on guided image filtering," *IEEE Transactions on Computers*, vol. 65, no. 9, pp. 2678–2693, Sept 2016.

[4] J. Zbontar and Y. LeCun, "Stereo matching by training a convolutional neural network to compare image patches," *CoRR*, vol. abs/1510.05970, 2015. [Online]. Available: http://arxiv.org/abs/1510.05970

[5] W. Enkelmann, "Obstacle detection by evaluation of optical flow fields from image sequences," *Image and Vision Computing*, vol. 9, no. 3, pp. 160–168, June 1991.

[6] W. J. Kim and I.-S. Kweon, "Moving object detection and tracking from moving camera," in *2011 8th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, Nov 2011, pp. 758–759.

[7] T. Low and G. Wyeth, "Obstacle detection using optical flow," in *in Proceedings of the 2005 Australasian Conf. on Robotics & Automation*, 2005.

[8] K. Souhila and A. Karim, "Optical flow based robot obstacle avoidance," *International Journal of Advanced Robotic Systems*, vol. 4, no. 1, p. 2, 2007.

[9] S. Sousa Filho and F. C. Flores, *Attribute Operators for Color Images: Image Segmentation Improved by the Use of Unsupervised Segmentation Evaluation Methods*. Cham: Springer International Publishing, 2017, pp. 249–260.

[10] P. Sidike, D. Prince, A. Essa, and V. K. Asari, "Automatic building change detection through adaptive local textural features and sequential background removal," in *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, July 2016, pp. 2857–2860.

[11] S. Nissimov *et al.*, "Obstacle detection in a greenhouse environment using the kinect sensor," *Computers and Electronics in Agriculture*, vol. 113, no. C, pp. 104–115, April 2015.

[12] V. Hanumante *et al.*, "Low cost obstacle avoidance robot," *International Journal of Soft Computing and Engineering (IJSCE)*, vol. 3, no. 4, 2013.

[13] N. Gageik *et al.*, "Obstacle detection and collision avoidance using ultrasonic distance sensors for an autonomous quadrocopter," University of Wrzburg, Aerospace Information Technology, Tech. Rep., 2012.

[14] G. Fu *et al.*, "An integrated triangulation laser scanner for obstacle detection of miniature mobile robots in indoor environment," *IEEE/ASME Transactions on Mechatronics*, vol. 16, no. 4, pp. 778–783, 2011.

[15] C. De Wagter, S. Tijmons, B. D. W. Remes, and G. C. H. E. de Croon, "Autonomous Flight of a 20-gram Flapping Wing MAV with a 4-gram Onboard Stereo Vision System," *IEEE International Conference on Robotics & Automation (ICRA)*, pp. 4982–4987, 2014.

[16] E. C. Sobel, "The locust's use of motion parallax to measure distance," *Journal of Comparative Physiology A*, vol. 167, no. 5, pp. 579–588, 1990. [Online]. Available: http://dx.doi.org/10.1007/BF00192653

[17] H. Lamela, J. R. Lopez, and E. Garcia, "Low cost rangemap obtaining for mobile robots based on a triangulation rangefinder," in *Industrial Electronics, Control and Instrumentation, 1997. IECON 97. 23rd International Conference on*, vol. 3, Nov 1997, pp. 1284–1287 vol.3.

[18] P. Kalden and E. Stern, "Development of a low-cost laser range-finder (lidar)," Master's thesis, Chalmers University of Technology, 2015.

[19] (2016) Camera module - raspberry pi documentation. [Online]. Available: https://www.raspberrypi.org/documentation/hardware/camera/

[20] K. Hufkens. (2016) Raspberry pi camera v2: spectral response curve. [Online]. Available: http://khufkens.com/2016/06/05/raspberry-pi-camera-v2-spectral-response-curve/

[21] (2017) Recording motion vector data. [Online]. Available: http://picamera.readthedocs.io/en/release-1.12/recipes2.html#recording-motion-vector-data

[22] (2016) Picamera api. [Online]. Available: http://picamera.readthedocs.io/en/release-0.8/api.html

[23] S. Igor. (2017) I2c communication in raspberry pi. [Online]. Available: http://radiostud.io/howto-i2c-communication-rpi/

[24] P. Bharath. (2014) Opencv: Qr code detection and extraction. [Online]. Available: http://dsynflo.blogspot.nl/2014/10/opencv-qr-code-detection-and-extraction.html

[25] (2017) Splitting to/from a circular stream. [Online]. Available: http://picamera.readthedocs.io/en/release-1.12/recipes2.html#splitting-to-from-a-circular-stream

[26] Cascade classifier training. [Online]. Available: http://docs.opencv.org/trunk/dc/d88/tutorial_traincascade.html

[27] Face detection using haar cascades. [Online]. Available: http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html

[28] P. Santana, R. Mendona, and J. Barata, "Water detection with segmentation guided dynamic texture recognition," in *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Dec 2012, pp. 1836–1841.