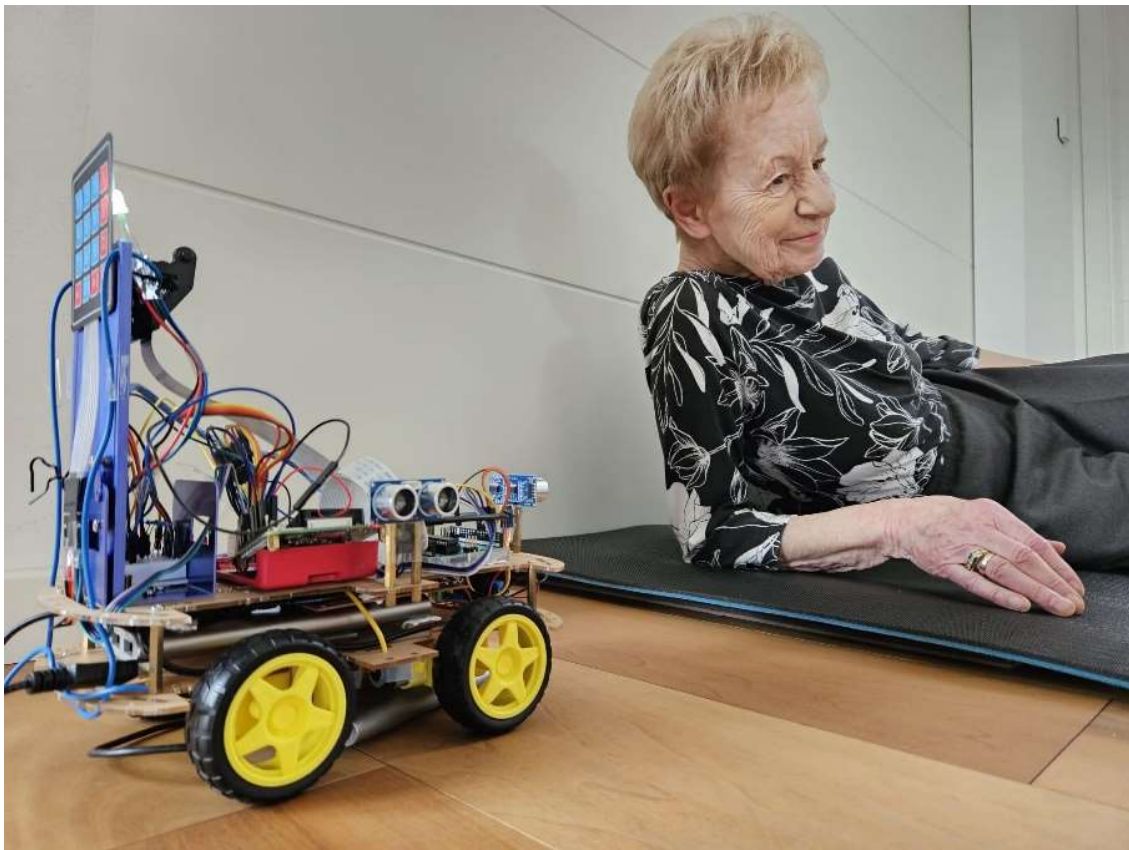


Autonomous robot for fall detection and emergency alerts

Restoring my grandmother's independence



Author: Victoria Hoffmann
Supervisor: Adriana Mikolášková
Maturitátsarbeit
MNG Rämibühl
08.01.2024

Abstract

Elderly people suffer more frequent and more severe falls as they age. To ensure swift treatment for their injuries, I invented a robot that can navigate autonomously with sensors collecting spatial data. Based on a neural network, I trained a computer vision model that enables the robot to detect falls. The robot's alert system allows it to send emails with attached pictures of the person lying on the floor. The main hardware components comprise a Raspberry Pi 4B+ (microprocessor), an Arduino Uno (microcontroller), a camera, four motors, two ultrasound sensors, and a keypad. The robot uses the "Bookworm" operating system and is mainly programmed in Python.

The main results are that the robot correctly identifies "lying people" with an accuracy of over 80%, and that "walking" as well as "chair-sitting" people are correctly classified as "not-lying." However, problems in identification seem to arise when people are sitting on the floor.

Beyond better hardware components, possible improvements include methods to autonomously follow the person instead of searching the entire apartment and additional fall detection systems e.g., using an accelerometer. Incorporating a language processing model and enabling live streaming of relevant images could facilitate an initial medical assessment of the situation.

Contents

PREFACE	3
1 INTRODUCTION	4
2 LITERATURE	6
2.1 AUTONOMOUS NAVIGATION	6
2.2 IMAGE RECOGNITION.....	6
2.2.1 <i>Neural Networks</i>	6
2.2.2 <i>Yolo: A Convolutional Neural Network</i>	8
2.2.3 <i>Platform and Library for Computer Vision Implementation</i>	10
3 ROBOT DESIGN AND DEVELOPMENT PROCESS	10
3.1 SIMPLIFIED MODE OF OPERATION	10
3.2 FUNCTIONALITIES	12
3.2.1 <i>Autonomous Driving</i>	12
3.2.2 <i>Autonomous Object Detection</i>	16
3.2.3 <i>External Alerting</i>	17
3.3 DETAILED MODE OF OPERATION	19
3.4 SOFTWARE AND HARDWARE CONFIGURATION	21
3.4.1 <i>Software Architecture</i>	21
3.4.2 <i>Hardware Architecture</i>	23
3.5 EMPIRICAL TESTING.....	25
4 RESULTS	29
4.1 ONE-PERSON CASE	29
4.2 PERSON-FREE CASE.....	34
5 DISCUSSION	37
6 CONCLUSION	39
7 BIBLIOGRAPHY	40
7.1 TRADITIONAL SOURCES.....	40
7.2 ONLINE SOURCES.....	41
8 APPENDIX	43
8.1 CODE	43
8.1.1 <i>main.py</i>	44
8.1.2 <i>led_final.py</i>	46
8.1.3 <i>movement.py</i>	47
8.1.4 <i>ultrasound_final.py</i>	49
8.1.5 <i>keypad_final.py</i>	50
8.1.6 <i>image_recognition_final.py</i>	51
8.1.7 <i>alert_email_final.py</i>	53
8.1.8 <i>p_key_final.py</i>	54
8.1.9 <i>var_final.py</i>	54
8.1.10 <i>ultrasound_2_direction.ino</i>	55
8.2 EXTENSIONS.....	57
8.2.1 <i>Language processing model</i>	57
8.2.2 <i>Autonomous phone control</i>	60
8.3 MATERIALS.....	60
8.3.1 <i>Software</i>	60
8.3.2 <i>Hardware</i>	61
DECLARATION OF INDEPENDENCE	64

Preface

Ever since I participated in a summer camp on computer science in Cambridge, UK, I have been fascinated by autonomous robots. My passion for computer science further blossomed during my stay at the Harvard Summer School, where I took an intensive course on algorithms and data structures in Java. Taking every opportunity to build my knowledge in this field, my high school matura thesis offered a first chance to pursue the endeavor of creating a robot myself, as I experienced a dilemma in my family: my grandmother's wish to remain as independent as possible versus my parents' and my concern for her well-being. I wanted to develop a robot that would satisfy all the parties involved. Specifically, I became interested in whether an autonomous robot was available—or could be built—that would detect when a person had fallen and notify a third party to call for external help. Surprisingly, neither such a robot nor its underlying code seemed to be available “off the shelf,” and I was intrigued by the possibility of creating one myself. To achieve my objective, I conducted research on each individual hardware component and studied software implementations of the different functionalities I wanted the robot to have—for example, image recognition. I then programmed and manually assembled the hardware and software elements into an interdependent configuration. With every new component a new world opened up—not only for my robot, but also for me. In addition to teaching myself new skills in computer science and robotics, I also learned how to approach solutions from different angles by logically breaking down problems to their core. Inventing this robot combined the fields of hardware architecture, software development, and empirical analysis, each of which I could enjoy in its own way.

Today, I look back on an immensely exciting project with many lessons learned over the course of the year. I especially want to thank my supervisor Adriana Mikolášková for all her support during my project. In particular, she helped me to refine my project and to try out different approaches to certain problems. Furthermore, I want to thank my computer science teacher Christoph Vogel, who supported me not only in finding a way for the robot to cover the whole apartment in its search for a person in need of assistance but also by introducing me to concepts such as UML charts. Finally, I thank the 18 people who tirelessly assumed various positions to help me train and test the computer vision model.

1 Introduction

Switzerland is facing an aging population. In 2021, 19.0% of its population was already 65 or older [101], and this percentage is expected to increase in the coming decades. This aggravates the issue of how and where these seniors are supposed to live—i.e., either in retirement homes, where accommodation is scarce, or at home, where safety cannot always be guaranteed, particularly for those who live alone. For many seniors, it is vital to continue living independently as they age. They usually prefer to stay in the neighborhood they are familiar with. Relatives, however, are interested in being alerted as swiftly as possible in case of an emergency [1]. Thus, there is a need for more support in everyday life—a problem aggravated by the lack of caregivers. Yet, with recent dramatic improvements in technology, caregivers can be supported—and arguably even replaced—by smart devices capable of observing medical conditions, detecting accidents, and alerting help.

For elderly people, falls can be particularly troublesome because they can trigger a rapid deterioration in health. For example, if mobility is reduced due to a broken limb, muscle depletion accelerates. Moreover, falls are problematic if they go unnoticed for a prolonged period due to the person's smaller circle of contacts who could notice said accidents. There are a range of devices that aim to detect falls and alert contacts in the case of an emergency, but all suffer from certain disadvantages, as follows.

Wearable devices, such as alert buttons, are dependent on the user remembering to wear them. They are particularly problematic for individuals suffering from dementia, who tend to remove unfamiliar objects [1]. Furthermore, such devices are of no use when the person is unconscious or simply unable to reach the button. Similarly, monitoring systems that track daily routines are unreliable, as they might falsely alert relatives due to minor changes in those routines, which could be due to simple forgetfulness [12]. A disadvantage common to all these devices is that the alerts they provide offer no information on the person's actual well-being [12]. The use of video surveillance with closed-circuit cameras provides the necessary information but remains controversial due to privacy concerns. Moreover, the presence of cameras might create unnecessary stress [1,12]. Furthermore, the cameras are usually fixed and therefore have blind spots [12]. To minimize those blind spots, a mobile robot equipped with adequate sensors could be used—an approach taken by Takuma et al. [12]. In their device, a computer, a camera, and infrared lasers were mounted on a small table affixed to a vacuum cleaner. The device then measured the distance to the target person, calculated their direction of movement via triangulation, and could thereby follow them. Once skeletal information was collected, falls could then be registered by calculating the difference between the y-coordinates of the head and knees (see Figure 1). A very small distance was considered to be a fall. Takuma et al. encountered several problems with their device, including stability and power consumption, locating a person who had already fallen, failing to recognize a fall when the person's head was outside the camera frame, falsely alerting when a person was simply bending down, and managing the interactions between different software systems [12]. Nonetheless, the authors claim that their device achieved results that were 80% better than those obtained with stationary devices.

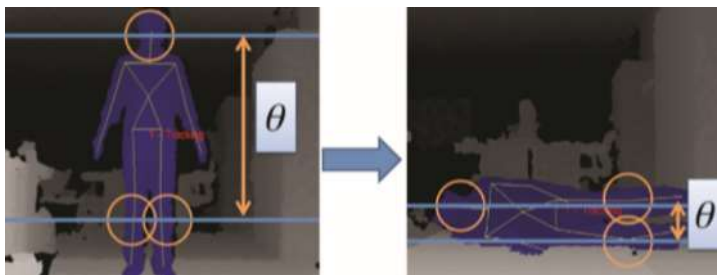


Figure 1: Fall determination using the difference between head and knees according to [12].

Another approach [1] introduces the integration of a security system in a smart illumination apparatus to issue alerts in the event of a fall, fire, or break-in. It is described as being capable of automatically locking a safe room in case of need. Although the concrete implementation is not provided, the detection of emergency situations is achieved with thermography, infrared, and color sensors. These are used in addition to monitoring medical blood pressure and heart rate sensors that are directly applied to the skin [1]. While interesting, this approach raises issues such as individuals potentially feeling restricted in their autonomy. Moreover, the tendency of patients with dementia to remove unfamiliar devices remains.

Yet another device [11] employs machine learning to analyze the center of mass of a moving object. The model is integrated into a home assistant and requires user interaction for feedback when a certain confidence threshold has not been reached, which means that the model could not clearly identify whether somebody had fallen. Thereby the model dynamically learns and can adapt to different environments. However, specifics are not provided, as the patent application provides only a general overview with no indication of any concrete implementation.

Recently, another camera-based approach has been developed, employing a machine learning model to analyze inconsistent behavior [13]. Over time, the model gathers data, learning that frequent falls may be typical for toddlers but are rare and dangerous for elderly people. It would also recognize that while lying on the floor is completely normal in a gym-type environment, the same posture in the kitchen could indicate a concerning event. In addition to its implementation of computer vision algorithms, the device can also work with natural language processing to interpret audio commands. Furthermore, privacy issues should not be a concern, as the visual processing to determine the posture and position of the person in question is based on pixelated images or even skeletons. However, said cameras are fixed and have blind spots, inevitably implying that they cannot capture or detect every situation within their surveillance domain. Such a limitation could potentially result in missed alerts—for instance, when an individual falls in an area not covered by the cameras.

In this thesis, the overarching goal was to enable elderly people to maintain their independence while still ensuring their safety. Therefore, I designed and programmed an autonomous vehicle that navigates through the entirety of an apartment while keeping a lookout for people who are lying on the ground. Detecting people in need has become more feasible thanks to neural networks, which have greatly improved due to optimized algorithms and increased computing power [6].

This thesis first covers a thorough introduction of the concepts utilized by my robot (chapter 2) followed by a detailed account of its design and development process (chapter 3). I then describe the outcomes of my work (chapter 4) and reflect on them in the ensuing discussion (chapter 5). Finally, I draw conclusions from my project (chapter 6). Extensive additional information is provided in the appendix.

2 Literature

As far as I am aware, no autonomous robot designed specifically for fall detection exists that is commercially available. Therefore, in the following chapter I describe the fundamental concepts required for my robot: autonomous navigation and image recognition.

2.1 Autonomous Navigation

Various kinds of sensors can be used to enable spatial navigation. Active sensors such as ultrasound sensors determine spatial information by evaluating the responses to signals they emit. Passive sensors, such as cameras, receive external inputs from their surroundings without further interacting with them. Combining several sensors often yields maximal information, compensating for the shortcomings of individual sensors. For instance, GPS sensors do not perform well indoors, as radio signals often have difficulty penetrating solid walls. Infrared sensors, on the other hand, are highly sensitive to external factors such as surface texture and lighting conditions [9]. Cameras provide information on motion by tracking distinct features, but this approach is computationally expensive and ineffective in poor lighting conditions. Lasers are another means of gathering spatial data, but they are error-prone in large empty spaces or in proximity to glass structures [8].

There are two primary approaches to navigation: map-based and mapless. Map-based examples include “grid-based maps,” where each cell in a grid is marked as either an obstacle or as free space. Meanwhile, “topological maps” represent characteristic features as nodes and define their relationships and proximity as edges [8]. Such maps allow free space to be extracted, enabling path determination with shortest-path algorithms such as the Dijkstra’s algorithm [2]. The term “map-using” refers to scenarios where a map is provided externally, enabling the robot to localize itself and track the direction of its own movements. With a known initial position, the estimated position is continuously updated in a process called “relative localization.” Conversely, in the absence of a known initial position, a beacon or landmark is required for the robot to estimate its own position. This is known as “absolute localization” and is often used for indoor flying robots. However, the requirement for beacons limits this type of navigation to pre-known environments [8].

Conversely, in a mapless scenario, a map is not provided but can instead be constructed beforehand or concurrently with localization—a process known as simultaneous localization and mapping (SLAM), which is a criterion for a robot to be considered as truly autonomous. However, robots can also autonomously navigate without maps by relying on prominent landmarks, e.g., walls, to guide movement [9].

2.2 Image Recognition

Computer vision has recently become very popular, particularly within clinical and medical applications. Neural networks are employed to interpret images and detect objects. This section is organized into three subsections: an overview of neural networks in general; an exploration of Yolo, a convolutional neural network model specifically designed for image interpretation; and a description of tools utilized to implement computer vision.

2.2.1 Neural Networks

Artificial neural networks (ANNs), commonly known simply as neural networks (NNs), were first developed in 1980 [6]. Neural networks are inspired by the human brain, particularly in terms of structure and learning processes. They comprise artificial neurons, which serve as processing units and feature a multiple-input, single-output configuration. The neurons are represented with nodes that are organized into several layers, including an input layer, where data is received, and an output

layer, where the result is determined. The model may also include a variable number of hidden layers situated between the input and output layer [14].¹

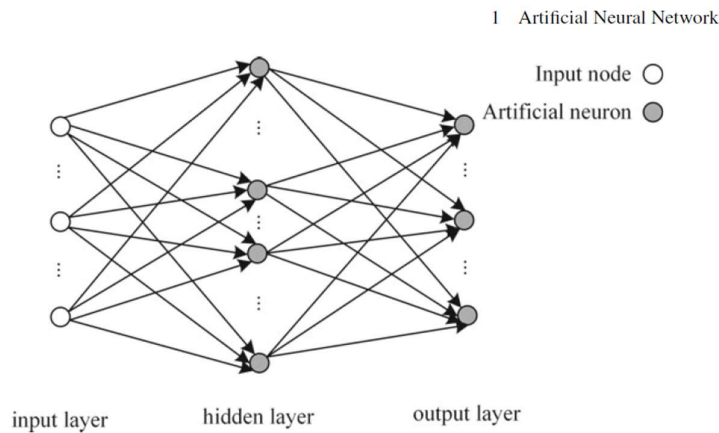


Figure 2: Scheme of a shallow neural network [14]

The nodes within the network are interconnected by edges, each of which is assigned a weight denoting its significance. Initially, these weights are assigned at random, but they are gradually adjusted as the neural network undergoes training (or “learning”). When external input is received by a node of the input layer, this input is multiplied by the weight of the corresponding edge. Considering that each node receives multiple inputs from nodes of the preceding layer, these weight-multiplied inputs are added up. More specifically, the operation performed is known as an “activation function,” which is essentially a weighted summation. Typically, the domain of this function lies between $[0,1]$ or $[-1,1]$. Upon reaching a certain threshold, nodes switch from a state of inhibition to one of excitation or vice versa [14]. There are different forms of learning, i.e., unsupervised and supervised learning [103].

