Technology
Arts Sciences
TH Köln

University of Applied Sciences Cologne

Special Aspects of Mobile Autonomous Systems

REPORT

Autonomous Object Hunting

by:

TIM MENNICKEN

MANUEL AUDRAN

ROBERT ROSE

Cologne, February 5, 2020

Abstract

The purpose of this project is to build a mobile robotic system that is able to move in an arbitrary environment while searching for an object. It avoids obstacles, draws a map of the already navigated way and "hunts" (searches and identifies) for an unspecified object. The exact shape of the target object is variable and can be specified in the source code. The hunt is over, if the object is identified with at least a pre-defined certainty.

For the purpose of object detection a camera module in conjunction with image processing in a neural net is used. The system has several sensors to monitor its own movement as well as orientation in space and to detect surroundings. Movement is archived through several driven wheels. The camera module is able to move in an angle of 360 degrees.

While the basic navigation is carried out on the mobile system itself, path mapping and object detection are outsourced to external hardware. This requires a reliable way to transfer data from and to the appended hardware. Any device that supports WiFi as well as TCP/ IP is able to communicate with the system. As the system should work in any location without the need of fixed network infrastructure (e.g. a router), it creates a WiFi hotspot for external devices. The system offers a TCP socket on a defined port. If any device establishes a TCP connection, data can be exchanged via well defined data structures.

This work implements design and realization of the hardware as well as the software of the system. The software is designed object oriented. All used hardware components are instantiated through regarding software classes. Furthermore an inter-process communication between different systems is realized via TCP/ IP connections. The system is working reliable and fulfills most of the tasks defined in the beginning of the project. Only the drawing of a map of the surroundings could not be implemented completely.

There are two minor problems, which could not be fairly addressed in this work. On the one hand the used distance sensors are not able to deliver reliable measurement results in all circumstances, on the other hand the synchronization between implemented threads and processes can be improved further. The impact of these inconveniences are overcome by a proper error handling, which admittedly not cures the root reason.

Contents

List of Figures

List of Tables

1 Introduction

This research project has the objective to design a system that is able to navigate on its own in an unknown environment, search and identify an object[1] and build a map of the navigated route as well as a map of its surroundings. The system as a whole will consist of a mobile device, further named robot, and an auxiliary external computer. As the robot on its own offers limited computing power, it will outsource computing intensive tasks to the external computer. Hence multiple processes on different platforms will be involved. To be able to operate in any arbitrary location, the system shall be independent of any additional hardware as for example networking devices. Besides it is desirable to design the system in such a manner, that supplementary software, which is not planned now, can be added easily by additional external computers. Figure 1 shows a potential connection between the robot, i.e. the symbol in the middle, and other devices via a wireless LAN[2].

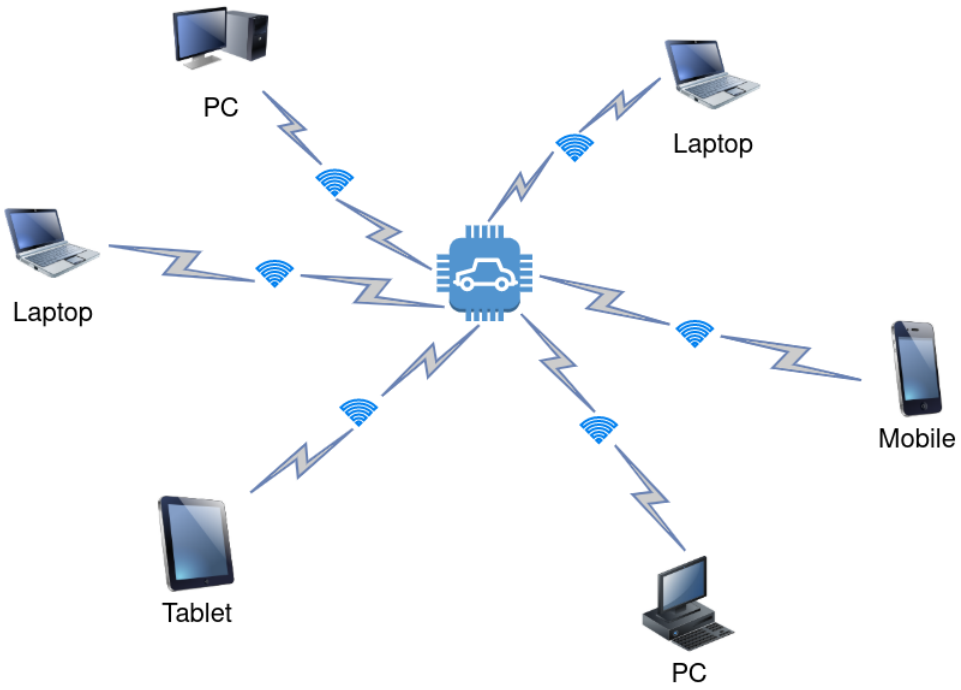

Figure 1: Overview of possible connections

The robot will steer by controlling multiple motors, which will be attached to wheels. Furthermore it will be able to capture its surroundings and any obstacle near it by distance measurement sensors. They will be mounted around the robot in order to register surroundings in any direction. Additional sensors will keep track of the orientation in space and movements of the robot. The target object will be identified by processing data of a visual sensor, i.e. a camera module attached to the robot.
The aim of the robot is to avoid collisions in any environment and find the target object. When the target object is found it shall stop navigating and indicate the successful identification of the object to the user.

[1] The action of searching and identifying an object will further be referenced as object hunt.
[2] A LAN (local area network) is a computer network that interconnects computers within a limited area.

2 Hardware

The hardware of the device can be separated in two subsystems. On the one hand, the robot, which will navigate in the environment, on the other hand the external computer with a static location. This section only focuses on the hardware of the robot. The hardware of the external computer is not further described in this report as any recent computer running that is running Microsoft Windows, Apple iOS or a linux distribution, with a working WiFi interface can be used.
Following you will find a short description and the use case of each part used in the robot. All parts mentioned in this section will be combined to one mobile robotic system.

2.1 Chassis

The chassis forms the foundation of the robot. It consists out of two identical floors, arranged above each other. Drive motors, wheels, revolution sensors and the power unit are mounted on the first floor. Attached to the second floor are the processing unit, PCB board, ultra sonic sensors, smart movement sensor and the camera unit. Each floor has several holes in order to lay the cables from processing unit and PCB board properly.

2.2 Gear Motors

Every movement the robot as a unit can make is realized through four direct current gear motors, each attached to a wheel. Since the processing unit can not operate the gear motors directly, dedicated motor drivers are needed in order to operate the gear motors properly. The L293DNE quadruple half-h drivers [1] are suited and will be used in this project. Each motor driver can operate two gear motors forwards as well as backwards.

2.3 Stepper Motor

The stepper motor is connected to an attachment for the camera module. A stepper motor with 5 V DC operating voltage, 64 steps per revolution and four phases is chosen. The stepper motor can be driven by a single L293DNE [1] motor driver. This allows to turn the camera module theoretically by an arbitrary angle. However, attention should be paid to the cable, which connects the camera module to the processing unit. It is long enough to enable the camera module some free moving space. If it is stressed to much, the equipment can be damaged permanently.