When a neural network learns by adjusting the weights according to predefined rules, without having examples to learn from, this process is known as “unsupervised learning.” An illustration is “competitive learning,” characterized by the “winner takes all” learning rule. The rule stipulates that the node with the highest response value following the weighted summation is allowed to adjust its weights, thereby becoming more significant, while the weights of the non-winning nodes’ edges stay unchanged. In a slight variation, the nodes surrounding the winning node can also slightly adjust their weights.

Supervised learning occurs when the corresponding “real output”² for each input is known. The output of the neural network is calculated by “feeding forward” values, meaning that the values traverse the network without encountering any cycles³. After the neural network’s output is determined, the discrepancy between the output of the neural network and the real output is calculated. This error is

¹ Different terms are used for neural networks depending on the number of hidden layers they incorporate. A network with no hidden layer is referred to as a “single-layer NN.” A network with only one hidden layer is referred to as a “shallow NN” or a “common NN.” This type can be used for any continuous function. A network with two or more hidden layers is known as a “deep neural network,” which can be used for any discontinuous function and is often employed if increasing the number of nodes fails to improve performance.

² The “real output” is the actual or wanted output of a given input.

³ A “cycle” is a structure where the input can circle back to previous layers before the output is determined. In networks without cycles the information is transmitted in one direction only (input layer to output layer) before the output is determined.

then sent back from the output layer to the input layer, adjusting the weights of the edges along the way accordingly. This process of feeding the values forward and sending the error backwards is known as “backpropagation” [14].

For example, if a neural network is trained to distinguish between cats and dogs, the weights are initially set at random. When a picture of a dog is then passed in, the neural network’s initial prediction might turn out to be 0.6 dog and 0.4 cat. In reality, however, the real output would be 1.0 dog, prompting an adjustment of the weights. With these adjustments, the next time a dog image is passed in, the prediction will ideally have improved—so that it might be 0.7 dog and 0.3 cat. However, it is important to use an equal quantity of training data for both categories, as the network may otherwise become biased toward the category on which it has been more extensively trained.

To optimize the neural network, the dataset is divided into training, testing, and validation sets. Furthermore, the number of nodes as well as the number of hidden layers can be adjusted. To determine the optimal number of nodes, their number can be either incrementally increased or decreased from a high starting value. The latter approach works well because during training, the impact of some nodes tends to zero, allowing their potential removal.

Generally, the number of hidden layers is only increased if the neural network does not perform accurately in spite of a large number of nodes. In contrast to making predictions after training, the actual learning phase of the neural network can be time-consuming, as parameters such as weight and number of nodes are not yet fixed and require adjustment. A significant risk during training is that the network overlearns the provided samples, which is more likely to occur with backpropagation. As a consequence, the training indicates minor errors, but the predictions of the trained model are flawed with major errors—this known as “overfitting.” Another unwanted result of training occurs when large errors are already indicated during the learning process, leading to poor predictions—also known as “underfitting.” Both overfitting and underfitting should be avoided during neural network training [14].

2.2.2 Yolo: A Convolutional Neural Network

The human brain analyzes its surroundings primarily through vision, receiving 83% of its information via this sense and 11% through hearing. Therefore, when imitating the way humans perceive their environment, computer vision is fundamental, while natural language processing is also of interest to understand how humans interact. One of the most significant neural networks in these domains is the so called “Convolutional Neural Network,” which also serves as a core component of the robot discussed in this thesis. The Convolutional Neural Network is classified as a deep feedforward network, implying that it incorporates two or more hidden layers through which information progresses in one direction only without cycling back in between. Like all other neural networks, it contains an input layer as well as an output layer. The hidden layers serve varied purposes: the convolutional layer extracts visual features such as outlines and shadows to reduce the number of connections in the neural network; the pooling layer reduces the size of the image; and the fully connected layer has the role of backpropagation in the neural network [14,3].

Convolutional neural networks are employed to analyze images due to their proficiency in recognizing patterns. The convolutional neural network known as “Yolo” is a significant development in the field of computer vision. Its name—an acronym for “you only look once”—refers to its unique architecture. Contrary to the usual approach of splitting the process of image analysis into “feature extraction” and “regression and classification,” Yolo employs a single-stage detection process, making it more suitable for real-time image analysis [3].

Yolo-v8 is an improved version of Yolo that utilizes unsupervised and supervised learning in combination [3,102]. Furthermore, it generates so-called “anchor-free” models. Anchor boxes are rectangular areas positioned within images where the neural network most frequently predicted bounding boxes during training. A bounding box is the rectangular output area indicating where an object was detected. It is common to predict an object’s location by calculating the offset from anchor boxes [103].

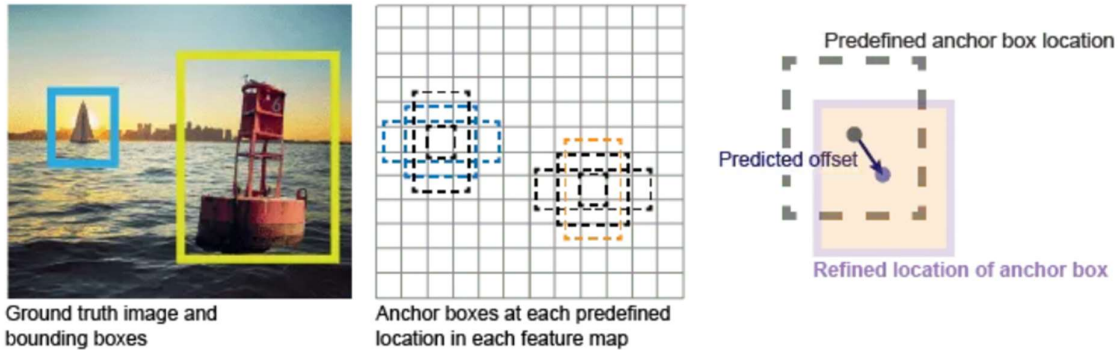


Figure 3: Anchor boxes in old Yolo versions [103]

However, Yolo-v8 improves this process by localizing only the grid cell containing the center of the object, while disregarding the surrounding grid cells [3]. As a result, the model is much more efficient, as the number of anchor boxes is significantly reduced [10]. To analyze the performance of the model, several situations must be distinguished:

- TP – A “True Positive” evaluation relates to a situation in which the true class of an object has been correctly identified, e.g., there was a dog which was correctly identified as a dog.
- TN – A “True Negative” evaluation relates to a situation in which there is no object and the model has correctly not identified any object, e.g., there was no dog and nothing was categorized to be a dog.
- FP – A “False Positive” evaluation relates to a situation in which the object has not been associated with the right class. Examples are that there is no object but the model has incorrectly identified one, or there is a dog that is incorrectly identified as a cat.
- FN – A “False Negative” evaluation relates to a situation in which there is an object but the model has failed to identify, e.g., there was a dog which has not been identified as a dog.

Accuracy can then be calculated as the fraction of true evaluations over all evaluations, with

$$\frac{TP+TN}{TP+TN+FP+F} \quad (1)$$

Precision is calculated as the fraction of True Positives among all positive evaluations, with

$$\frac{TP}{TP+FP} \quad (2)$$

[14]. This approach has been used for my analysis.

2.2.3 Platform and Library for Computer Vision Implementation

In my work, I have utilized several tools to implement the principles described in the previous paragraphs, the most important of which are Roboflow and OpenCV.

Roboflow is a platform that enables users to annotate images and train computer vision models. It is particularly user-friendly, as Roboflow's training stops as soon as overfitting begins. However, "being a proprietary product, the algorithmic details about the network are not disclosed to the public" [10, page 6].

OpenCV is an end-user friendly, open-source library primarily used to identify, classify, and track objects. The library contains built-in methods for processing visual inputs and analyzing their patterns [5].

The most widely known methods are [4]:

- To import an image: `cv2.imread(path, flag)`—if the path is invalid, an empty matrix is returned
- To save an image: `cv2.imwrite(filename(inc. extension), image)`
- To display an image: `cv2.imshow(windowname, image)`
- To close the window: `cv2.destroyAllWindows()`

3 Robot Design and Development Process

In this chapter, I first describe the robot's overarching, simplified mode of operation (3.1), followed by the functionalities of the individual elements (3.2) and the detailed mode of operation (3.3). I then set out the configuration of the most important software and hardware components (3.4). Lastly, I turn to the process that I followed to test the capabilities and limits of my robot (3.5).

3.1 Simplified Mode of Operation

In this section, I outline the methodical procedure by which the robot's functional capabilities are initiated. Upon pressing the "*" key on the membrane switch module, the computer vision model is downloaded. The robot is then designed to start its quest once the "1" key is pressed. In a first step, the robot is programmed to take pictures that are then analyzed by my computer vision model in search of objects. In this thesis an object is considered to be an instance of the computer vision model's class "lying person." In the event that an object—i.e., someone lying on the floor—is identified, external help is alerted through the dispatch of an email requesting help. Attached to this email are pictures with the object marked accompanied by a confidence level indicating the likelihood that the image actually contains said object. In the case that no object has been identified, the Raspberry Pi then establishes a connection with the Arduino, which is responsible for controlling two ultrasound sensors. These sensors measure the distances and, subsequently, the robot's direction of movement is determined. To minimize the probability of driving in the wrong direction, the two distances are measured three times before the robot moves. The cycle of taking pictures and moving then begins anew.

The overview flowchart illustrates the simplified process described above:

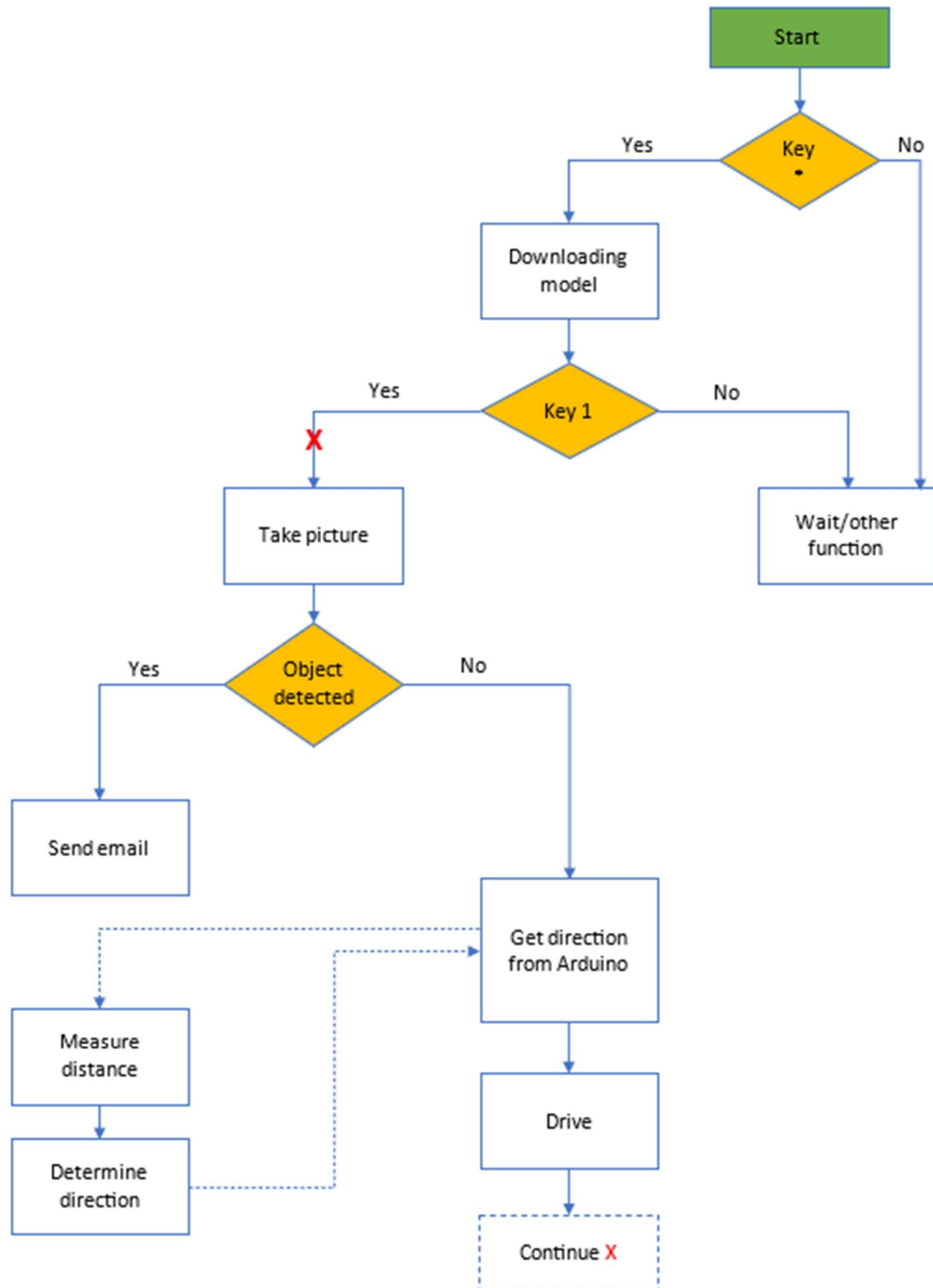


Figure 4: Simplified overview flowchart—decision-based order of fundamental functionalities

3.2 Functionalities

This section is divided into three main subsections, detailing the robot's three core functionalities: autonomous driving, object detection, and external alerting.

3.2.1 Autonomous Driving

This section is divided into two main subsections, namely driving and autonomous navigation in an indoor environment.

Driving

The main hardware components utilized for autonomous driving are a Raspberry Pi 4+ and an Arduino Uno. In this context, the Raspberry Pi 4+ acts as the processor, which is essentially a small but powerful computer capable of running an operating system and managing complex tasks. The Arduino Uno, on the other hand, acts as the microcontroller, which is more suited to handling real-time sensor data and simple, repetitive tasks. The Raspberry Pi 4+ controls the direction and speed of four motors, while the Arduino Uno operates two ultrasound sensors, performs measurements, and calculates distances, thereby deciding on the direction of movement, which it reports to the Raspberry Pi. In this context, I introduced the concept of primary and subordinate devices. As the primary device, the Raspberry Pi is responsible for controlling all the various components and coordinating the execution of their different functions. Conversely, the Arduino, as a subordinate device, performs specific predefined functions and then reports back to the primary device.

When assembling the hardware, I connected the motors on each side in a parallel circuit. Due to the positioning of the fixing screws, I had to mount the motors facing in opposing directions. Therefore, it was essential to cross-wire the motors on each side to ensure that they turned in the same direction, rather than in opposing directions.

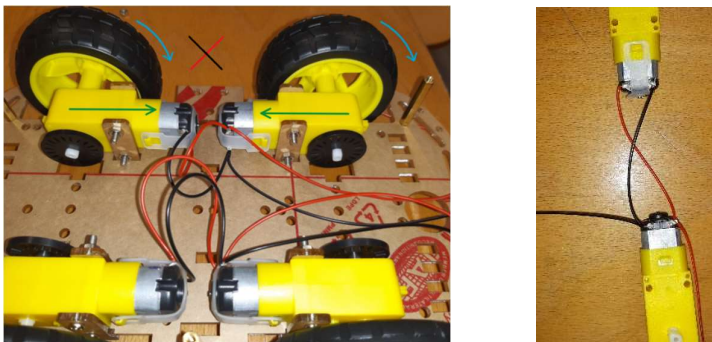


Figure 5: Wiring of the motors (left) with cross-wiring (right)

Furthermore, I soldered a small circuit in order to connect a 100mF 25-volt capacitor in parallel to the motors. This is essential, as the motors operate based on the constant reversion of polarity, which can lead to electromagnetic interferences. Moreover, the motors tend to draw large amounts of current from the Raspberry Pi, which could cause constant rebooting [7].

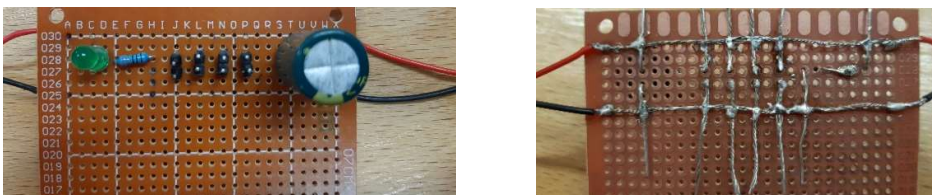


Figure 6: Capacitor circuit (top view left, bottom view right)

The circuit was built and the program threw no exceptions—but nothing moved. I therefore tested the motors with an Arduino where I knew from previous projects that my code worked. The motors did not turn there either, suggesting a hardware issue. I then tested whether current was flowing in the circuit, and discovered it was not. I then noticed that the H-bridge⁴ was damaged. After replacing the H-bridge and a few cables, the motors finally turned.