2.4 Revolution Sensors

A brake plate is mounted to each gear motor. With this plates the revolution of the respecting motor can be measured. The plates offer a resolution of 20 counts per revolution, which is sufficient precise for this project. A TCST1103 [2] optical sensor attached to each brake plate allows to get notification of the status of the regarding brake plate. A resistor connected in series with the phototransistor output

leads to a signal fluctuating between ground and the applied voltage, depending on if the light of the emitter can pass the brake plate or is blocked by it. An auxiliary circuit consisting of an operational amplifier is interposed and outputs a clean edge, if the fluctuating voltage surpasses a threshold value. This edge can be detected by e.g. a microcontroller. In combination with the wheel diameter the revolution measurement is used to calculate the distance traveled as well as the speed of the robot.

2.5 Ultra Sonic Sensors

In total four ultra sonic distance sensors are mounted on the robot. The HC-SR04 [3] model is used for all ultra sonic distance sensors. They allow the robot to measure the distance to obstacles in the front, in the back, to the left and to the right. As the HC-SR04 operates on 5 V, a voltage divider has to be interposed between processing unit and each HC-SR04 module. The ultra sonic sensor information is crucial for the robot to navigate, as it is the only information it can receive about its surroundings.

2.6 Smart Movement Sensor

The smart movement sensor is used to get information about the orientation of the robot in space. This is needed in order to locate the robot with respect to the starting position of the hunt. Furthermore this sensor can be used to read acceleration data of the robot. The sensor communicates via an I^2C[3] interface.

2.7 Power Unit

For the robot to be mobile a rechargeable power source is needed. A portable power bank with a voltage of 5 V and li-polymer battery cells is used. It provides an output current of 3 A, which is sufficient to allow simultaneous control of all attached motors and electronics. Furthermore it offers a capacity of 20000 mAh, to allow the robot exploration of spacious environments without recharging.

2.8 Processing Unit

The processing unit of the robot is one of the most important arts of this project. there are several specifications, which it needs to comply with in order to realize the objectives of this project. The specifications are the following:

1. Low power consumption

   As the robot is battery driven, the processing unit should consume as low power as possible to extend battery life.

2. Extensive and accessible GPIO interface

[3] I^2C (Inter-Integrated Circuit) is a two wire, synchronous, serial computer bus invented by Phillips Semiconductor in 1982.

The sensors and actuators are connected directly or via auxiliary circuits to the processing unit. Therefore it is important to have sufficient GPIO pins for all planned hardware.

3. Ability to create a WiFi hotspot

   The robot needs to maintain its own WiFi network in order to operate in any location and be independent of additional hardware.

4. Supports TCP/ IP[4]

   The connection to external computers needs to be bidirectional and reliable. TCP/ IP satisfies these specifications and is a state of the art solution.

5. Supports I^2C

   I^2C is needed for communication with the smart movement sensor (see section 2.6).

6. Supports multithreading

   The robot needs the ability to perform various tasks simultaneously. therefore a CPU[5] with multiple cores is inevitable to archive the objectives of this project.

Taking all specifications into consideration, a Raspberry Pi 4 model B is selected to be used as the processing unit of the robot. It fulfills all mentioned specification, is less expensive than comparable products and offers a big community. Since the Raspberry pi 4 is available in different versions, differentiate in the available RAM, the version with 4 GB RAM is chosen. This enables potential for further additions.

[4] TCP/ IP, or the Transmission Control Protocol/ Internet Protocol, is a suite of communication protocols used to interconnect network devices.

[5] Central Processing Unit

2.9 Entire Overview

The resulting hardware, without power supply, of the robot and its interconnections can be seen in figure 2.

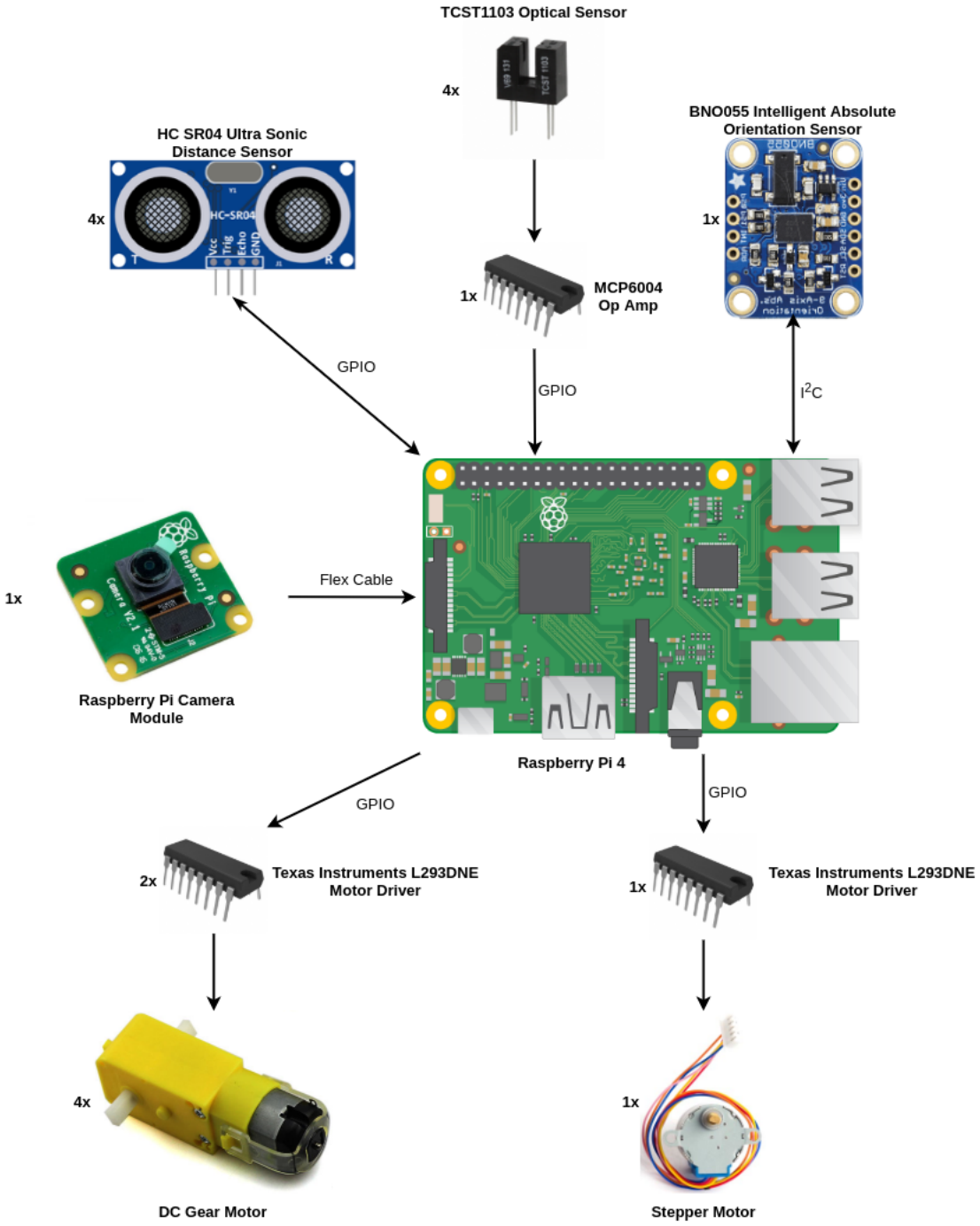

Figure 2: Combined hardware

2.10 PCB Board

The wiring of all components is expected to be rather complex. Since breadboard connections are prone to errors and can get unclear, if multiple connections are involved, a PCB board is designed

for the circuit. Furthermore each device will be connected by a dedicated connector in order to allow easy assembly and disassembly of the system as a whole as well as replacements of single devices.
At first a test circuit is designed for every part. In this manner the function of each sub circuit can be controlled and potential modifications can be implemented before designing the PCB. When each circuit is working as desired, the single sub circuits are combined in one system wide circuit and transferred to an electronic design automation software. In this project the education version of EAGLE [6] is used.
The circuit can be assigned to a two layer PCB board of the dimension 82 mm x 65 mm. It is designed to be easily plugged in on top of the 40 pin GPIO connector of the Raspberry Pi. The mounting holes have the same dimensions as the ones of the Raspberry Pi. Hence the same screws to install the Raspberry Pi, are used to attach the PCB board. All GPIO contacts are lead through to allow easy measurement of signals while the PCB board is installed on top of the Raspberry Pi. Figure 3 shows the fully populated PCB board.

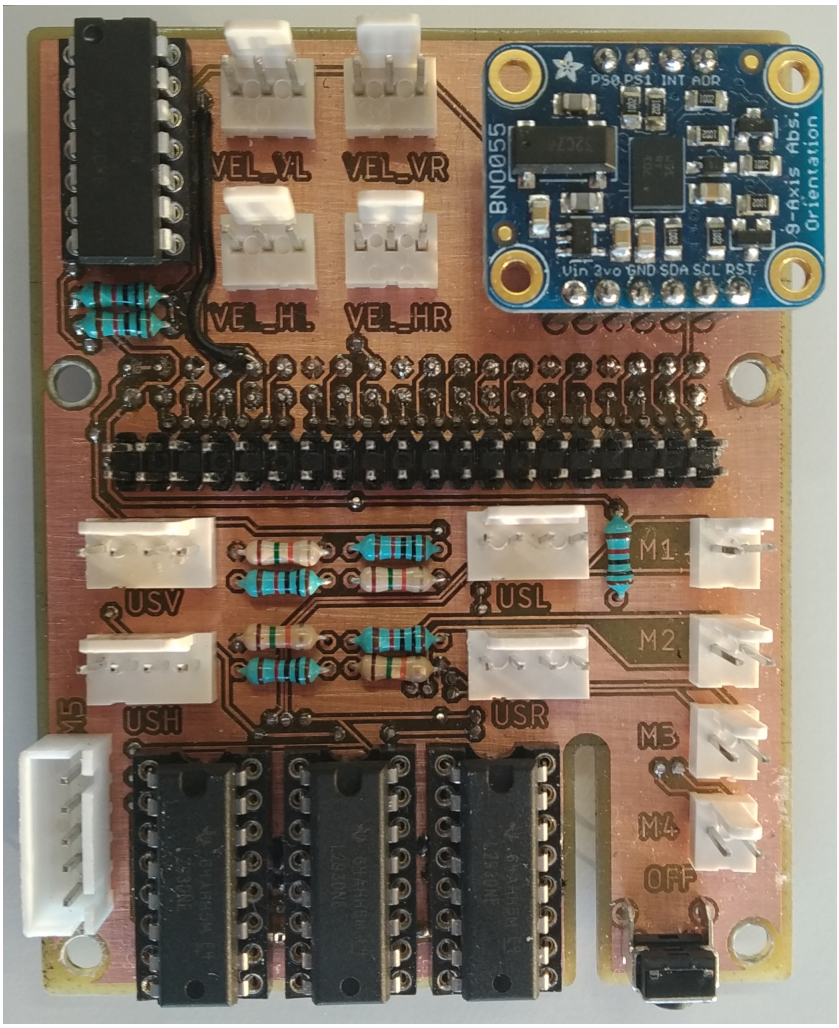

Figure 3: Populated PCB board

3 Internal Software

The in this section described processes are internal processes, i.e. running on the robot itself. In total there are three custom processes running on the robot. Figure 4 shows the structure of the developed software. This section focuses on the base and the navigation process. The image streaming process is referenced in section 4.