After this problem had been resolved, I realized that my right wheels would only turn forwards but not backwards and assumed the problem was related to the wiring of the motors. Motors have three pins: one to enable the motor, i.e., to make it turn (ENA), one for driving it forwards (A), and one for driving it backwards (B). These pins can be set to either “LOW” (no current) or “HIGH” (current). Unfortunately, it was not clear which pin was which.

In total there are 2^3 combinations of setting the pins to HIGH or LOW but instead of trying all eight I produced a table where I logically excluded all combinations that were impossible, e.g. HIGH, HIGH, HIGH, as the motors cannot turn forwards and backwards at the same time. Furthermore, I knew that the enabling pin (ENA) had to be set to HIGH as it enables the motor to turn. As the motors did turn forwards, I could deduce that pin B (column 3) was as intended the one responsible for turning backwards as it was the only one set to LOW. Furthermore, I then knew that one of the two pins set to HIGH (in column 1; row 1, row 2) was the enabling pin. But when I tested the motors to turn backwards, they did not move indicating to me that I had misassigned the enabling pin. Therefore, I tested A being the new ENA and ENA being the new A which then made the motors successfully turn backwards.

Table 1 exemplifies how I systematically approached the problem:

	Original assignment	Test 1	Test 2	Test 3	New assignment
1	ENA	HIGH	HIGH	LOW	A
2	A	HIGH	LOW	HIGH	ENA
3	B	LOW	HIGH	HIGH	B
	Desired behavior	forwards	backwards	test	
	Actual behavior	forwards	no move	backwards	

Table 1: Overview of wiring options for motors, as part of an error analysis.

Autonomous navigation

As a relatively simple method for autonomously navigating an apartment, the robot’s primary guideline for movement is to follow walls. This process necessitates two distance measurements:

Upon activation, the Arduino requires the first ultrasound sensor to measure the distance to the nearest object in front of the robot. After a predefined delay, the second ultrasound sensor measures the distance to the closest thing on the right, i.e., the wall. Ultrasound sensors measure the distance by registering the time between emitting an ultrasonic signal and receiving its reflection;⁵

$$distance(cm) = \frac{time(\mu s)}{2} * 0.03432 \frac{cm}{\mu s} \quad (3)$$

⁴ An H-bridge circuit enables voltage polarity reversal and is utilized here to facilitate forward and backward operation of DC motors.

⁵ The measured interval is the time that the sound takes to traverse the distance from the robot to the object and back. Therefore, the time has to be divided by 2 and then multiplied with the speed of sound to calculate the distance.

After calculating the distances in two directions, the Arduino differentiates among four possible scenarios, as illustrated in Figure 7, and returns one of the corresponding values.

If something is closer than 15 cm on the right and in front, the robot assumes it has hit a cul-de-sac and reverses. This scenario is depicted in Figure 7 by a red hatched area.

If something is closer than 40 cm on the right and something else is closer than 50cm in front, the robot foresees a potential cul-de-sac and turns left. This situation is illustrated in Figure 7 by a light blue hatched area.

If something is closer than 40 cm on the right and there is nothing in front for at least 50cm, the robot advances. This is represented in Figure 7 by a green hatched area.

If there is nothing on the right for more than 40 cm, the robot recognizes that the wall turns a corner and turns right. This scenario is shown in Figure 7 as a dark blue hatched area.

The sketch in Figure 7 illustrates the four cases:

Ultrasound – Decision

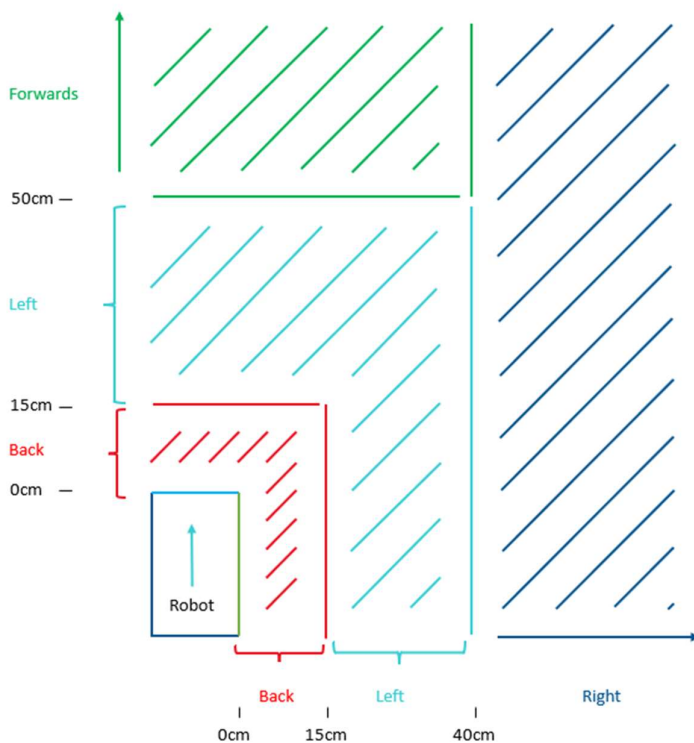


Figure 7: Illustration of movements depending on distance measurements in two directions

In Figure 8, an illustrative example of a living room and a kitchen is presented, demonstrating a typical layout. The diagram uses solid lines to represent the walls, which define the boundaries and structure of the rooms. Dotted lines indicate doors, which is important because closed doors restrict the search area the robot can cover. By recognizing walls and doors, as shown in this figure, the robot can follow the walls continuously, allowing for a systematic and thorough exploration of the apartment. Figure 8

3.2.2 Autonomous Object Detection

To detect a person lying on the floor, I used the computer vision library OpenCV, mentioned in Chapter 2. Initially, I familiarized myself with this library by using pre-developed models that encompassed facial recognition, position estimation, and hand detection. These models aided my understanding of OpenCV's functionality and how the models are typically implemented in code. However, they were not directly applicable to my specific problem of detecting people on the floor.

My original plan was to utilize position estimation to define certain cases where external help should be alerted. As it seemed quite complex to adapt the source code, I decided to train my own model using data that I generated myself. For training my object detection model, I used the neural network Yolo-v8, also mentioned in chapter 2. Furthermore, I chose object detection—which also localizes the object—over image classification, which solely determines the presence of an object within an image. To ensure the robot could handle a variety of potential scenarios, I maximized the diversity of my image dataset, using footage of eight individuals representing different ages and genders, who assumed various positions: lying on their back, front, or side with their arms and legs positioned at various angles. Furthermore, I took the pictures from various distances and angles in different locations. This led to a database of 3325 pictures. I then used the Roboflow platform to label the pictures by outlining all people lying or sitting on the ground. I marked everything else as a null image,⁷ which indicates to the neural network that there are no objects to be framed within bounding boxes in those images. Finally, I utilized Roboflow to train my model with Yolo-v8, achieving a precision of 98.8%, which I deemed more than satisfactory for my purposes. I had originally planned to apply the model on a live stream, to detect objects in real time, which one usually does with Yolo-v8 as it is an anchor-free neural network (also mentioned in chapter 2). Therefore, I programmed my code to access a public website that allowed users to plug in user credentials of the desired model and of the visual data to be processed. However, this turned out impossible for me, as a firewall denied my code to enter a link to my livestream on the website. I circumvented this problem by downloading the model on to my Raspberry Pi instead of using the public link. I then applied the model locally—without a livestream link—on three images per cycle of collecting visual data and driving. The model then successfully indicated whether an object was found and how certain it was that this actually was an object and saved the picture with the corresponding bounding box.

The most significant problem I had was how to circumvent issues concerning the availability of up-to-date software. I encountered many problems when trying to get the camera to work properly, which was essential for my project. I had originally installed the “Bullseye” operating system, as this was the newest version at that time. However, after trying to “fix” my code when the camera did not work, I discovered that the camera function was “deprecated” for this operating system, which essentially meant that the camera could not open due to a bug in the operating system. I then proceeded to download a previous version of the operating system entitled “Buster.” There the camera worked, but the problem I had was the installation of the OpenCV library (section 2.2.3), which was necessary for image recognition. Either the connection broke off or dependencies had not been installed, meaning that I needed to download other software packages first. After connecting the Raspberry Pi directly to the ethernet and finding the best version for my operating system, I could download and access the OpenCV library. Once the other functionalities had been programmed, I wanted to test my robot and run the computer vision model on the Raspberry Pi. When I ran my program, however, an error message indicated that my operating system was too old to run the Yolo-v8 model. Luckily, a new

⁷ More precisely, I initially planned to distinguish lying from sitting people. However, due to the comparatively small amount of data (125 out of 3325 pictures), image recognition quality was low and I abandoned the distinction between different non-fall poses.

operating system called “Bookworm” was released at that time. Having learned that Bookworm limits modifications to the operating system, I then downloaded OpenCV in a virtual environment, imported my programs, and ran the main file, which finally worked.

3.2.3 External Alerting

To alert external help when a person lying on the floor was detected, I had initially programmed the Raspberry Pi to take full control of my phone and let it make a call. However, this method was prone to mistakes as the mouse and keyboard automatically fulfilled their clicking and typing tasks in spite of the fact that the screen was sometimes too slow to load. As a consequence, the commands were sometimes executed in a very untimely manner, leading to incorrect operations. Therefore, I decided that the robot would alert external help via an email (see standard alert text in Figure 9). The email consists of a text body requiring immediate assistance and includes the pictures of the person in need, with the corresponding bounding box around the object (as illustrated in Figures 10 and 11). Moreover, the level of confidence regarding whether what was detected is actually a fallen person is also indicated (Figure 12).

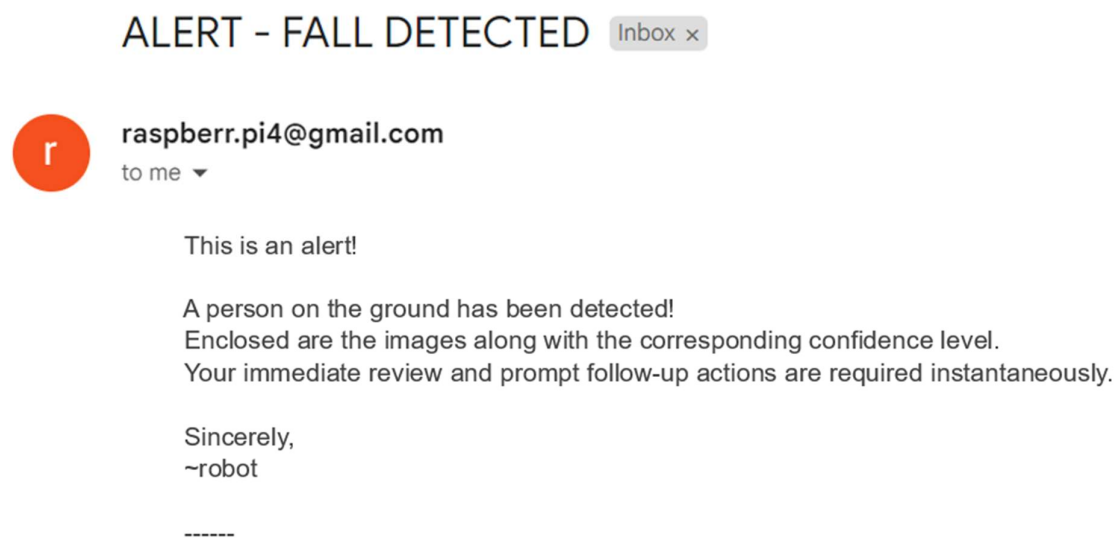


Figure 9: Default alert email

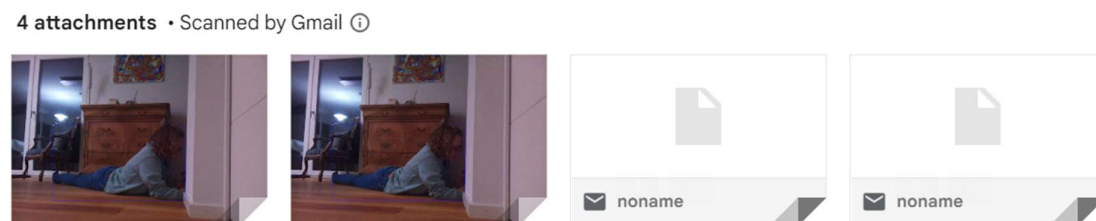


Figure 10: Attachments of two pictures in which an object was detected, in addition to one file per image that is automatically created and cannot be opened



Figure 11: Detail of Figure 10 showing the predicted bounding box (thin blue line) delineating the detected person on the floor

The model returns the properties of the bounding box of the images containing an object. The confidence is of particular interest.

```

Image 0:
  No objects detected.

Image 1:
  Objects detected!
  {
    "x": 1376,
    "y": 1387,
    "width": 1696,
    "height": 966,
    "confidence": 0.8709307312965393,
    "class": "fallen",
    "class_id": 0,
    "image_path": "orig1.jpg",
    "prediction_type": "ObjectDetectionModel"
  }

Image 2:
  Objects detected!
  {
    "x": 1376,
    "y": 1382,
    "width": 1696,
    "height": 970,
    "confidence": 0.8904001712799072,
    "class": "fallen",
    "class_id": 0,
    "image_path": "orig2.jpg",
    "prediction_type": "ObjectDetectionModel"
  }

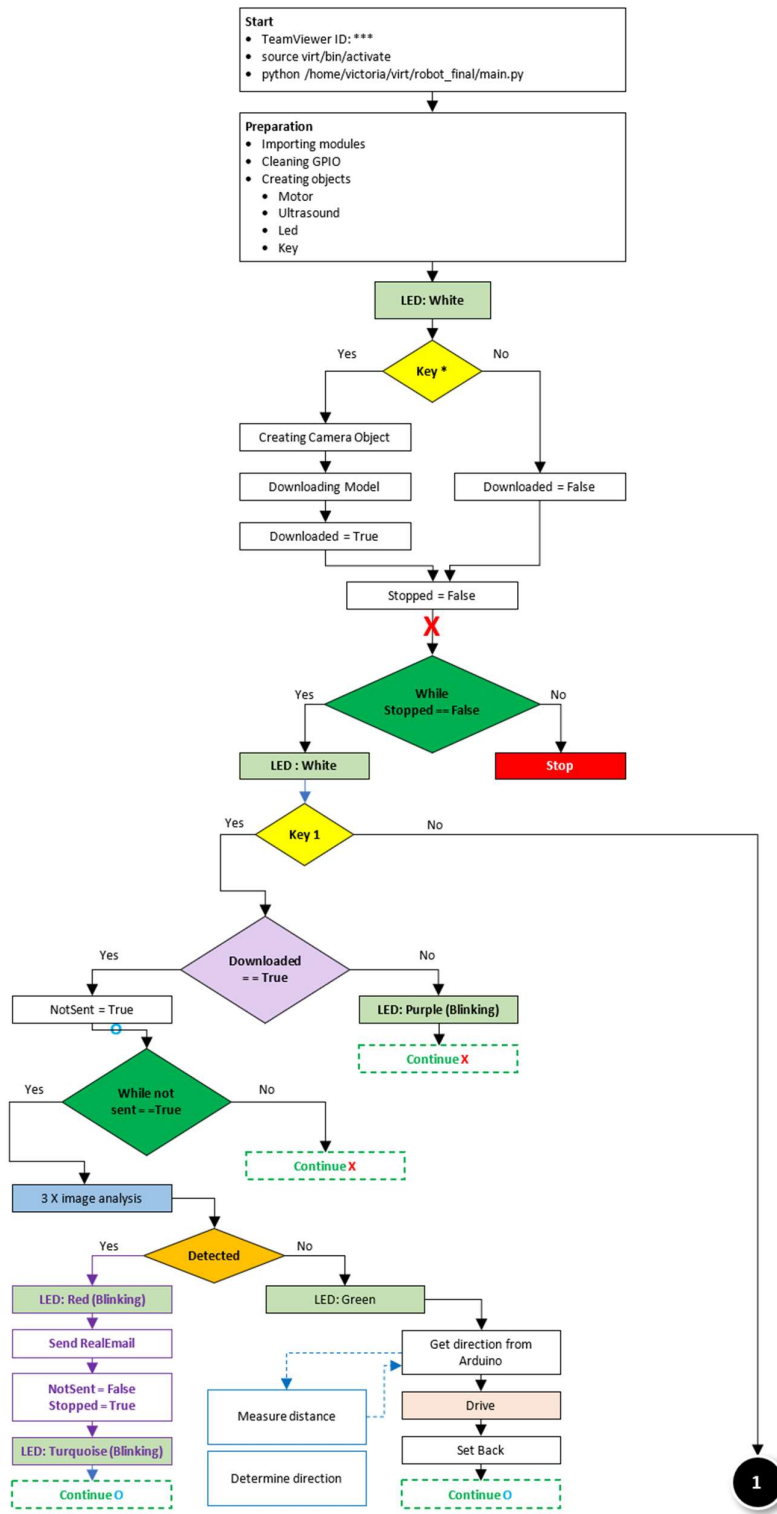
```

Figure 12: Example confidence level of having detected an object in the image

On very rare occasions, the model fails to detect a person lying on the floor, even though one is present. Therefore, I decided that three pictures should be taken before the robot would restart the cycle of analyzing images and driving. In Figure 10 above it incorrectly failed to identify any lying person (False Negative case) in one out of three images.

3.3 Detailed Mode of Operation

The flowchart below indicates the full functionalities of the robot as opposed to the simplified version in chapter 3.1.