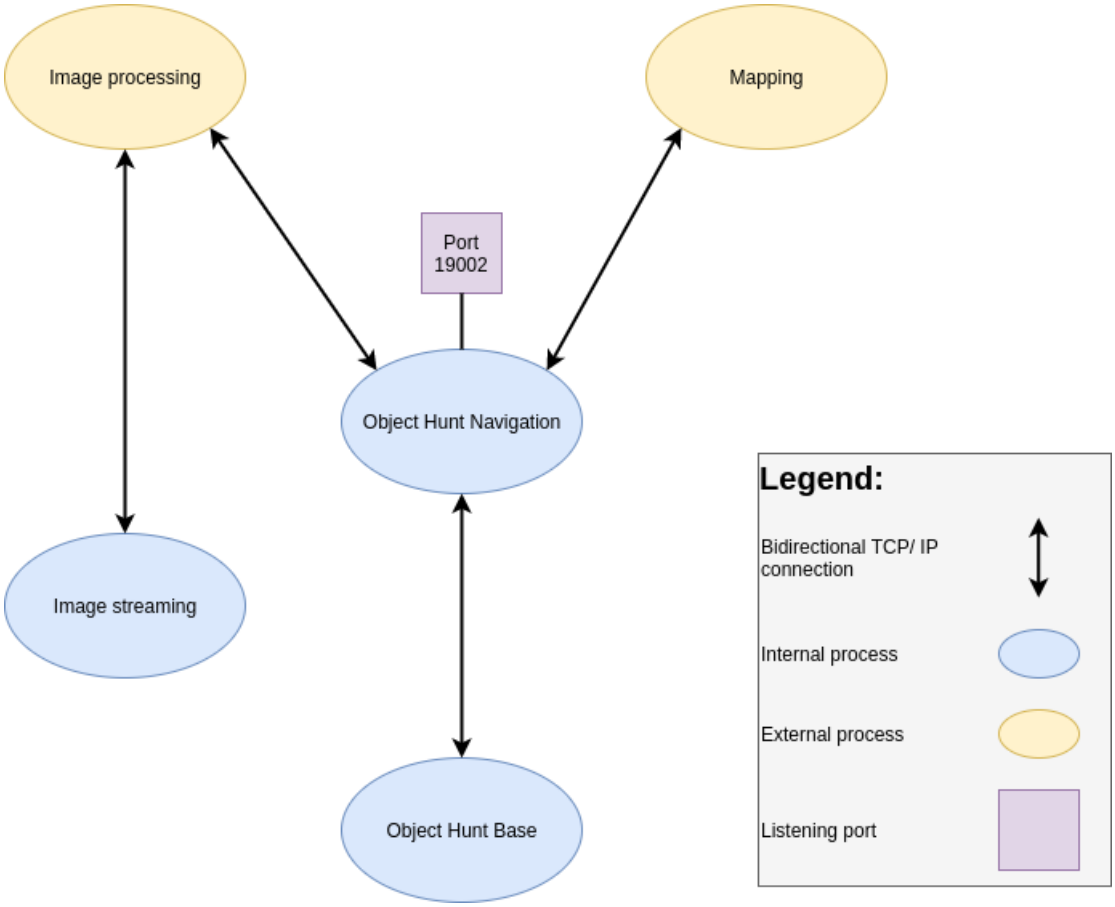

Figure 4: Software structure

3.1 Design Principles

The system in its entirety shall not be bound to any programming language. In that way virtually any language that supports the usage of raw sockets in collaboration with TCP/ IP can be used to develop software for the system. At the time of writing this report Java, Python and C++ are already employed.

Furthermore the internal software underlies particular specifications, which are defined at the beginning of the project. They define, among other things, how the system as a whole will behave and handle dedicated tasks, i.e. inter-process communication. The compliance of these principles ensure that the developed software works in the expected manner and offers the possibility for adding further processes in the future.

The defined principles are listed in the following:

- Communication
  - Via TCP/ IP
  - Independent from other devices
  - Beyond system border
- Economical
  - No unnecessary CPU consume
  - Fast reaction times
  - Event driven
- Documentation
  - Complete
  - Easy accessible

3.1.1 Communication

All participating processes communicate via TCP/ IP. Therefore a purpose-built way of exchanging data is developed. Messages are generally separated in action identifiers, commands, requests and responses. Table 1 declares the usage of each message type.

Action identifiers	This messages consist of exactly one byte in all implementations. It is put in front of each other message type and tells the receiving process what data is about to come in.
Commands	Gives a specific order to the base process. Only the base process can handle commands. A command does not implicate any result, so the sending process will get no feedback, if the command succeeded or not.
Requests	Asks for the cars state. That can be in particular a reading of one or several sensors, speed or orientation data. Every request implicates a response with the desired value. Each request contains an identifier field.
Responses	A response message will always follow an earlier request. It can either indicate an invalid request or contain the desired data. Each response contains an identifier field. It will adopt the identifier of the respecting request.

Table 1: Definition of messages

There exist several implementations of each message type. A motor will most likely be controlled by a respecting command message. A sensor measurement on the other hand can be triggered by a correct request. Please refer to the project documentation page [7] for further information about which messages exist and their exact composition.

3.1.2 Economical

The application shall be both CPU as well as time efficient. Therefore the participating processes are implemented in a multi-threaded manner. Each thread gets created at initiation and is dedicated to one task. A thread will only be active, if the respecting task needs to executed. Otherwise the thread will be in an idle state and consume no unnecessary CPU time. Tasks are getting triggered by certain events, i.e. a hardware interrupt, an incoming message, an expired timer and so on. All events are initialized in a way that allows them to get polled from a respecting file descriptor. This allows threads and processes to wait in an idle state for specific events to happen.

In a multi-threaded context one has to take care of simultaneous access of shared resources. Therefore the threads are synchronized via synchronization mechanisms, i.e. mutexes. A mutex can be shared between multiple threads. If one of the threads needs to access a shared resource, it will lock the mutex. When another thread tries to access the same resource in the same time, it would need to lock the mutex as well. When the mutex is locked already, the calling thread is queued until the mutex is unlocked. As soon as the mutex gets unlocked by the initial thread, the queued thread will lock it and access the resource. In this work exclusively scoped locks are used. A scoped lock gets created whenever a mutex shall be locked. The respecting mutex is passed to the constructor of the scoped lock. The scoped lock object will be destroyed, when its scope is left and releases the mutex. This prevents a so called deadlock situation, in which a mutex is locked and cannot be unlocked anymore. A possible scenario for this is, if the locking thread experiences an error and gets stopped before unlocking the mutex.

The communication between different threads is realized through pipes. A pipe offers a writing as well as a reading end. Thus it allows simultaneous writing and reading by two threads without additional synchronization mechanisms. The inter-thread communication is mainly used to exchange information, trigger or abort tasks and notify each other about particular events.

3.1.3 Documentation

A complete and accurate documentation of the source code is produced. The software documentation tool Doxygen [9] is used to generate valid HTML[6] documents from annotated header files. The created files display the relations between various programming structures (classes, namespaces, etc.).

[6] Hypertext Markup Language (HTML) is the standard markup language for documents designed to be displayed in a web browser

Besides developers can add special denoted comments with explanatory content which are going to be adopted in the HTML files. Additional several auxiliary HTML pages with explanatory context regarding the project are created and linked to the pages created by doxygen. All created pages [7] are public accessible via the internet.

Moreover a Github repository [8] is arranged. It contains the source code, helper scripts, mechanical construction files, schematics, PCB files, the documentation and everything else regarding this project. The data contained in the Github repository allows to rebuild the whole system developed in this work from scratch.

The Github repository and the online documentation are linked to each other. It is highly recommended to refer to the online resources for information about the robot as they will be kept up to date in case of changes to the system.

3.2 Base Process

The base process interacts directly with all controllable hardware parts listed in section 2. Therefore it is a crucial process for the correct function of the robot. If the base process crashes unexpected, any other process concluded in the project will be disconnected or shut down as well. The base process uses only standard libraries from the C++14 standard.

Please see figure 5 for a simplified overview of the relations between the contained classes. Please note that the class diagram contains no exception classes for a well-arranged view.

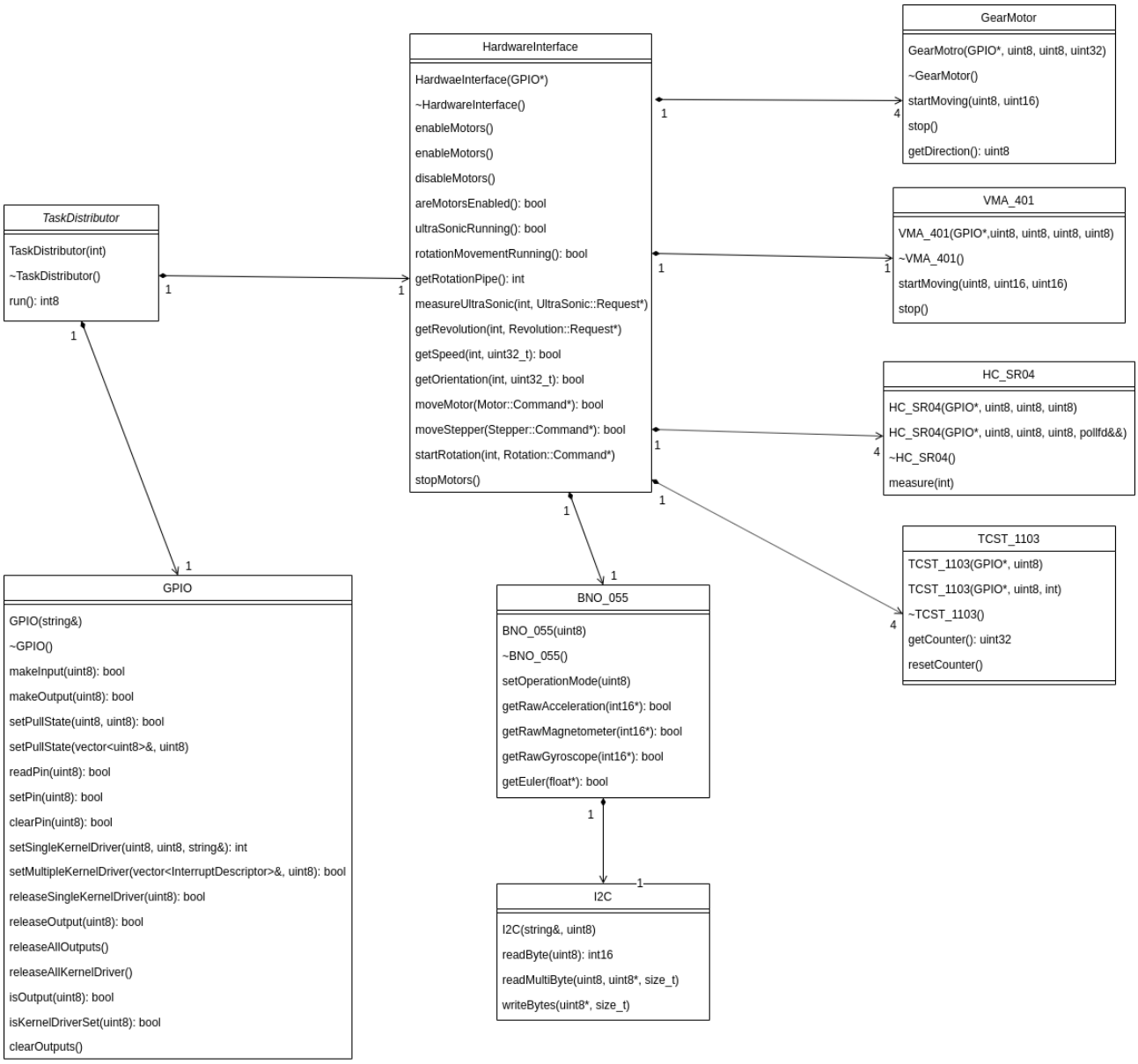

Figure 5: Class diagram of the base process

3.3 Navigation Process

The navigation process establishes a connection to the base process. It requests periodical sensor readings. The responses are computed and depending on the outcome it sends movement commands to the base process. The actual movement of the robot is based on a continuous communication between both processes. The base process will stop all motors, if the navigation process experiences an error and gets shut down. The navigation process uses only standard libraries from the C++14 standard.
Besides the navigation process handles connections to external processes and forwards specific messages to the base process. Therefore it acts as the central point of communication between internal and external processes.

Please see figure 6 for an overview of the relations between the contained classes.

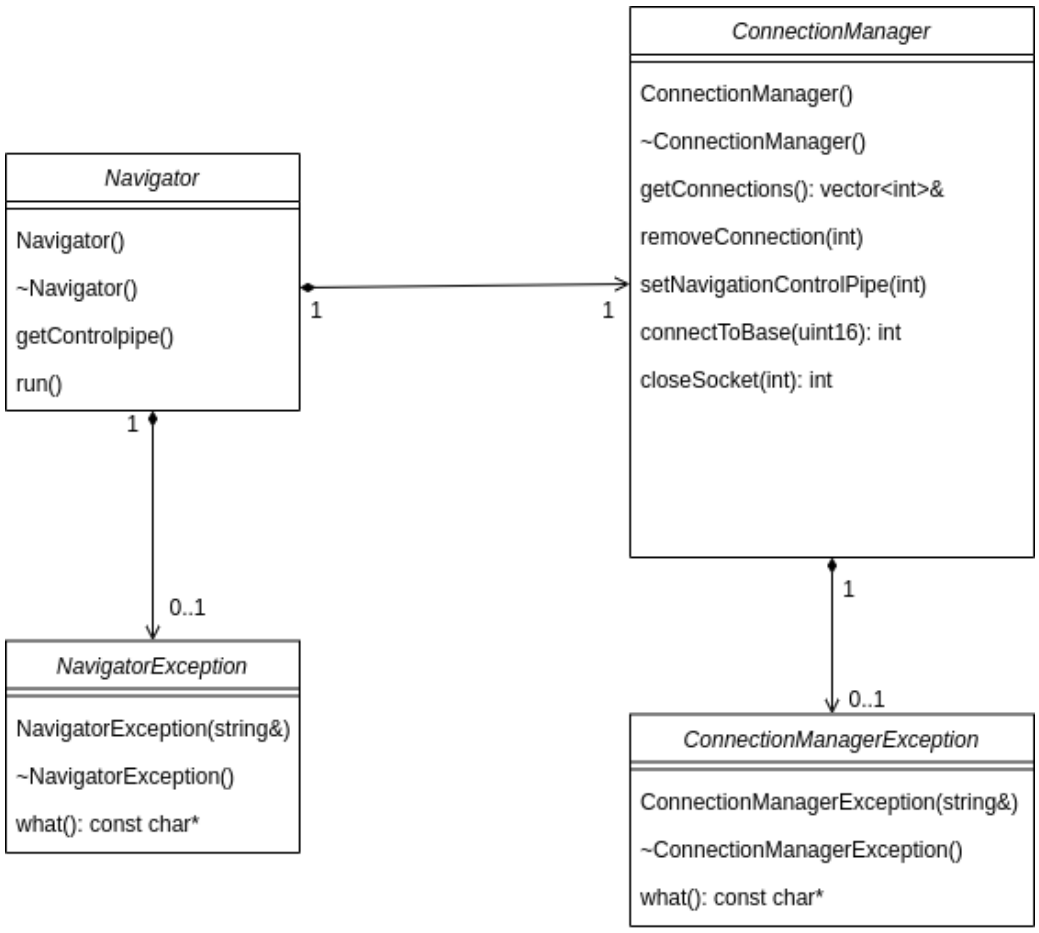

Figure 6: Class diagram of the navigation process

4 Image Processing

4.0.1 Definition of the object detection task

The deep learning computer vision area consists of different tasks, which serve different purposes. In figure 7 we can see the representation of these different tasks. In our project we will take a closer look on the object detection. Like we can see in the image, object detection means detecting multiple objects. But the task it not only to detect and classify them, it is also the task of the program to locate the objects. This will be evaluated on the bounding box around the object. The desired result can be seen in the following figure.