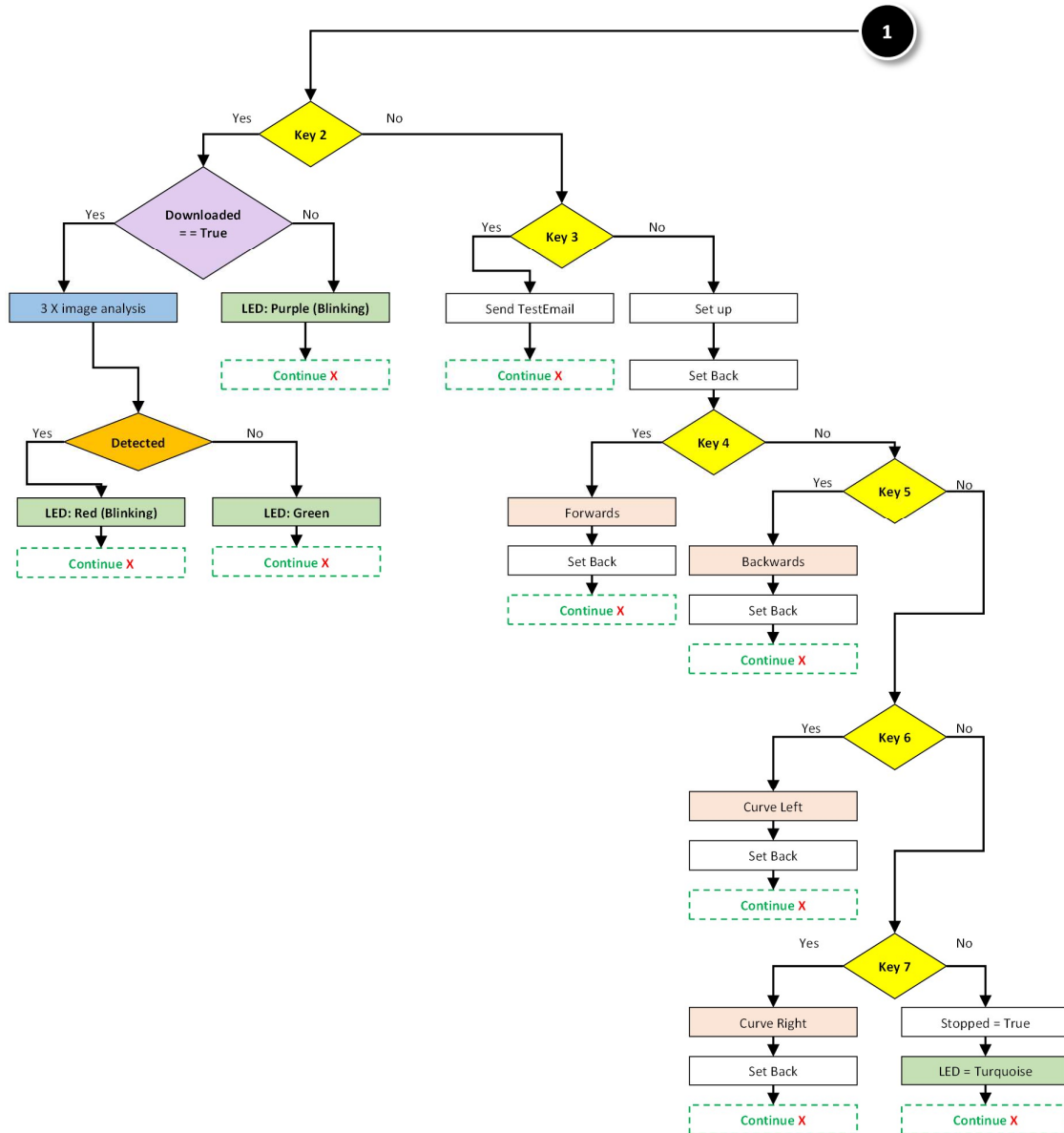


Figure 13: Detailed flowchart with functionalities of the main components

Currently, the robot is activated by running the file “main.py.” Therefore, one must first log in to the Raspberry Pi (with ID and password not specified here) and take over control via TeamViewer. Once logged in, the virtual environment is accessed via console with the command “source virt/bin/activate” where “virt” is the virtual environment. The main file is then executed with the command “python /home/victoria/virt/robot_final/main.py”. Instead of a simple on/off button, I chose a membrane switch module to activate the robot. This choice facilitates program expansion and the integration of additional functionalities at a later stage. The Raspberry Pi remains in standby mode until a key is pressed on the membrane switch module. To download the computer vision model, “*” must be pressed as the first key, otherwise no image recognition can be performed.

Upon pressing the “1” key, the robot starts its quest of locating a person lying on the floor. Following this initiation, the robot operates completely autonomously, first taking pictures and then analyzing them.

- If an object has been detected in at least one image, the image(s) of the folder “recognized” with the corresponding confidence level(s) is attached to an email. The robot then stops.
- On the other hand, if no object is detected, the raspberry commands the Arduino to determine the direction via ultrasound sound sensors and proceeds to start driving.

The remaining keys are designed to facilitate robot testing. They are outlined below:

- If key “2” is pressed, pictures are taken, analyzed, and stored at a location where previous pictures are overwritten if the model has been downloaded before (key “*”). Regardless of whether pictures could be taken or not, the robot pauses and awaits the next key instruction.
- If key “3” is pressed, the three pictures in the folders “recognized” and “not recognized” are attached to an email along with the corresponding confidence levels, and the email is then sent to the “test emergency contact.”
- If key “4” is pressed, the robot drives forward.
- If key “5” is pressed, the robot reverses.
- If key “6” is pressed, the robot follows a forward left-hand curve.
- If key “7” is pressed, the robot follows a forward right-hand curve.
- If any of the other nine keys is pressed, the program stops.

3.4 Software and Hardware Configuration

3.4.1 Software Architecture

I organized my code into small units to facilitate testing and debugging. More precisely, I structured my code into different classes. A class is a blueprint version of an object, which is also called an “instance.” A class has variables (so-called “attributes”) and functions (so-called “methods”) for each object. For example, the attributes of an LED would be its pin numbers, whereas its method could be to glow red. If several LEDs are to be operated, writing the same methods and variables for all of them would make the code hard to follow and highly susceptible to errors. Instead, classes prevent code duplication by allowing the creation of instances for each LED where only certain parameters differ between the objects, such as the GPIO pins of the Raspberry Pi that the LEDs are connected to.

Instances are created with:

```
instanceName = className(parameters)    led1 = Led()
```

Methods of an instance are called:

```
instanceName.methodName(parameters)    led1.red(3)
```

The diagram below is a UML class chart that illustrates the different classes I created, along with their attributes (including their types) and methods. The underlined variables are class variables which are used by every instance while all others are instance variables that can differ from object to object. The “-” signs specify the attribute or method as private, whereas the “+” signs specify that they are public and can be used by any other class.

Class⁸

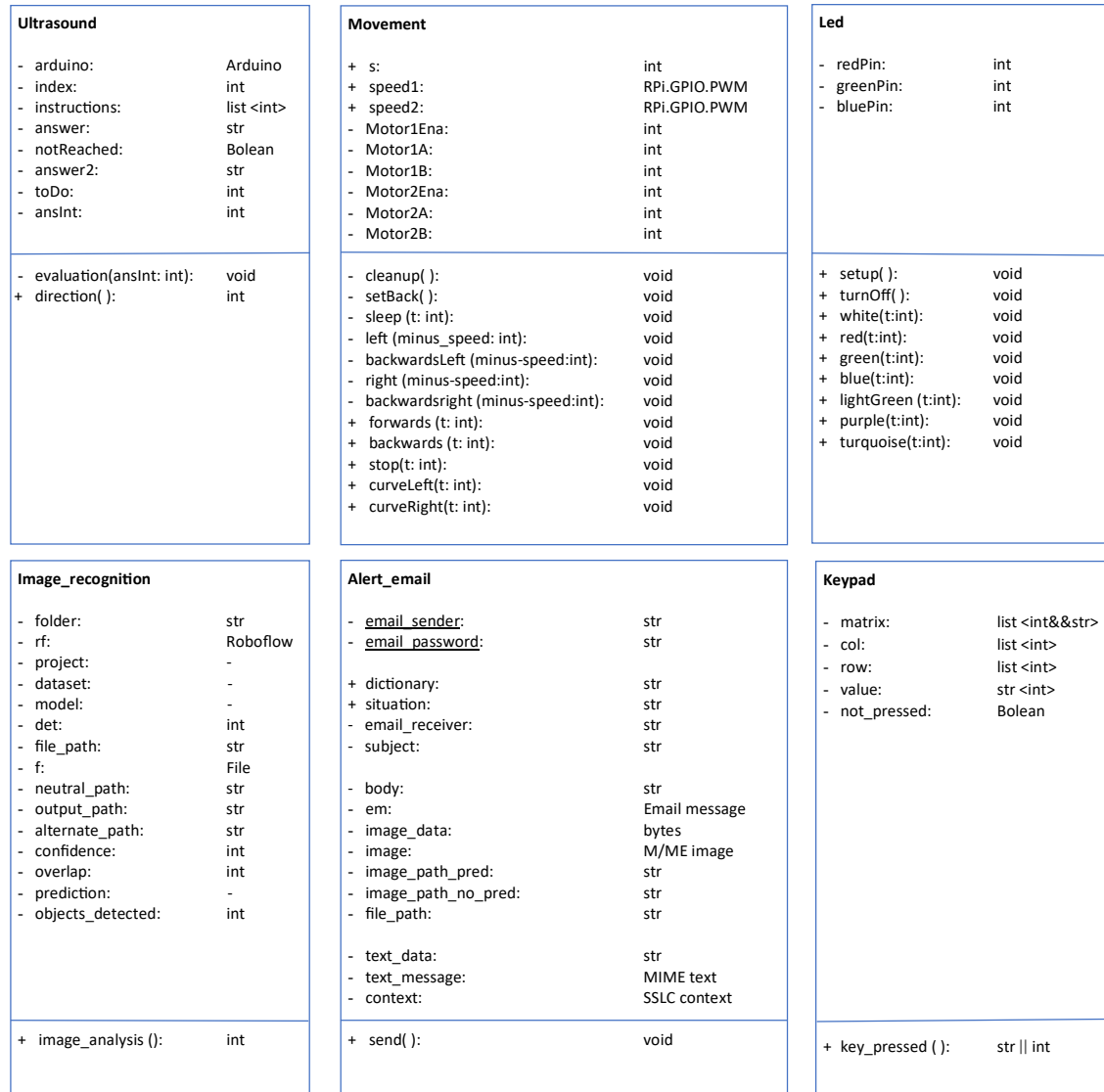


Figure 14A: Overview of UML classes, including attributes, types, and methods

⁸ There are no associations indicated as they are only called by the file "main.py" and do not directly depend on each other.

The type "-" signifies in this UML chart that in order to find the real type, more time would have to be spent to analyze the source code of Roboflow models.

Due to some initial confusion over the fact that in Python attributes and methods cannot be truly private, as in languages such as Java, not all attributes have been consistently implemented as being private yet.

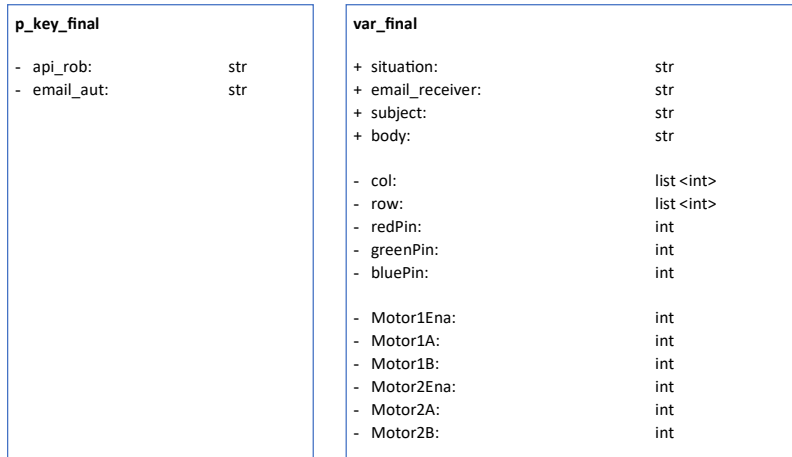


Figure 14B (continuation): Overview of UML classes, including attributes, types, and methods

- The class "Ultrasound" establishes the connection to the Arduino and interprets the direction of movement.
- The class "Movement" controls the direction and speed of the motors.
- The class "Led" controls the LED's color and duration of shining.
- The class "Image_recognition" takes and saves images, analyzes them with the computer vision model and determines the number of objects.
- The class "Alert_email" attaches images to a warning message and sends the email.
- The class "Keypad" determines which key is pressed on the membrane switch module.
- The dictionary "p_key_final" stores passwords.
- The dictionary "var_final" stores variables such as plain text and pin numbers.

3.4.2 Hardware Architecture

The sketch below shows the different hardware components, with colors indicating what they are connected to. The same colors on different hardware components signify that there is a cable connection.

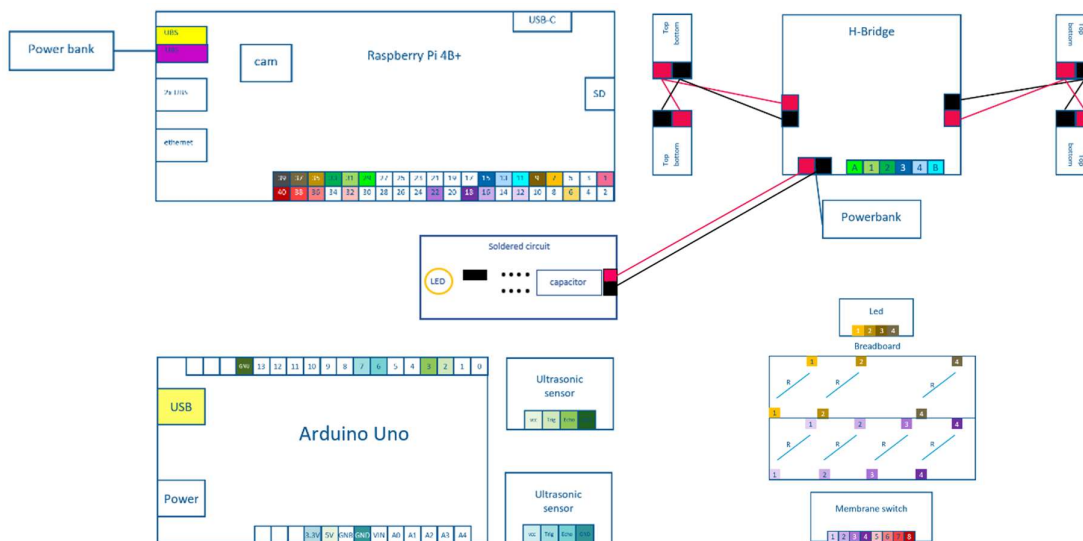


Figure 15: Wiring of hardware components

The hardware components are:

- A Raspberry Pi 4B+, which is the primary device that has overall control. It is directly connected to the Arduino Uno, the H-bridge, the breadboard, the membrane switch, the LED, and the power bank.
- An Arduino Uno, which is the subordinate device that determines the direction by controlling ultrasound sensors and reports back to the Raspberry Pi.
- Two ultrasound sensors, which measure distances by emitting and receiving ultrasonic waves.
- Four DC motors, which are responsible for setting the robot in motion.
- An H-bridge, which connects the motors.
- A soldered circuit with capacitor, which is connected to the H-bridge and serves as temporary power storage.
- A breadboard on which hardware components are connected and resistors can be built in.
- A membrane switch module in the form of a keypad, which allows users to execute tasks manually.
- An LED, which indicates running processes.
- Two power banks, which serve as a power supply.

All these hardware components, when assembled together, result in a small, robust robot that will hopefully soon be used in households to take care of the elderly.

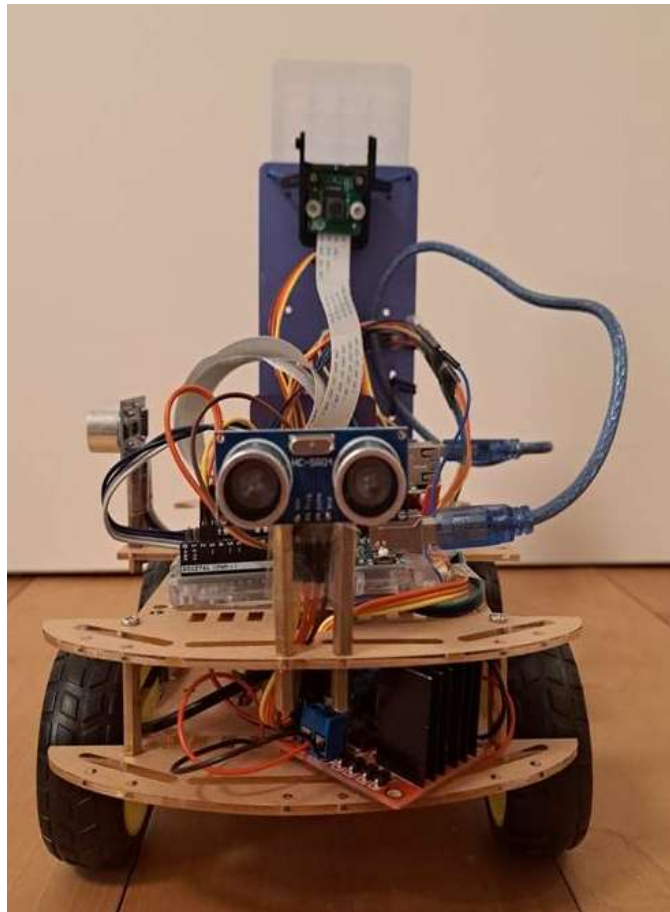


Figure 16: My robot

3.5 Empirical Testing

This section elaborates on how I tested the quality of my robot’s assessments of people in different situations. I was thereby able to determine when the robot operated flawlessly, pinpoint its constraints, and understand the elements upon which its success or failure hinges. These tests mainly related to the question of whether images retrieved by the robot were accurately matched to how it was trained. Figures 17 and 18 exemplify the principles of this matching before I detail my testing procedure:

When creating the computer vision model, I marked the outline of the person lying (as shown in “How trained” in Figure 17). During testing, the model would therefore annotate images as “fallen” and enclose the object in a blue bounding box (denoted as “True Positive” in Figure 17). However, there were instances where the presence of a lying person was not correctly registered by the model (as shown in “False Negative” in Figure 17). Minimizing these False Negatives is vital, as they represent situations when an alert should be executed but is not.

Images		
How trained:	True Positive - where it correctly detected:	False Negative - where it should have detected, but did not:
		

Figure 17: Visual representation of training examples (“How Trained”) compared to actual assessments of the robot in which an alert should be sent (“True Positive” and “False Negative”)

However, the robot should not alert anybody if no accident has occurred (Figure 18). Therefore, I had trained the model with “null images” where I did not annotate anything because nobody was lying on the floor (“how trained”). If the robot did not identify any object during testing because no person was lying on the floor, this was assessed as a True Negative. Sometimes, however, it did identify an object (surrounded by the blue bounding box) even though no target was lying on the floor (False Positive).




Images		
How trained:	True Negative— where it correctly did not identify:	False Positive— where it detected but should not have (false alarm):
		

Figure 18: Visual representation of training examples (“How trained”) compared to actual assessments of the robot in which no alert should be sent (“True Negative” and “False Positive”)

In order to systematically test the quality of my robot in its evaluation of situations in which people had fallen to the floor, I proceeded with the following steps:

1. Preparing the evaluation
2. Defining criteria for assessing the robot’s performance
3. Taking test pictures and having them assessed by the robot
4. Coding, i.e., assessing, the pictures according to different criteria and conducting random checks
5. Systematic data cleaning to ensure high quality of the sample
6. Data analysis according to pre-selected quality criteria

First, I developed a list of questions that I wanted to address with my robot and that are detailed in the results chapter. For example, I assumed that the robot would recognize people equally well independent of their actual position on the ground. To be able to test these assumptions, I developed a list of parameters (e.g. position) that would need to be represented in my evaluation and assigned attributes (e.g. lying) to each one. Taken together, these parameters led to coding guidelines for later usage that included:

- Correctness of the robot’s evaluation (True Positive, True Negative etc.)
- The position of the person (lying vs. sitting, etc.)
- Characteristics of the person (gender, adult vs. child)
- Presence of the person in the training data (yes/no)
- The visibility of the person’s extremities (how many legs visible, etc.)
- The perspective of the robot (front vs. back, etc.)

In addition, I defined a number of non-human test objects that I could use to test the robot’s recognition capabilities—for example, objects such as stuffed animals, a dog, various blankets and cushions, and so on.

Second, I defined the criteria by which I planned to analyze the robot’s strengths and its limits. I first defined the situations that I would judge as true or false and as positive or negative with the following classification, based on the literature presented in chapter 2:

True Positive (TP): A person lying on the floor has correctly been identified, i.e., the alarm is executed because there is a person in need.

True Negative (TN): The model has correctly predicted that there is no person lying on the floor, i.e., no alert is executed because no accident has happened.

False Positive (FP): The model predicts that there is a person lying on the floor even though there is none, i.e., there is a “false alarm.”

False Negative (FN): There is a person lying on the floor who should have been identified, i.e., there is no alert even though the person is in need.

I then referred to three values: accuracy, precision, and confidence (see also chapter 2).

Accuracy is defined as the ratio of all correct assessments relative to all assessments made. In other words, accuracy describes how often the robot correctly identifies a situation, whether a person is actually lying on the floor or not.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+F} \quad (4)$$

Precision is defined as the ratio of all correctly identified, positive situations relative to all situations with a positive evaluation. In other words, precision describes how often an alarm that was issued was actually correct (and conversely, *1-precision* provides the percentage of “false alarms”).

$$Precision = \frac{TP}{TP+FP} \quad (5)$$

The confidence measure is a value between 0 and 1 provided directly by the robot and signifies how certain the underlying model is that the identified object is indeed an object that should be recognized. Any confidence under 40% is defined to be “no-object,” and therefore nothing is returned from the model. Confidence levels above 40% are determined to be an “object,” and therefore the model returns an image containing a bounding box around the object and returns values such as the confidence (values between 0.4 and 1), the location of the bounding box, etc.

Third, I invited 12 people for a session of picture-taking, making sure that the following two criteria were fulfilled: I sought a heterogeneous group encompassing a spectrum of different ages and genders. Additionally, I ensured that my sample of testers included some with whom the model had been trained and some who were unknown in the database. During this session I took 716 pictures. In selecting the picture settings, I paid attention to different positions within the room and different lighting conditions. In addition, I included “empty” pictures in which only furniture was visible, but no object that should be recognized by the robot, I also included several “experimental” settings that I used in order to identify the limits of my robot, i.e., to find out in which situations it would perform better and in which worse. After taking a number of pictures, I immediately checked their quality in order to ensure that the procedure worked. Out of 716 photo shots, only two had to be excluded as the robot actually failed to take a picture, suggesting an excellent failure rate of only 0.28%.

Fourth, I coded the remaining 714 pictures along the parameters and attributes and manually assigned values to each of the parameters—most importantly, whether the robot had correctly identified fallen people. As illustrated in Figure 19, this led to a database of 716 entries, of which 590 were pictures of a single individual, 97 of objects such as stuffed animals, 18 of furniture (= pictures without special

items), and 9 were experimental pictures in which I changed the vertical angle of the robot. After coding about 20 pictures, I realized that my original coding guidelines were incomplete with respect to the concrete position that a person sitting on the floor could assume. Particularly, I ventured that the position of the legs (flat on the ground vs. bent) might influence the recognition capabilities of the robot. Therefore, I specified the missing details in the coding guidelines and started coding from scratch. After having coded all pictures, I conducted random tests with about 5% of all pictures to minimize any mistakes. For example, a typical random check involved the identification of the perspective of the robot without looking at the code I had originally assigned. By and large, the original coding seemed appropriate, although it would have been beneficial to involve a second person to further minimize mistakes: sometimes, making code assignments was a question of judgment—for example, whether or not a black-socked foot was indeed visible in low lighting. In such cases of doubt, a discussion with a second coder would have been helpful to increase reliability.

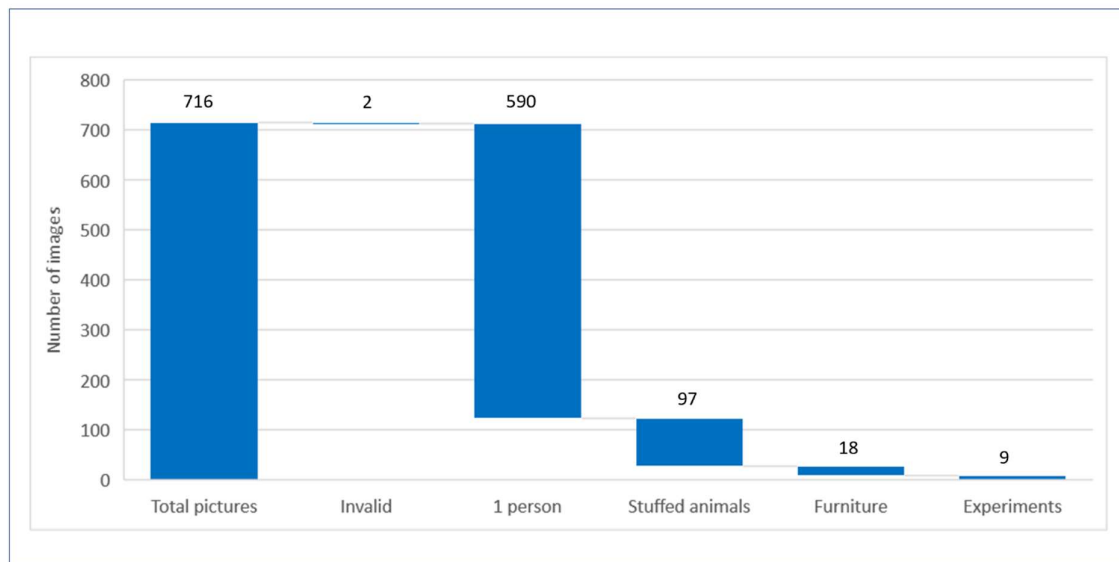


Figure 19: Number of test images per category

Fifth, I engaged in systematic “data cleaning” to ensure that the database was of sufficient quality for analysis with Excel. This particularly related to removing typographic errors in my codes such as “fornt” instead of “front” or subtler problems with unifying near-duplicates such as “knee” and “knees” or “back” vs. “back ” where the extra blank character at the end would make automated pivot analysis difficult.

Sixth, in the final step, I analyzed the data. To do so, I went through the data to check for plausibility and get a first systematic impression—for example, whether the number of correct identifications of people lying on the ground was credible, or whether confidence levels were in a reasonable range. I then analyzed the data in Excel by applying combinations of different filters, only to realize that this procedure was too time-consuming and error-prone—i.e., there were too many different variables to manually evaluate at the same time. Instead, I learned how to apply so-called pivot analyses that automatically apply filters to the database. While still complex, this procedure allowed me to systematically investigate the performance of the robot, as laid out in the following chapter.

4 Results

To examine in which scenarios my robot is able to detect a fall and consequently alert external help, I conducted tests for two different cases: the one-person-containing case (section 4.1) and the person-free case (4.2).

4.1 One-Person Case

My first series of tests contained exactly one person. This is the most important case, as I assume the robot will be used by elderly people who are living alone.

Position of the Person

My main hypothesis was that the person's position does not influence the quality of determination of whether they have fallen to the floor or not. This assumption was based on the fact that I conducted an extensive training with 3325 pictures. I investigated this assumption by comparing the accuracy of correctly identified "objects" and "no-objects" of a person in different positions, as shown in Figure 20.

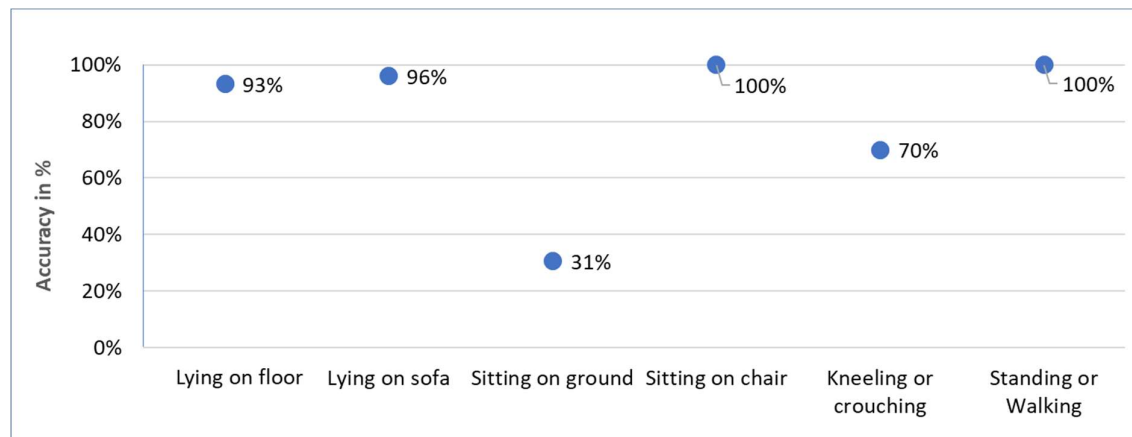


Figure 20: Accuracy of assessment depending on the position of the person (basis: 590 images)

The accuracy of a person "lying on the floor" as well as "standing or walking," "sitting on chair," and "lying on sofa" are all well above 90%, which I deemed extraordinarily high.

Specifically, I find it quite remarkable that the accuracy of the category "lying on sofa" (True Negative) is 96%, particularly since I did not train the model on people "lying on sofa." However, I did train the model on not identifying a sofa or a person sitting on a sofa as an object, which makes the high accuracy somewhat understandable.

At the same time, my testing dataset for the category "kneeling or crouching" is not extensive, which makes the more modest accuracy of 70% more understandable. Yet, it is important to recognize these positions effectively, because they are "dynamic" (in the sense that a person could be in the process of standing up or lying down) and as such merit more attention. One way to deal with these positions could be to examine the series of three sequential images and then compare differences. More straightforwardly, one could also include them more prominently in the training data set of a future model.

Compared to the other values, the accuracy of the "sitting on floor" category of 31% is very low. In other words, the robot incorrectly identified the seated person as lying down (False Positive) in two-

thirds of the cases. The low values are not surprising for two reasons: First, a seated position resembles a lying position much more closely than, for example, a standing position, because of the angle of the upper body. While a distinction based on small differences is still possible, if a low value of importance has been assigned to the angle of inclination during training, it is clear that the robot will have problems classifying those similar images. This problem is illustrated in Figure 21. Second, and more importantly, the training set of this category is much too small, with 125 images out of 3325 in total, and would need to be expanded in an updated version of the model.

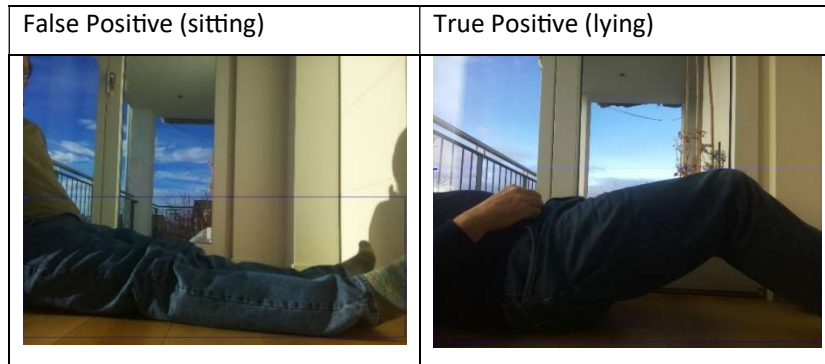


Figure 21: Illustration of the difference between a sitting position with both legs down (left: False Positive) and a lying position with both legs up (right: True Positive).

Different Sitting Positions

Due to the low accuracy when people are sitting on the floor, it was important to know which “sitting position” is hardest to identify. Based on the illustration in Figure 22, my assumption was that the more a sitting position resembles a lying position, the harder it will be for the robot to make a correct assessment. I investigated this by comparing the accuracy of correctly *not* identifying people in different sitting positions.

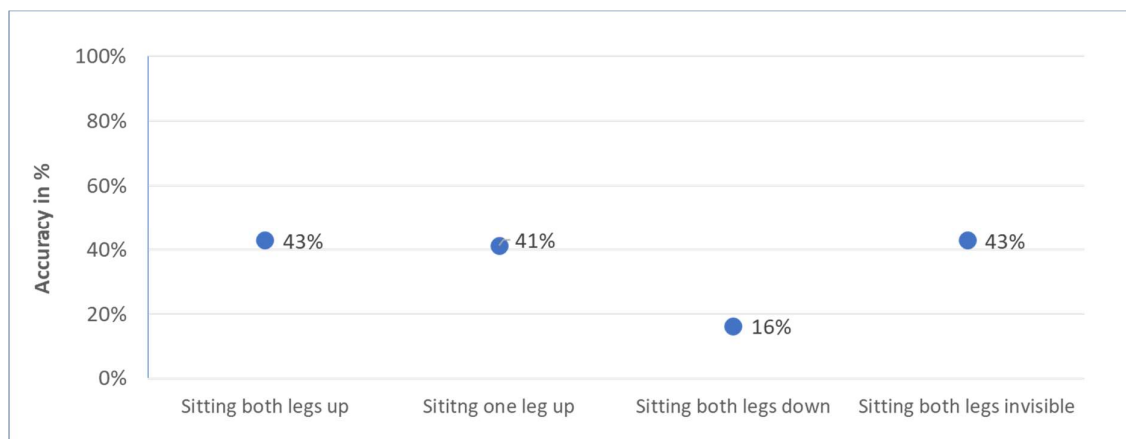


Figure 22: Accuracy of determination in different positions of people sitting on the floor (basis: 95 images)

The cases of people sitting with “both legs up,” “one leg up,” or “both legs invisible” are equally accurate at around 40%. The accuracy of sitting with both legs down, however, is below 20%, as the resemblance to someone lying on their back is very strong. Further training of the underlying model would have to take these findings into account.

Different Lying Positions

Due to the high importance of the category “lying on floor,” I also analyzed the accuracy of people lying on the floor in different positions, similar to the analysis of sitting people. Based on my learnings so far, I assumed that the position of a person lying on the floor would not influence the quality of determination. This assumption was fueled by my thinking that the ability of the robot to correctly assess a situation would be based on the shape of the object rather than on the angle of any physical features such as a face or knee. I investigated this question with a comparison of the accuracy of correctly identifying “objects” in different lying positions.