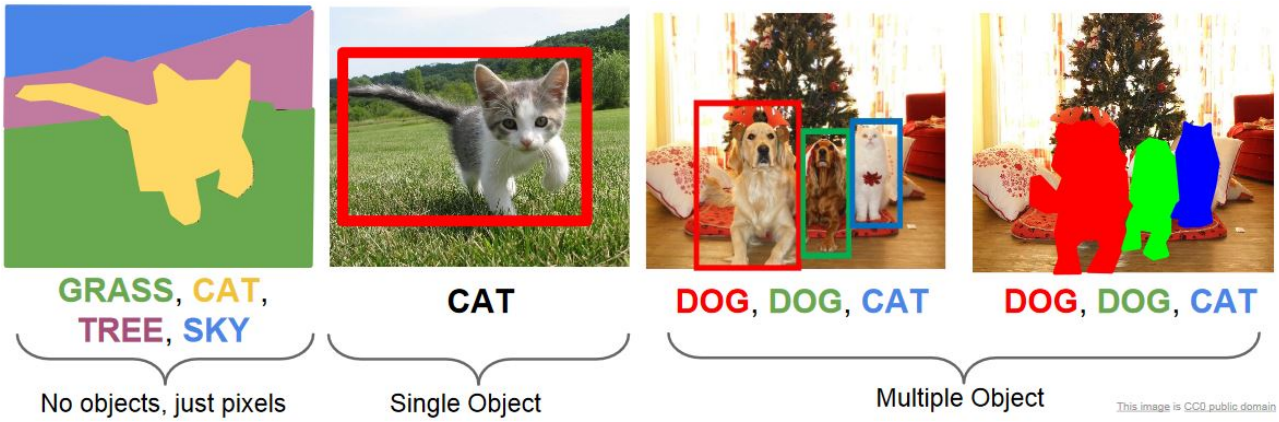

Figure 7: Representation of image processing tasks

4.0.2 Requirements on object detection

In our project we only define the object, which we want to detect. It is not defined how many of these objects will be present, but for the expandability of the project, we will use methods capable of detecting multiple objects.

Also, the image processing needs to be real-time capable. The robot navigates autonomously around an unknown area and should detect the objects, when they appear. It is not desirable to have delays in the detection. For this also a sufficient accuracy is needed to detect the object under different lighting and background conditions.

4.1 Basic model selection

Regarding these requirements, we need a specialized model to match the real-time capabilities, while having a decent accuracy. In the image recognition area, the drive for increasing accuracy leads the progress. But it comes with the cost of increasing complexity of the models. This leads to increasing computational cost, which decrease the real-time capabilities. Mobile phones and embedded system

are not capable of providing this performance. For this purpose, the MobileNet [7] network was created. It drastically decreases the number of trainable parameters with different techniques. The main difference from conventional networks is the depthwise convolution. In the conventional convolutional network, the convolution determines features by applying kernel filters on the different input channels and combines the features to reach a new representation [7].

This results in a computational cost of:

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F$$

The MobileNet splits these two steps up and computes them separately [8, 9]. First the depthwise convolution is applied by applying a single filter per input channel. In the second step the pointwise convolution combines the resulting features. Both operations present an own layer and are followed by a batch normalization and a ReLU activation. This separation results in a new computational cost of:

$$D_K \cdot D_K \cdot M \cdot D_F \cdot + M \cdot N \cdot D_F \cdot D_F$$

Comparing these to costs, the depthwise convolution reaches a reduction of $\frac{1}{N} + \frac{1}{D_K^2}$.

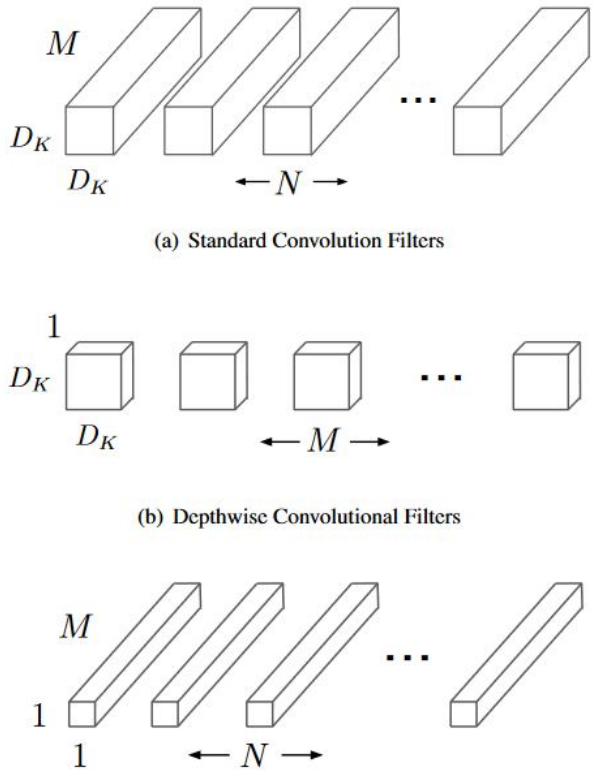

Figure 8: Comparison of standard convolution (a) and Depthwise Convolution [8, 9]

The Mobile network was tested one time with standard convolution and with depthwise separable convolution. The result was only 1,1 % loss in accuracy, but a reduction of 25,1 million parameters.

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

Figure 9: Comparison of standard and depthwise separable convolution

4.2 Object Detection System

For the object detection there are mainly three different techniques, which are used.

4.2.1 Faster R-CNN

One approach is the Faster R-CNN, which is the combination of two modules. First we have a deep convolutional network, which extracts the features. After this the feature map will be shared with a separate network, which predicts the region proposals. In this RPN module the feature map will be fed into two fully-connected layers. One layer computes the box regression and the other one computes the box classification. These predictions will be reshaped through a RoI (Region of Interest) pooling layer and will then be used to classify the found objects within the proposed regions.

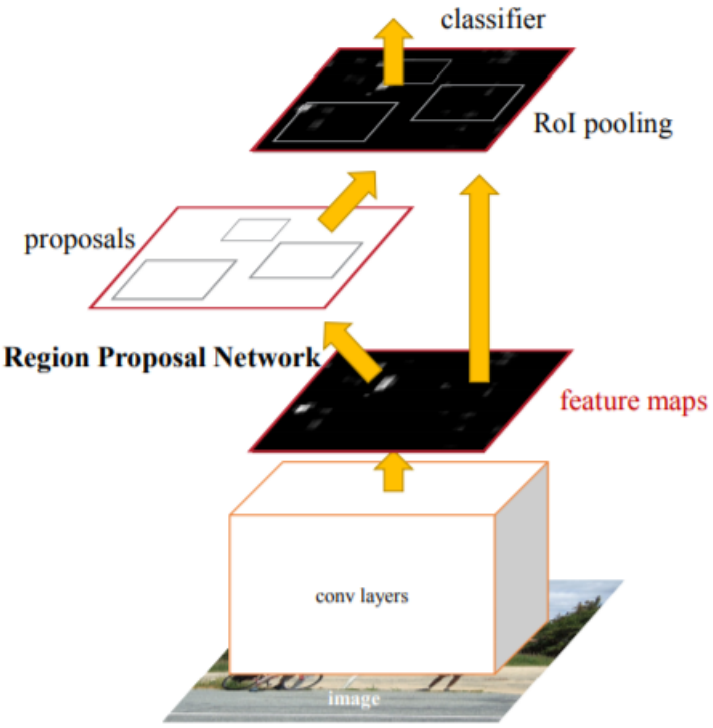

Figure 10: Architecture of Faster R-CNN [8]