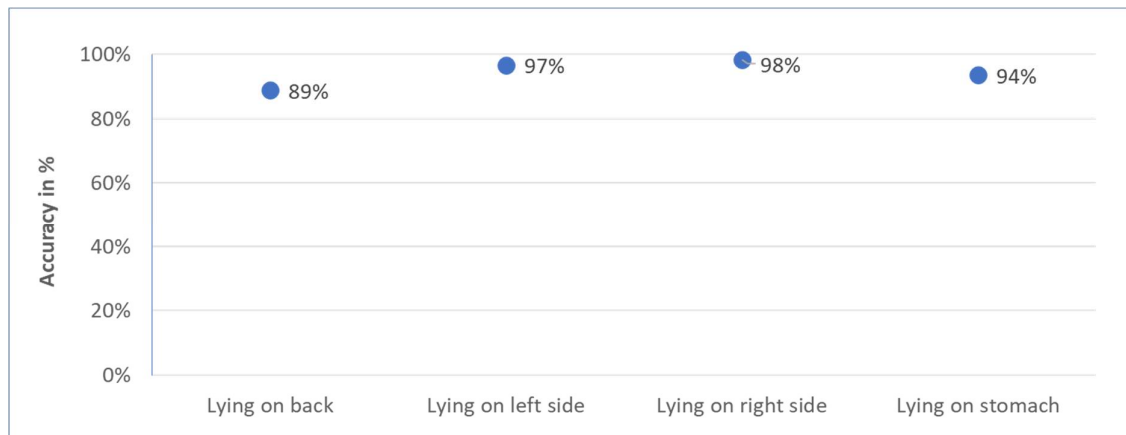


Figure 23: Accuracy of determination in different positions of people lying on the floor (basis: 373 images)

Accuracy in all cases is very high: around 90% or above. However, the accuracy of identifying people lying on either side was even higher, which prompted the question of whether seeing two distinct legs is of high importance.

Visibility of Extremities

Following my previous analysis, I assume that the visibility of extremities influences the quality of determination regarding whether a fall has occurred. I explored this question based on the accuracy of correctly distinguishing between “objects” and “no-objects” depending on the extremities that can be seen⁹.

⁹ If something is placed in front of a lying person (e.g., a table leg), the model recognizes objects on either side of the item. In a further analysis the confidence level for the different extremities can thereby be analyzed.

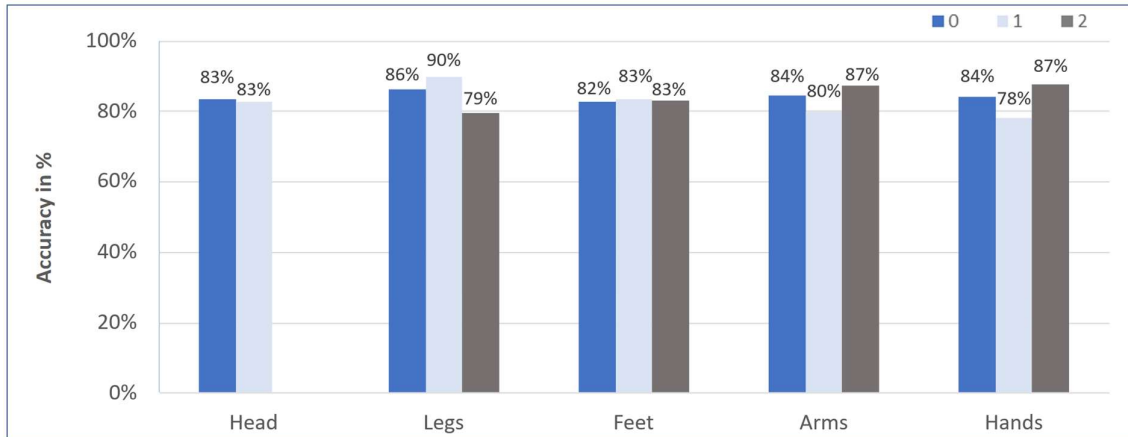


Figure 24: Accuracy of determination depending on number of visible extremities (basis: 590 images)

The number of visible feet and whether the head is visible or not have virtually no influence on accuracy. Accuracy depending on the number of visible hands and arms is also almost the same. Yet, astonishingly, the accuracy is around 5% worse when one arm or hand is visible than when no arm or hand is visible at all.

Even more interestingly, seeing two legs decreases the accuracy by around 10% in comparison to when one or no legs are visible. This is because the ratio of sitting to not-sitting people in the training data is greater with two legs than with one or zero. As the number of False Positives is larger with the number of sitting people than with not-sitting people, this might have had a major influence on accuracy. An alternative explanation could lie in the coding procedure, which was based on only one person, and even though care was taken to make appropriate coding judgments, this could be a source of bias.

To further explore this topic, I also investigated the precision of correctly distinguishing between “objects” and “no-objects” depending on the extremities that can be seen (Figure 25).

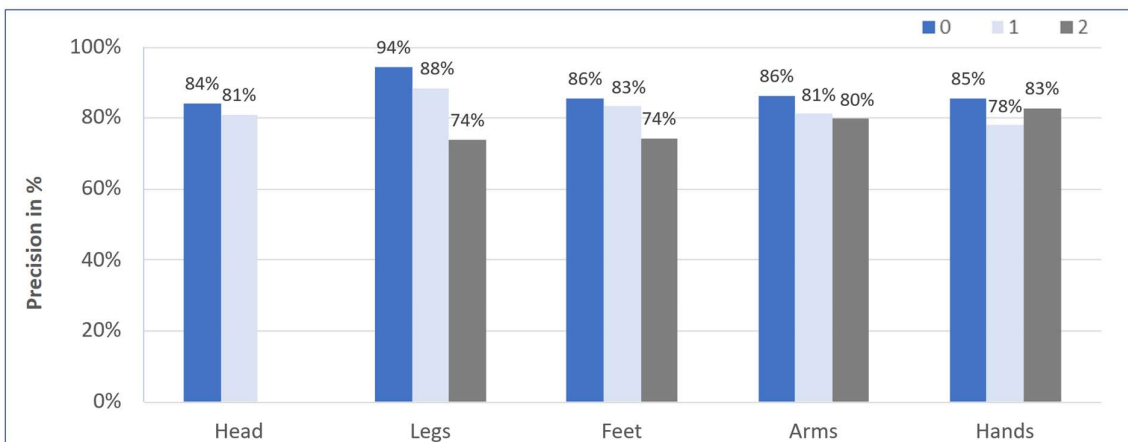


Figure 25: Precision of determination depending on number of visible extremities (basis: 590 images)

Precision based on the visible extremities is high, at around 80% overall. However, surprisingly, false alarms are generally executed more often when more extremities can be seen, which provides another indication that this should be analyzed in more detail in a further step of the project with improved training and coding.

Age and Gender

I also analyzed several characteristics specific to individual people. My assumption was that the gender and age (adult vs. child) of a person should not generally influence the quality of determination of whether a fall has occurred. However, I am aware that the distribution of genders in my training and testing data could affect outcomes. Again, I investigated this by measuring the accuracy of correctly distinguishing between “objects” and “no-objects” based on the type of person.

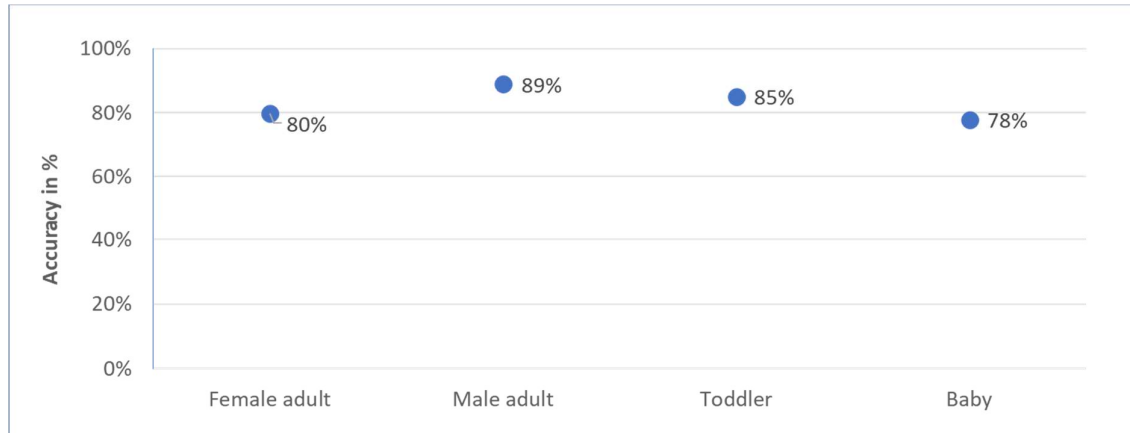


Figure 26: Accuracy of determination depending on type of person (basis: 590 images)

Interestingly, accuracy is higher for male adults than for female adults. I trained the computer vision model with two men and six women (ratio 1:3) and I deduced tests with four men and six women (2:3), which led me to believe that the model would correctly identify women more often.

The accuracy for toddlers and babies is high, in spite of the fact that I trained without those “types of people.” However, the values have less significance, as I only had the opportunity to test those categories with one toddler and one baby. Furthermore, a future model should not identify babies as fallen, as they frequently lie on the floor and it is no cause for concern.

Influence of Knowing or Not Knowing the Person

Based on my extensive training data, I assumed that whether a person has been used for training or not would not influence the quality of determination of whether a fall has occurred. I explored this assumption by measuring the accuracy of correctly distinguishing between “objects” and “no-objects” based on whether the model had been trained with this person.

	Accuracy	Precision
Person not known	84%	85%
Person known	79%	70%

Table 2: Accuracy of determination depending on whether the person is known from training or not (basis: 590 images)

Both accuracy and precision are very high for people the model has not trained with. This signifies that the training base was large enough for it not to influence the results.

Nevertheless, it is at first surprising that the accuracy of images with “known-people” is lower than that for “not-known-people.” I assume this counterintuitive effect occurs because I could only test the robot with two people who had served as training subjects, whereas I could employ 10 who were not known to the model. Therefore a few False Negative and False Positive assessments of the people known to the model could have had a disproportionate influence.

Position of the Robot

As not only the position of the person can vary but also the angle from which the robot approaches and hence the perspective the robot adopts, I also analyzed this case.

Specifically, I assumed that the position of the robot does not influence the quality of determination of whether a fall has occurred. I tested this assumption by measuring the accuracy of correctly distinguishing between “objects” and “no-objects” based on different positions of the robot, as shown in Figure 27.

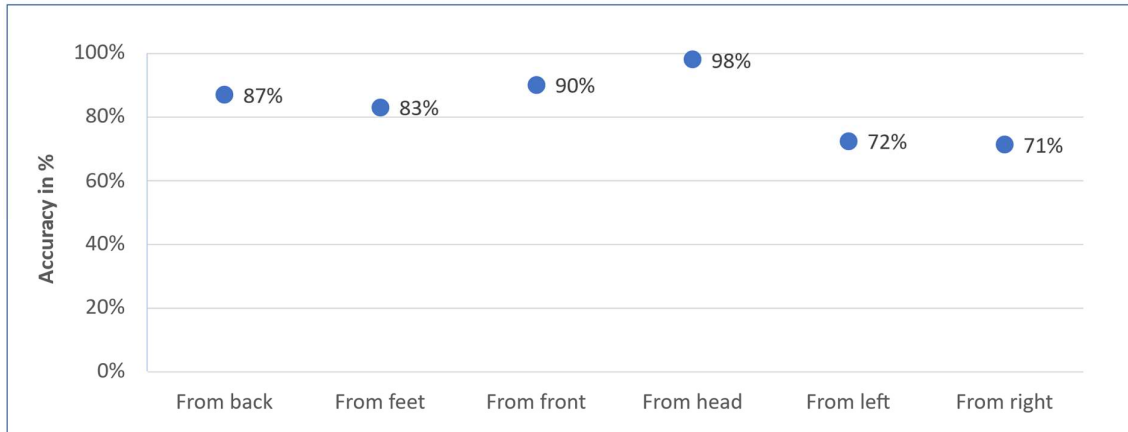


Figure 27: Accuracy of determination depending on the position of the robot (basis: 590 images)

All six positions of the robot are based on about 100 pictures each and exhibit a high accuracy of over 70%. Yet, it is interesting to note that the highest accuracy occurs when the robot approaches from the direction of the person’s head. I assume this effect occurs because the robot can only approach from the head when a person is indeed lying on the floor. As the accuracy of correctly detecting a lying person is very high, the accuracy of the category “from head” is also very high. Hence, the accuracy depending on the robot’s position is also influenced by the position of the person. Conversely, I assume that the somewhat lower accuracy of pictures taken “from left” or “from right” is influenced by the fact that these perspectives are typical for people sitting on the floor, which are harder for the robot to predict. This is corroborated by a simultaneous analysis of the position of the person and the robot (not shown in Figure 27).

4.2 Person-Free Case

My second series of tests relates to pictures that contained no person, which is also a likely situation if the user lives alone.

No Person Shown

My basic assumption was that no object would be detected if no person was shown in the image. I investigated this by measuring the accuracy of correctly not identifying “other-/no-objects.” Figure 28 shows that—to my surprise—the accuracy varies widely.

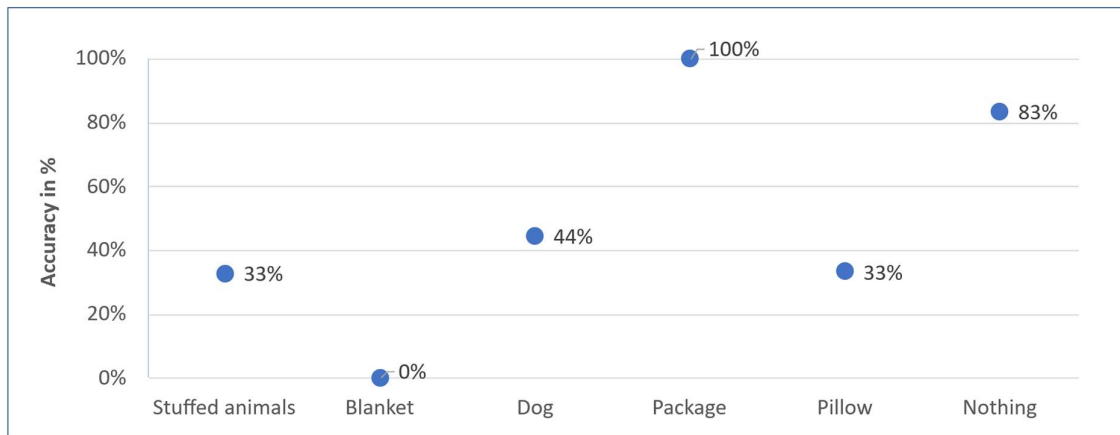


Figure 28: Accuracy of determination if no person is present (basis: 115 images)

For example, the package was determined with an accuracy of 100%, meaning it was always correctly identified as no person. Moreover, an “empty” space, i.e. one containing only furniture, was correctly recognized with an accuracy of 83%. In contrast, however, the blanket was always incorrectly identified as a lying person. At first sight, this is surprising. However, one has to bear in mind that the computer vision model was trained for shapes. Figure 29 illustrates this: the image on the left shows how I outlined the “lying person” and the image on the right shows what shape was saved. It is easy to assume that the blanket can take a shape similar to the outline in the right picture. Correspondingly, the robot falsely identified the blanket as a lying person as the model was trained in a tidy environment.

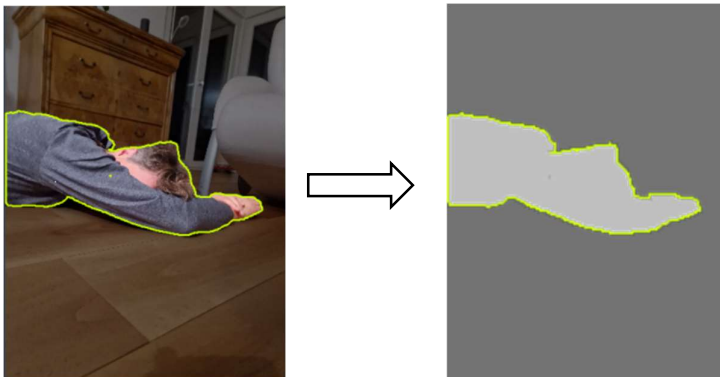


Figure 29: Illustration of training data: lying person (left) and abstract shape of the lying person (right)

Stuffed Animals

Given the importance of shape in the assessment, I further analysed the role shape with stuffed toy animals.

My assumption was that human-like soft toys would be more likely be recognized as a lying person than non-human-like ones. The corresponding accuracy measurements reveal large differences (Figure 30). For example, the dolphin (100% accuracy) and the owl (75%) were mostly correctly identified as a non-person, while the seal was always incorrectly identified as a person. However, these

three categories suffer from too few testing samples because, unfortunately, Roboflow limits the number of times a machine learning model can be run within a specific time frame. I tested the bear the most extensively, with 30 images, as it seemed to bear the closest resemblance to a human baby. Yet, the bear was incorrectly identified more often than not as a “lying person” (27% accuracy). Surprisingly, however, the dog—which ostensibly bears less resemblance to a human—was equally often incorrectly identified as a lying person (29%).

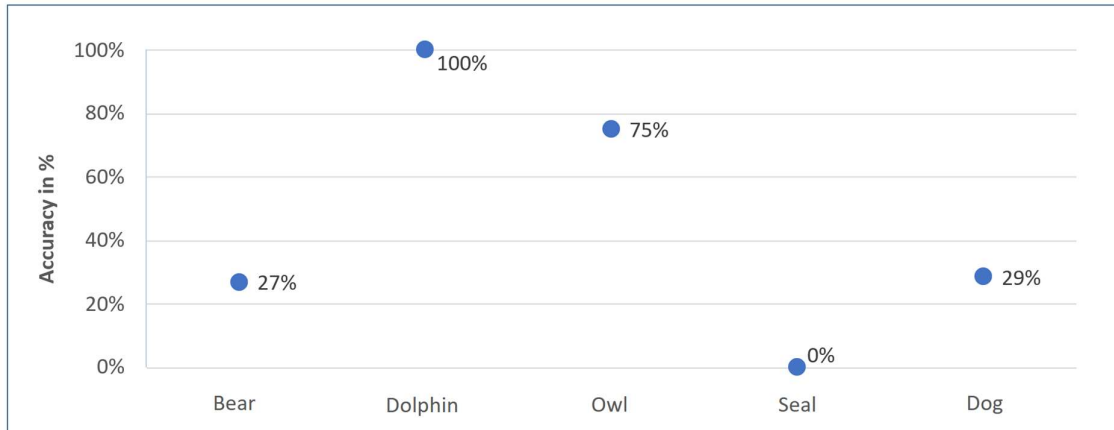


Figure 30: Accuracy of determination if no person is present (basis: 46 images)

5 Discussion

The newly built robot described in this thesis was able to navigate autonomously, assess situations, and issue alert messages in case a person had fallen. The results in chapter 4 provide a basis to evaluate the functionality of the robot and suggest areas of improvement. In my discussion, I cover three broad areas: object detection, navigation, and functional extensions.

Object detection

Overall, I was astonished by the capabilities of the computer vision model to recognize image content. This is evident from the high accuracy rates, but also became clear while coding the testing images: in many dark and backlit pictures, it proved difficult for human eyes to discern the picture content, but the robot still achieved correct assessments with high confidence scores.

With respect to accuracy, most importantly, the robot was capable of correctly identifying nine out of 10 people lying on the floor. This accuracy was practically independent of a range of influence factors, such as the person's position or whether their extremities were visible. While an accuracy of 100% would obviously help to convince potential users of the robot's advantages, the current version can be understood as a first prototype, and it is plausible that the accuracy will increase with improved training. At the same time, the robot made mistakes in more than half of the situations while assessing situations with people sitting on the floor and in distinguishing stuffed animals and other objects from real people. This is an area that needs significant improvement, but given the small number of training pictures that were used, major improvements should be possible.

A related critique that could be voiced is that my training in the main categories "lying" and "non-lying" bears many simplifications in comparison to real-life situations. This can easily be improved with a larger database for training. Specifically, I would re-train my model in the same categories "lying-adult," "ground-sitting-adult," and "null." However, I would employ a training set that is equally distributed between those categories. Furthermore, the "null" category should contain not only people who are "walking" and "chair-sitting," but also "toddlers & babies" and "inanimate objects." This seems relevant as "toddlers & babies" lying on the floor is an everyday event. I would also choose to train the "null" category with more "inanimate objects," as I trained the current model in an overly tidy environment, which reduced accuracy during testing. Also, "dynamic positions" such as getting up could be covered in new training data with adequate coding.

Another question concerns the coding of the test images. It is currently a limitation in the data evaluation that certain images can be interpreted differently—does half a hand qualify as a hand? Is the head influential in general, or is it facial features that are important for the categorization? Therefore, it would have been useful to have somebody else evaluate all the data as well to increase reliability of the assessments, meaning that potentially ambiguous data could have been compared.

Navigation

Importantly, the accuracy of the robot's assessments is independent of its perspective, which means that the way the robot navigates is, in principle, sufficient for the task at hand. One might question whether people might trip over the robot, but this seems unlikely as the device does not roam around freely, interfering with daily activities, but rather always follows a wall. Nevertheless, other ways to facilitate navigation should be considered.

Specifically, the robot is currently limited in its autonomous navigation by the fact that an open stairwell would cause it to fall. Similarly, while the robot can avoid obstacles, it is limited to those with a rectangular footprint, as it is currently programmed to make only 90-degree turns. In other words, chairs must not be placed against the wall. An alternative means of navigating the apartment would

be for the camera images to be processed in real time to plot the unit's future path. This would make direct use of the new Yolo-v8 functionality of real-time processing, with which the environment could be mapped. Another possibility would be to follow the target by employing triangulation, which would avoid a searching process. However, these options were beyond the scope of this thesis.

Functional Extensions

A particularly useful functional extension would be direct phone control and natural language processing so that a person who has fallen can directly send messages via the robot. To implement this, I have developed a program that can call desired contacts autonomously—however, this still needs to be refined due to occasional lagging between the robot and the phone. Furthermore, I have programmed a voice recognition model which I will soon implement into my code. It can distinguish between sentences such as “Please help, I am hurt” and “I am fine” and will be instrumental to determine whether to only send an email warning that a person might have fallen or to execute an emergency phone call. Moreover, it would be helpful if the robot could have a more extensive interaction with external help, i.e., interpreting their email response and sending close-ups of certain angles if required.

In order to prepare the robot for use on a commercial level, it would be good to use adaptive “incremental learning” where more training data become available over time and the robot continues learning, as opposed to “traditional learning” where the computer vision model has completed its training in advance. Furthermore, the robot could then learn that it is normal for people to lie on the floor in certain areas—for example, while exercising—whereas in others it is a cause for alarm.

6 Conclusion

In this thesis, I have presented a new robot that I have designed, built, programmed, and tested that is capable of autonomously navigating a home, identifying whether a person has fallen to the floor, and sending an alert email in case of an emergency.

When comparing my robot to other devices for fall detection as presented in the introduction, several advantages can be listed: My automated robot remains fully functional even when a person is unconscious, which does not apply for traditional emergency alert buttons. Furthermore, my robot provides concrete information on the user's state of well-being by sending images that can be analyzed by the recipient. This does not affect the user's privacy, as pictures are only saved when they need assistance. As my robot autonomously navigates around an apartment, there are no blind spots provided all internal doors remain open. Disadvantages mainly relate to the accuracy of the assessments that the robot currently makes, particularly in distinguishing between people sitting and lying down. However, the overall performance of my computer vision model suggests that this should be easily surmountable with more extensive training.

To conclude, my matura thesis provided me with an extremely enriching experience in which I was able to combine designing, building, and programming a robot and conducting an empirical analysis of its performance. Not only did I manually assemble hardware components and learn how to solder electric circuits, but I also deepened my knowledge in object-oriented programming, designing my software, generalizing the code such that functionalities could be easily modified, and logically breaking down and tracing errors. Furthermore, I learned how to conduct extensive empirical tests and how to evaluate large amounts of data. As well as reiterating that it was an extremely fun experience, I am especially glad to say that my robot might play a useful role in households where elderly people would otherwise have to move into retirement homes. I hope that my autonomous robot will improve their quality of life by helping them continue to live independently.

7 Bibliography

7.1 Traditional Sources

1. De Haes, Bert Antoon; Pelgrims Roeland; Verrept, Stijn (2020): An Elderly Care and Security System: EP 3 723 456 A1
2. Florczyk Stefan (2005): Robot Vision, Video-based Indoor Exploration with Autonomous and Mobile Robots, Introduction to Chapter 3, and Chapters 3.2 to 3.4, Wiley VCH Verlag GmbH & Co, Weinheim.
3. Hussain, Muhammad (2023): YOLO-v1 to YOLO-v8, the Rise of YOLO and Its Complementary Nature toward Digital Manufacturing and Industrial Defect Detection, *Machines* 11 (7), page 677.
4. Kumar Suresh B.; Raju Viswanadha S.; Maheswari Uma V. (2023): OpenCV libraries for computer vision: Applications of Visual AI and Image Processing, edited by Pancham Shukla, Rajanikanth Aluvalu, Shilpa Gite and Uma Maheswari, Berlin, Boston: De Gruyter, 2023, pages 1-22.
5. Mar-Hernandez, Pedro Guillermo; Ibarra-Angulo, Pedro Luis; Grijalva- Acuna, Juan Carlos; Abril-Garcia; (2023) *Journal Computer Technology* Dec. Vol. 7, No. 19 pages 1-9
6. Richeng, Cheng (2020): A survey: Comparison between Convolutional Neural Networks and YOLO in image identification, *J. Phys.: Conf. Ser.* 1453 012139.
7. Röbenack, Klaus (2020): *Mobiler Eigenbauroboter mit Arduino, Aufbau und Programmierung 3., überarbeitete und erweiterte Auflage*, page 15.
8. Roberts, James (2011): Enabling Collective Operation of Indoor Flying Robots, Chapters 1.2.3 and 1.2.4., University Thesis, Ecole Polytechnique Fédérale de Lausanne.
9. Rojas Castro, Dalia Marcela (2017): The RHIZOME architecture: a hybrid neurobehavioral control architecture for autonomous vision-based indoor robot navigation, Chapter 2, University Thesis, Université de La Rochelle
10. Selcuk, Burcu; Serif, Tacha (2023): A Comparison of YOLOv5 and YOLOv8 in the Context of Mobile UI Detection In: Younas, M., Awan, I., Grønli, TM. (eds) *Mobile Web and Intelligent Information Systems. MobiWIS 2023. Lecture Notes in Computer Science*, vol 13977. Springer, Cham.
11. Shin, Dongeek; Patel, Shwetak; Chaudhry, Rizwan et al. (2021): Radar-Based Monitoring of a Fall by a Person: WO 2021/118570 A1
12. Takuma, Sumiya; Yutaka, Matsubara; Miyuki, Nakano (2015): A Mobile Robot for Fall Detection for Elderly-Care, *Procedia Computer Science* 60, 870 –888
13. Veretennikov, Stanislav; Baskakov, Vladimir; Maltsev, Anton (2021): System and Method for Smart Monitoring of Human Behavior and Anomaly Detection: WO 2021/202274 A1
14. Xiu, Zhang; Xin, Zhang; Wei, Wang (2023): *Intelligent Information Processing with Matlab*, Springer Verlag.

7.2 Online Sources

101. GlobalData (2021), Switzerland Population Distribution in 2021, by Age, accessed on 12/08/2023 [https://www.globaldata.com/data-insights/macroeconomic/switzerland-population-distribution-in-by-age/#:~:text=The%20population%20of%20Switzerland%20reached,1.66%20million\)%%20of%20the%20population.](https://www.globaldata.com/data-insights/macroeconomic/switzerland-population-distribution-in-by-age/#:~:text=The%20population%20of%20Switzerland%20reached,1.66%20million)%%20of%20the%20population.)
102. Shabrozshahab, Muhammad (2023): Yolo v8, www.medium.com, accessed on 12/08/2023, <https://medium.com/@muhammadshabrozshahab/yolo-v8-104f1375242c>
103. Khandelwal, Renu (2022): Supervised, Unsupervised and Reinforcement Learning, www.medium.com, accessed on 12/08/2023, <https://arshren.medium.com/supervised-unsupervised-and-reinforcement-learning-245b59709f68>
104. Malviya, Nikita (2023): Object Detection Anchor Box VS Bounding Box, www.medium.com, accessed on 12/08/2023, <https://medium.com/@nikitamalviya/object-detection-anchor-box-vs-bounding-box-bf1261f98f12>
105. The Robotics Back-End, Tutorial, Raspberry Pi Arduino Serial Communication – Everything You Need To Know, <https://roboticsbackend.com>, accessed on 19/11/2023, <https://roboticsbackend.com/raspberry-pi-arduino-serial-communication/>
106. MacDonald, Gaven (2013): Membrane Matrix Keypad with the Raspberry Pi, accessed on 19/11/2023 <https://www.youtube.com/watch?v=yYnX5QodqQ4>
107. Roboflow Enterprise (2023), Roboflow Docs, Raspberry Pi, accessed on 19/11/2023: <https://docs.roboflow.com/deploy/raspberry-pi>
108. The PyCoach (2023), How to Send Emails with Python [New Method 2023], accessed on 19/11/2023 https://www.youtube.com/watch?v=g_i6ILT-X0k
109. Distrelec (2023), Primary device: Raspberry Pi 4B+, accessed on 06/01/2024: https://media.distrelec.com/Web/WebShopImages/landscape_large/87/fa/raspberry-pi-b-plus-30001887fa.jpg
110. Makercreativo (N/A): Arduino Uno R3, accessed on 06/01/2024: <https://www.makercreativo.com/store/producto/arduino-uno-r3/>
111. CamDo (N/A): SanDisk Extreme Micro SD card, accessed on 06/01/2024: <https://camdo.com/products/sandisk-extreme-micro-sd-card-128gb-with-sd-adapter>
112. Robotistan (N/A): Raspberry Pi Camera Modul V2, accessed on 06/01/2024: <https://www.robotistan.com/raspberry-pi-camera-modul-camera-modul-for-raspberry-pi>
113. Random Nerd Tutorials (N/A): Arduino Membrane Keypad, accessed on 06/01/2024: <https://randomnerdtutorials.com/arduino-membrane-keypad-tutorial/>
114. Digitec (N/A): Breadboard, accessed on 06/01/2024: <https://www.digitec.ch/en/s1/product/dfrobot-breadboard-breadboard-half-size-development-boards-kits-32988244>
115. Amazon (N/A): VBOTCOR DC 3V-12V TT Dual Shaft Gear Motor, accessed on 06/01/2024: <https://www.amazon.com/VBOTCOR-3V-12V-DUAL-SHAFT-MOTOR/DP/B09MG3B12S?th=1>
116. Amazon (N/A): Dual h-bridge Motor Drive Controller for Arduino Smart Car Robot Power Motor, accessed on 06/01/2024: <https://www.amazon.com/H-Bridge-Controller-Arduino-Stepper-Driver/dp/B06W5MNTSF>
117. Farnell (N/A): Rubycon 25PK100MEFC10X16, Aluminum Electrolytic Capacitor, 1000uf, 25v, 20%, radial, accessed on 06/01/2024: <https://ch.farnell.com/en-CH/rubycon/25pk100mefc10x16/aluminum-electrolytic-capacitor/dp/2749339>

118. Robocraze (N/A): What is Ultrasonic Sensor: Working Principle & Applications, accessed on 06/01/2024: <https://robocraze.com/blogs/post/what-is-ultrasonic-sensor>
119. Sparkfun (N/A): Diffused LED - RGB 10mm, accessed on 06/01/2024: <https://www.sparkfun.com/products/11120>
120. Reichelt (N/A): Metal oxide resistor, accessed on 06/01/2024: https://cdn-reichelt.de/bilder/web/artikel_ws/B400/!WID1W.jpg
121. AliExpress (N/A): Veroboard, accessed on 06/01/2024: <https://de.aliexpress.com/item/32321476411.html?gatewayAdapt=glo2deu>
122. Reichelt (N/A): Robot chassis kit for all ARDUINO systems, accessed on 06/01/2024: https://cdn-reichelt.de/bilder/web/artikel_ws/A300/ROBOT03-02.jpg
123. Reichelt (N/A): Raspberry Pi 30x30x10-mm fan for NESPi housing, accessed on 06/01/2024: https://cdn-reichelt.de/bilder/web/artikel_ws/A300/RPI_FAN_30X30_NEU_01.jpg
124. Digitec (N/A): USB cables, accessed on 06/01/2024: <https://www.digitec.ch/en/s1/product/ugreen-usb-ausb-a-usb-cables-21185719>
125. Digitec (N/A): USB-C cables, accessed on 06/01/2024: <https://www.digitec.ch/de/search?q=usb+c>
126. Techtonics (N/A): USB Cable For Arduino UNO/MEGA (USB A to B) - 0.3m, accessed on 06/01/2024: <https://www.techtonics.in/usb-cable-for-arduino-uno-mega-usb-a-to-b>
127. Grandado (N/A): cable M-F/M-M/F-F Jumper Breadboard, accessed on 06/01/2024: <https://images.nexusapp.co/assets/e8/d1/f7/307223219.jpg>
128. Digitec (N/A): Powerbank, accessed on 06/01/2024: <https://www.digitec.ch/en/s1/product/aukey-pb-xd26-26800-mah-45-w-9650-wh-powerbanks-8991738>
129. Galaxus (N/A): Screws, nuts and washers, accessed on 06/01/2024: <https://www.galaxus.ch/en/s4/product/agt-assortment-box-hexagon-bolts-nuts-washers-hex-drivers-14458880>

8 Appendix

8.1 Code

Structure of the folders and files:

On the raspberry:

```
/
├── home/
│   ├── victoria/
│   │   ├── virt/
│   │   │   ├── robot_final/
│   │   │   │   ├── alerting_final/
│   │   │   │   │   ├── documents_email/
│   │   │   │   │   │   ├── not_recognized/
│   │   │   │   │   │   │   ├── no_prediction1.jpg
│   │   │   │   │   │   │   └── no_prediction2.jpg
│   │   │   │   │   │   └── recognized/
│   │   │   │   │   │       ├── prediction3.jpg
│   │   │   │   │   │       └── textfile_recognition/
│   │   │   │   │   │           └── prediction_precision.txt
│   │   │   │   │   ├── __pycache__/
│   │   │   │   │   └── alert_email_final.py
│   │   │   │   ├── camera_final/
│   │   │   │   │   ├── __pycache__/
│   │   │   │   │   └── image_recognition_final.py
│   │   │   │   ├── control_final/
│   │   │   │   │   ├── __pycache__/
│   │   │   │   │   └── keypad_final.py
│   │   │   │   ├── indication_final/
│   │   │   │   │   ├── __pycache__/
│   │   │   │   │   └── led_final.py
│   │   │   │   ├── motion_final/
│   │   │   │   │   ├── __pycache__/
│   │   │   │   │   ├── movement_final.py
│   │   │   │   │   └── ultrasound_final.py
│   │   │   │   ├── variables_final/
│   │   │   │   │   ├── __pycache__/
│   │   │   │   │   ├── p_key_final.py
│   │   │   │   │   └── var_final.py
│   │   │   │   └── main.py
```