4.2.2 You Only Look Once – YOLO

Another approach is to not look at regions with high probability of containing an object, but to split the whole image into a grid and predict first the class probability and secondly the offset for the bounding box for different bounding boxes in this grid. The only probabilities higher than a defined threshold will be taken into consideration. This reduces the architecture to one network and eliminates the bottleneck separate classification and bounding box regression. This approach is called YOLO (You Only Look Once), which is a lot faster than Faster R-CNN, but has the drawback of less accuracy [9].

4.2.3 Single Shot Detection – SSD

The last approach uses the same technique as YOLO, but has some diversifications. Single Shot Detection (SSD) is also a one-stage detection model. It predicts class and bounding box at the same time. Unlike YOLO it will not predict the certainty of the object being an object, its only predicts the class probabilities, but adds a background class. So when thebackground class has a high probability, it has the same effect as a low confidence score from the YOLO prediction.

Also, the SSD uses different grid layouts, which results in more predictions, but also in more accuracy. Therefore, the YOLO architecture delivers better real-time capabilities, because is it less computational intense.

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	$\sim 1000 \times 600$
Fast YOLO	52.7	155	1	98	448×448
YOLO (VGG16)	66.4	21	1	98	448×448
SSD300	74.3	46	1	8732	300×300
SSD512	76.8	19	1	24564	512×512
SSD300	74.3	59	8	8732	300×300
SSD512	76.8	22	8	24564	512×512

Figure 11: Performance comparison of different techniques [11]

4.3 Experiments

For our object detection experiments we used the following software components:

- Python 3.7.3
- OpenCV 4.1.2 [12]
- ImageZMQ [13]
- Imutils [14]
- ssd_mobilenet_v2_coco [15]

We used OpenCV, because it is a well-established computer vision library and designed for fast computation also on systems with low computational capabilities. It offers the DNN (Deep Neural Network) module, which implements methods to import pretrained models from different frameworks. We decided to take the pretrained MobilenetV2 in combination with SSD trained on the COCO dataset. For reasons shown in Section 4.2 and 4.1, we decided this network will give us enough computation speed to detect object while driving and sufficient accuracy to detect the object in different lighting or environment conditions. The COCO (Common Objects in Context) dataset is a commonly used dataset and offers a lot of classes. We defined the "sports ball" class as the desired object to detect but wanted to have flexibility to change the object if needed. Therefore, the COCO dataset was well-suited for our requirements.

The "ImageZMQ" package is a little library with methods to transport the OpenCV images over the network. It uses the ZeroMQ sockets to enable an easy and fast transport.

The "imutils" package is used to build up the presentation of the images.

4.4 Software

Because the images will be sent over network to an external device, two scripts will be started. One is the client script, which runs on the Raspberry Pi. The server script runs on the external device and receives and processes the images. It also sends the acknowledgement to signal the client to send another picture.

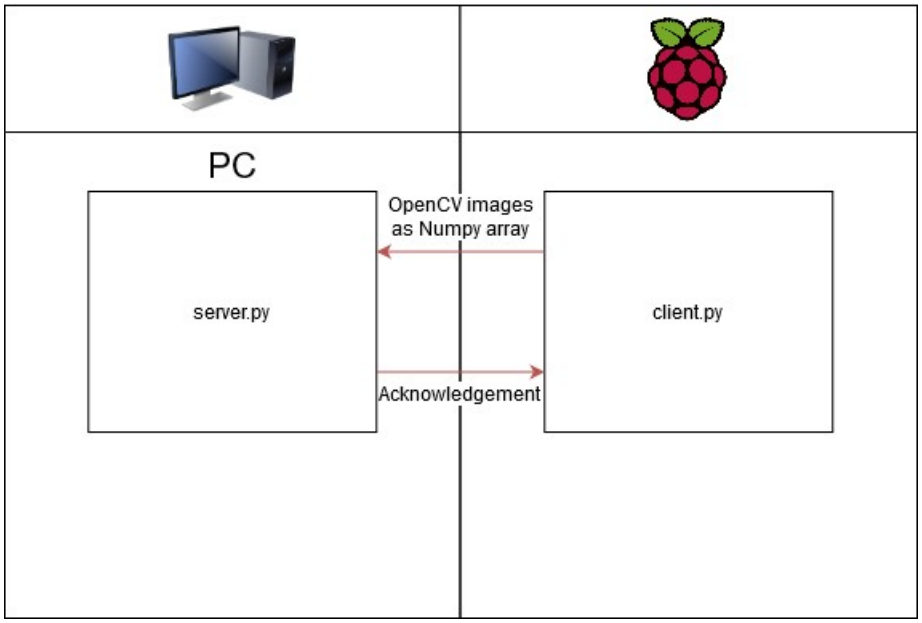

Figure 12: Server client communication

4.4.1 Server.py

The server first initializes the socket to receive the images and the connection to the base process (see section 3.2). Then the network will be initialized and the objects to consider.

The marker for a found object will be set to False before the loop. In the loop the image will be received if there is a connection. If the device is not already known, it will be added to the "lastActive" list. The image will be resized and then processed by the network. If an detection of the desired object occurred and the confidence is higher than the defined threshold, the presentation will be build. The bounding box will be added to the picture and the number of detected objects will be updated. This image will then be shown on the external device. The loop is also interrupted and the connection to the base process will be established to send the message of successful object detection. Then the picture of detection will be shown, until the user interrupts the script.

4.4.2 Client.py

The client initializes the connection to the server and initializes the camera. In an endless loop the client will send the images to the server.

4.5 Results

We could realize all the requirements we set for our object detection task. Our project has shown, that for real-time object detection it is possible to let the image processing run on an external device and still get sufficient results. Our object detection is flexible in detecting different classes, it is possible to choose one of all COCO classes. To get better accuracy and better performance an own dataset will be needed. With an own dataset, the network would only predict two different classes, the background and the desired object, which result in much less predictions and therefore better performance.

5 Conclusion

The system as a whole is working reliable and fulfills almost all tasks specified at the beginning of this project. Special attention is paid to the software design principles defined in section 3.1. While the system is running with full functionality, the average CPU load of the Raspberry Pi is considerable low. At the same time the robot reacts almost immediately to incoming events. This is mainly due to the thoughtful development of internal processes as well as outsourcing computing intensive tasks to external hardware. The robot is capable to handle further functional expansions.

This project was used as an opportunity to implement an economical system containing multiple platforms. The therefore necessary inter-process communication was seen as a special challenge. We decided on purpose against a supportive framework, as ROS[7] for example. On the one hand the usage of ROS introduces a rather impractical build tool (i.e. catkin), which makes it hard to use convenient techniques as remote development as well as remote debugging and restricts the usage of programming languages. On the other hand this project showed us the challenges of synchronizing multiple threads and processes on various devices. The gained experience can be used in future projects.

Moreover the project is well documented and fully open source. All related documents are stored and updated in a public accessible Github repository. State of the art source code documentation tools were used to create an almost consistently documentation of the source code. The created documentation are public accessible as well and linked to the Github repository.

5.1 Known Issues

The present system has known issues depending the hardware as well as the software, which should be addressed in future works.

The designed PCB board is comparatively small compared to the implemented functionality. This requires to use small circuit path sizes. Some circuit paths had an unwanted conducting connection, introduced by inaccuracies in the development process. The elimination of the unwanted connections demanded more time than planned for the PCB board. Furthermore the PCB is mounted right above the processor, which could prevent proper cooling of the Raspberry Pi. In a new design the PCB board would be designed bigger and placed below the Raspberry Pi.

The project revealed insufficient edge detection capabilities of the Raspberry Pi. Especially the used ultra sonic sensors are dependent on a proper edge detection. Most of the effect could be bypassed by modifications in software. However, these modifications introduce inaccuracy to the measurement result. this problem can be avoided by using a microcontroller for the sensor readings and pass the

[7] The Robot Operating System (ROS) is a set of software libraries and tools that help to build robot applications.

results to the Raspberry Pi.

Synchronization and collaboration of the threads implemented in the base process can be improved. Observation of the process log indicates rarely happening errors. This results most times in simultaneous access of shared resources. As for the time being there is no known way to reliable reproduce the error, it is a rather hard task to track and eliminate the cause. The throughout this project gained experience would be used to pay more effort to thread synchronization, in a new design of the software.

The ultra sonic measurements are conducted simultaneously in this project. This leads to mutual influence of the ultra sonic sensors. Therefore the ultra sonic sensors only deliver reliable results, when an object is detected in short distance (i.e below ca. 80 cm). Mainly this prevents the mapping process from drawing an accurate map of the surroundings. A possible solution is to conduct the ultra sonic measurement one after another, although this would be against the multi-threaded principal of this project. From our point of view a different kind of distance sensor is the preferred solution.

6 References

References

[1] Texas Instruments Incorporated, “L293x Quadruple Half - H Drivers – Data Sheet” 2016 [Online]. Available: https://www.ti.com/lit/ds/symlink/l293d.pdf. [Accessed: 31.01.2020].

[2] Vishay Semiconductors, “Transmissive Optical Sensor with Phototransistor Output” 2011 [Online]. Available: https://www.vishay.com/docs/83764/tcst1103.pdf. [Accessed: 31.01.2020].

[3] Elec Freaks, “Ultrasonic Ranging Module HC - SR04” [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf. [Accessed: 31.01.2020].

[4] Adafruit, “BNO055 Absolute Orientation Sensor” 2015 [Online]. Available: https://learn.adafruit.com/adafruit-bno055-absolute-orientation-sensor/overview. [Accessed: 31.01.2020].

[5] Raspberry Pi foundation, “Raspberry Pi 4 Model B” [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/. [Accessed: 31.01.2020].

[6] Autodesk, “EAGLE overview” [Online]. Available: https://www.autodesk.com/products/eagle/overview. [Accessed: 31.01.1020].

[7] Tim Mennicken, Manuel Audran, Robert Rose, “Object Hunt documentation page” [Online]. Available: https://teawolf-beep.github.io/index.html. [Accessed: 03.02.2020].

[8] Tim Mennicken, Manuel Audran, Robert Rose, “Object Hunt Github repository” [Online]. Available: https://github.com/Teawolf-beep/Object-Hunt. [Accessed: 03.02.2020].

[9] Dimitri van Heesch, “Doxygen” [Online]. Available: http://www.doxygen.nl/. [Accessed: 04.02.2020].

[9] Google, “MobileNets: Efficient Convolutional Neural Networks for Mobile VisionApplications” [Online]. Available: https://arxiv.org/pdf/1704.04861.pdf. [Accessed: 02.02.2020].

[10] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks” 2015 [Online]. Available: https://arxiv.org/abs/1506.01497. [Accessed: 02.02.2020].

[11] “One-stage object detection”, [Online]. Available: https://machinethink.net/blog/object-detection/

[12] Fei-Fei Li, Justin Johnson, Serena Yeung, “Lecture 11:Detection and Segmentation” [Online]. Available: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf. [Accessed: 02.02.2020].

[13] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg, “SSD: Single Shot MultiBox Detector” 2015 [Online]. Available: https://arxiv.org/abs/1512.02325. [Accessed: 02.02.2020].

[14] skvark, “opencv-python: OpenCV on Wheels” [Online]. Available: https://github.com/skvark/opencv-python. [Accessed: 02.02.2020].

[15] jeffbass, “imagezmq: Transporting OpenCV images” [Online]. Available: https://github.com/jeffbass/imagezmq. [Accessed: 02.02.2020].

[16] jrosebr1, “imutils” [Online]. Available: https://github.com/jrosebr1/imutils. [Accessed: 02.02.2020].

[17] Tensorflow, “Detection model zoo” [Online]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md. [Accessed: 02.02.2020]. =======

[6] Autodesk, “EAGLE overview” [Online]. Available: https://www.autodesk.com/products/eagle/overview. [Accessed: 31.01.1020].

[7] Google, “MobileNets: Efficient Convolutional Neural Networks for Mobile VisionApplications” [Online]. Available: https://arxiv.org/pdf/1704.04861.pdf. [Accessed: 02.02.2020].

[8] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks” 2015 [Online]. Available: https://arxiv.org/abs/1506.01497. [Accessed: 02.02.2020].

[9] “One-stage object detection”, [Online]. Available: https://machinethink.net/blog/object-detection/

[10] Fei-Fei Li, Justin Johnson, Serena Yeung, “Lecture 11:Detection and Segmentation” [Online]. Available: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf. [Accessed: 02.02.2020].

[11] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg, “SSD: Single Shot MultiBox Detector” 2015 [Online]. Available: https://arxiv.org/abs/1512.02325. [Accessed: 02.02.2020].

[12] skvark, “opencv-python: OpenCV on Wheels” [Online]. Available: https://github.com/skvark/opencv-python. [Accessed: 02.02.2020].

[13] jeffbass, “imagezmq: Transporting OpenCV images” [Online]. Available: https://github.com/jeffbass/imagezmq. [Accessed: 02.02.2020].

[14] jrosebr1, “imutils” [Online]. Available: https://github.com/jrosebr1/imutils. [Accessed: 02.02.2020].

[15] Tensorflow, “Detection model zoo” [Online]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md. [Accessed: 02.02.2020]. »»»> fe5836e38c93ccec12af9fab6d4b3a4ae2ada36e