On the Arduino:

```
ultrasound_2_direction/
├── ultrasound_2_direction.ino
```

8.1.1 main.py

```

#!/usr/bin/env python3
# description:
#   This is the main file.
#   It controls the followin actions:
#   - registering which key is pressed on the 'membrane swich module lpc'
#   - taking pictures and analyzing with a yolov8 model
#   - sending emails
#   - registering ultrasound inputs
#   - controlling motors

__author__ = "Victoria Hoffmann"
__email__ = "victoria_hoffmann@gmx.ch"

# import (my own classes)
from control_final.keypad_final import *
from camera_final.image_recognition_final import *
from alerting_final.alert_email_final import *
from motion_final.ultrasound_final import *
from motion_final.movement_final import *
from indication_final.led_final import *

# not my own
import RPi.GPIO as GPIO
from time import sleep

# handling GPIO channel in use
GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)
GPIO.cleanup()
print("cleaned")

# objects
m1 = Movement()
m1.setBack()

ultra = Ultrasound()

l1 = Led()
l1.white(1)

# downloading model
key0 = Keypad()
value = key0.key_pressed()
# if * is pressed the model is downloaded - else the model is not downloaded no image
recogniton can be done
if value == '*':
    c1 = Image_recognition()
    downloaded = True
else:
    downloaded = False

stopped = False
while (not stopped):
    l1.white(0.5)
    sleep(0.5)
    l1.white(0.5)

    key1 = Keypad()
    value = key1.key_pressed()
    if value == 1:
        if downloaded == True:
            notSent = True
            while (notSent):
                # taking pictures
                det = c1.image_analysis()
                if (det != 0):
                    # indicating detection
                    l1.red(1)
                    sleep(1)
                    l1.red(1)

                # sending email
                e1 = Alert_email('real')
                e1.send()

```

```

        notSent = False
        stopped = True
        for z in range(3):
            ll.turquoise(0.5)
            sleep(0.5)
    else:
        ll.green(1)
        for i in range(0, 2):
            toDo = ultra.direction() #
            m1.setup()
            m1.setBack()
            if toDo == 2:
                m1.curveRight(2)
            elif toDo == 3:
                m1.curveLeft(2)
            elif toDo == 4:
                m1.forwards(1)
            else:
                m1.backwards(1)

            m1.setBack()

    else:
        ll.purple(1)
        sleep(0.5)
        ll.purple(1)
        print("model hasn't been downloaded")

elif value == 2:
    if downloaded == True:
        det = c1.image_analysis()
        if (det != 0):
            ll.red(0.5)
            sleep(0.5)
            ll.red(1)
        else:
            ll.green(1)
    else:
        ll.purple(1)
        sleep(0.5)
        ll.purple(1)
        print("model hasn't been downloaded")

elif value == 3:
    e1 = Alert_email('test')
    e1.send()

else:
    m1.setup()
    m1.setBack()

    if value == 4:
        m1.forwards(1)
        m1.setBack()
    elif value == 5:
        m1.backwards(1)
        m1.setBack()
    elif value == 6:
        m1.curveLeft(2)
        m1.setBack()
    elif value == 7:
        m1.curveRight(2)
        m1.setBack()

    else:
        stopped = True
        for z in range(3):
            ll.turquoise(0.5)
            sleep(0.5)

```

8.1.2 led_final.py

```

# description:
# In this class 8 states of an Led are defined by setting the pins as high or low
# The color will be used to indicate what the main program is doing

__author__ = "Victoria Hoffmann"
__email__ = "victoria_hoffmann@gmx.ch"

import RPi.GPIO as GPIO
from time import sleep
from variables_final.var_final import *

class Led:
    # constructor
    def __init__(self):
        GPIO.setmode(GPIO.BOARD)
        self.redPin = situation_general['redPin']
        self.greenPin = situation_general['greenPin']
        self.bluePin = situation_general['bluePin']

    # setting pins as output
    def setup(self):
        GPIO.setmode(GPIO.BOARD)
        GPIO.setup(self.redPin, GPIO.OUT)
        GPIO.setup(self.greenPin, GPIO.OUT)
        GPIO.setup(self.bluePin, GPIO.OUT)

    def turnOff(self):
        self.setup()
        GPIO.output(self.redPin, GPIO.LOW)
        GPIO.output(self.greenPin, GPIO.LOW)
        GPIO.output(self.bluePin, GPIO.LOW)

    # all -> white
    def white(self, t):
        self.setup()
        GPIO.output(self.redPin, GPIO.HIGH)
        GPIO.output(self.greenPin, GPIO.HIGH)
        GPIO.output(self.bluePin, GPIO.HIGH)
        sleep(t)
        self.turnOff()

    # RGB
    def red(self, t):
        self.setup()
        GPIO.output(self.redPin, GPIO.HIGH)
        GPIO.output(self.greenPin, GPIO.LOW)
        GPIO.output(self.bluePin, GPIO.LOW)
        sleep(t)
        self.turnOff()

    def green(self, t):
        self.setup()
        GPIO.output(self.redPin, GPIO.LOW)
        GPIO.output(self.greenPin, GPIO.HIGH)
        GPIO.output(self.bluePin, GPIO.LOW)
        sleep(t)
        self.turnOff()

    def blue(self, t):
        self.setup()
        GPIO.output(self.redPin, GPIO.LOW)
        GPIO.output(self.greenPin, GPIO.LOW)
        GPIO.output(self.bluePin, GPIO.HIGH)
        sleep(t)
        self.turnOff()

    # mixed
    def lightGreen(self, t):
        self.setup()
        GPIO.output(self.redPin, GPIO.HIGH)
        GPIO.output(self.greenPin, GPIO.HIGH)
        GPIO.output(self.bluePin, GPIO.LOW)
        sleep(t)
        self.turnOff()

```

```

def purple(self, t):
    self.setup()
    GPIO.output(self.redPin, GPIO.HIGH)
    GPIO.output(self.greenPin, GPIO.LOW)
    GPIO.output(self.bluePin, GPIO.HIGH)
    sleep(t)
    self.turnOff()

def turquoise(self, t):
    self.setup()
    GPIO.output(self.redPin, GPIO.LOW)
    GPIO.output(self.greenPin, GPIO.HIGH)
    GPIO.output(self.bluePin, GPIO.HIGH)
    sleep(t)
    self.turnOff()

```

8.1.3 movement.py

```

# description:
# In this class the 4 motors are controlled (2x2 bc 2 are crosswired to eachother)
# methods:
# - __left (both directions)
# - __right (both directions)
# - forwards & backwards
# - curves (__left & __right)
#
# - sleep(x) for the duration of turning
# - precaution before each swich of direction setBack
# - GPIO cleanup after use of motors

__author__ = "Victoria Hoffmann"
__email__ = "victoria_hoffmann@gmx.ch"

# both motors running at the same time
import RPi.GPIO as GPIO
from time import sleep
import serial
from variables_final.var_final import *

class Movement:

    def __init__(self):
        # set up --- global variables be aware of possible sideeffects
        GPIO.setmode(GPIO.BOARD)
        GPIO.cleanup()
        GPIO.setmode(GPIO.BOARD)

        self.__s = 95 # how much percent of potential speed should be used
        self.setup()
        self.speed1 = GPIO.PWM(self.__Motor1Ena, 100)
        self.speed2 = GPIO.PWM(self.__Motor2Ena, 100)

    def setup(self):
        GPIO.setmode(GPIO.BOARD)

        self.__Motor1Ena = situation_general['Motor1Ena']
        self.__Motor1A = situation_general['Motor1A']
        self.__Motor1B = situation_general['Motor1B']

        self.__Motor2Ena = situation_general['Motor2Ena']
        self.__Motor2A = situation_general['Motor2A']
        self.__Motor2B = situation_general['Motor2B']

        GPIO.setup(self.__Motor1Ena, GPIO.OUT)
        GPIO.setup(self.__Motor1A, GPIO.OUT)
        GPIO.setup(self.__Motor1B, GPIO.OUT)

        GPIO.setup(self.__Motor2Ena, GPIO.OUT)
        GPIO.setup(self.__Motor2A, GPIO.OUT)
        GPIO.setup(self.__Motor2B, GPIO.OUT)

# cleaning

```



```

def cleanup(self):
    GPIO.setmode(GPIO.BOARD)
    print("cleanup")
    GPIO.cleanup()

def setBack(self):
    GPIO.setmode(GPIO.BOARD)
    print("setBack")
    GPIO.output(self.__Motor1Ena, GPIO.LOW)
    GPIO.output(self.__Motor1A, GPIO.LOW)
    GPIO.output(self.__Motor1B, GPIO.LOW)
    GPIO.output(self.__Motor2Ena, GPIO.LOW)
    GPIO.output(self.__Motor2A, GPIO.LOW)
    GPIO.output(self.__Motor2B, GPIO.LOW)
    self.sleep(1)

# duration of movement
def sleep(self, t):
    sleep(t)

# __left
def __left(self, minus_speed):
    GPIO.setmode(GPIO.BOARD)
    self.speed1.start(self.__s - minus_speed)
    GPIO.output(self.__Motor1Ena, GPIO.HIGH)
    GPIO.output(self.__Motor1A, GPIO.LOW)
    GPIO.output(self.__Motor1B, GPIO.HIGH)

def __backwardsLeft(self, minus_speed):
    GPIO.setmode(GPIO.BOARD)
    self.speed1.start(self.__s - minus_speed)
    GPIO.output(self.__Motor1Ena, GPIO.HIGH)
    GPIO.output(self.__Motor1A, GPIO.HIGH)
    GPIO.output(self.__Motor1B, GPIO.LOW)

# __right
def __right(self, minus_speed):
    GPIO.setmode(GPIO.BOARD)
    self.speed2.start(self.__s - minus_speed)
    GPIO.output(self.__Motor2Ena, GPIO.HIGH)
    GPIO.output(self.__Motor2A, GPIO.LOW)
    GPIO.output(self.__Motor2B, GPIO.HIGH)

def __backwardsRight(self, minus_speed):
    GPIO.setmode(GPIO.BOARD)
    self.speed2.start(self.__s - minus_speed)
    GPIO.output(self.__Motor2Ena, GPIO.HIGH)
    GPIO.output(self.__Motor2A, GPIO.HIGH)
    GPIO.output(self.__Motor2B, GPIO.LOW)

# both
def forwards(self, t):
    print("forwards")
    self.__left(0)
    self.__right(0)
    self.sleep(t)

def backwards(self, t):
    print("backwards")
    self.__backwardsRight(0)
    self.__backwardsLeft(0)
    self.sleep(t)

def stop(self, t):
    self.setBack()
    self.sleep(t)

# curve
def curveLeft(self, t):
    print("curveLeft")
    self.__right(0)
    self.sleep(t)

def curveRight(self, t):
    print("curveRight")
    self.__left(0)
    self.sleep(t)

```

8.1.4 ultrasound_final.py [105]

```

# description:
# In this class the serial port of the arduino is read and displayed

#     front ultrasonic sensor connected to pins 2,3
#     right ultrasonic sensor connected to pins 6,7

#     100 -> backwards
#     200 -> turns right (wall disappeared)
#     300 -> turns left (wall in front)
#     400 -> forwards

__author__ = "Victoria Hoffmann"
__email__ = "victoria_hoffmann@gmx.ch"
__credits__ = "Robotics Back-End" #https://roboticsbackend.com/raspberry-pi-arduino-serial-communication/

import serial, time

class Ultrasound:
    # constructor
    def __init__(self):
        # connection to arduino
        try:
            self.arduino = serial.Serial("/dev/ttyACM0", 9600, timeout=1)
            time.sleep(0.1) # wait for serial to open
        except:
            print("error occured when establishing connection")

    # helper method - evaluation of the direction
    def evaluation(self, __ansInt):
        # number from arduino to number that stands for direction
        __index = (__ansInt // 100) - 1
        self.__instructions[__index] += 1

        # if three times same direction
        if self.__instructions[__index] == 3:
            self.__notReached = False
            self.__todo = __index + 1 # +1 because direction starts at 1 but list index at 0
            return

    # determining direction
    def direction(self):

        self.__instructions = [0, 0, 0, 0] # list of possible outcomes and how often appeared
        self.__notReached = True # threshold
        self.__todo = 0
        if self.arduino.isOpen():
            print("{} connected!".format(self.arduino.port))

            while self.__notReached:
                if self.arduino.inWaiting() > 0:
                    # reading serial port and formatting correctly
                    __answer = self.arduino.readline()
                    __answer = str(__answer)
                    __answer2 = __answer.translate({ord(i): None for i in "br\\'\r\n"})
                    print(__answer2)

                    # which direction most often -> evaluation
                    __ansInt = int(__answer2) # str->int
                    self.evaluation(__ansInt)

                    # remove data after reading
                    self.arduino.flushInput()

            return self.__todo
        else:
            print("Error")
            return 0

```

8.1.5 keypad_final.py [106]

```

# description:
# In this class the button pressed on a 'Membrane Swich Module 1PC' is registered and
returned
#
# -column pins as output: high
# -row pins as input: high
#
# --> set column output low (one at a time and cycle through them)
# --> if button is pressed -> input low -> know which button
#
__author__ = "Victoria Hoffmann"
__email__ = "victoria_hoffmann@gmx.ch"
__credits__ = "Gaven MacDonald" # https://youtu.be/yYnX5QodqQ4?feature=shared

# imports
from time import sleep
import RPi.GPIO as GPIO
from variables_final.var_final import *

class Keypad:
    # arrangement of buttons on membrane (class variable)
    __matrix = [[1, 2, 3, 'A'],
                [4, 5, 6, 'B'],
                [7, 8, 9, 'C'],
                ['*', 0, '#', 'D']]

    # constructor
    def __init__(self):

        GPIO.setmode(GPIO.BOARD)
        GPIO.cleanup() # to avoid conflicts if it was not closed properly
        GPIO.setmode(GPIO.BOARD)

        self.__col = situation_general['col']
        self.__row = situation_general['row']

        # coloumn as output
        for j in range(4):
            GPIO.setup(self.__col[j], GPIO.OUT)
            GPIO.output(self.__col[j], 1)

        # row as input
        for i in range(4):
            GPIO.setup(self.__row[i], GPIO.IN, pull_up_down=GPIO.PUD_UP)

        self.__not_pressed = True

    # determin which key pressed
    def key_pressed(self):
        while (self.__not_pressed):
            for j in range(4):
                GPIO.output(self.__col[j], 0) # cycling through setting output coloumn as low

                for i in range(4):
                    if GPIO.input(self.__row[i]) == 0:
                        __value = Keypad.__matrix[i][j]
                        print("key pressed:" + str(__value))
                        while (GPIO.input(self.__row[i]) == 0):
                            pass

                        sleep(0.4)
                        self.__not_pressed = False
                        break

                GPIO.output(self.__col[j], 1)

        GPIO.cleanup()
        return (__value)

```

8.1.6 image_recognition_final.py [107]

```

# description:
#   In this class a model is used to detect if people are lying on the ground.
#   A picture is taken (3x)
#   If a person (on the ground) is detected the image is saved(person indicated with a
#   colored frame)
#
# model:
#   3325 images were annotated with Roboflow
#   Yolov8 was used to train the model
#   -mAP:      89.7%
#   -precision: 98.8%
#   -recall:   86.4%
#
# model citation
#   @misc{
#     matura2_self.dataset,
#     title = { Matura2 self.dataset },
#     type = { Open Source self.dataset },
#     author = { MNG },
#     howpublished = { \url{ https://universe.roboflow.com/mng-7rqvv/matura2 } },
#     url = { https://universe.roboflow.com/mng-7rqvv/matura2 },
#
#     journal = { Roboflow Universe },
#     publisher = { Roboflow },
#     year = { 2023 },
#     month = { sep },
#     note = { visited on 2023-11-27 },
#   }

__author__ = "Victoria Hoffmann"
__email__ = "victoria_hoffmann@gmx.ch"
__credits__ = "Roboflow" # https://docs.roboflow.com/deploy/raspberry-pi

# import
import os
import matplotlib.pyplot as plt
import cv2
from roboflow import Roboflow
import time
import shutil
from variables_final.p_key_final import *

class Image_recognition:
    # class variable where all the documents are
    __folder = '/home/victoria/virt/robot_final/alerting_final/documents_email'

    # constructor
    def __init__(self):

        # accessing my roboflow self.model
        self.rf = Roboflow(api_key=my_dictionary['api_rob'])
        self.project = self.rf.workspace("mng-7rqvv").project("matura2")
        self.dataset = self.project.version(1).download("yolov8")
        self.model = self.project.version("1").model
        self.model.confidence = 40
        self.model.overlap = 20

    def __remove_doc(self, path):
        if os.path.exists(path):
            os.remove(path)

    def image_analysis(self):
        __det = 0
        # creating or overwriting textfile
        __file_path = os.path.join(Image_recognition.__folder,
        'textfile_recognition/prediction_precision.txt')
        f = open(__file_path, "w")
        f.close()

        # picture taking and analyzing repeated 3 times
        for i in range(1, 4): # 1,2,3 chosen to be easy to read

            __neutral_path = os.path.join(Image_recognition.__folder, "orig" + str(i) +
            ".jpg")

```

```

os.system("libcamera-jpeg -o {}".format(__neutral_path)) # taking picture

# future location of img
__output_path = os.path.join(Image_recognition.__folder, "recognized",
"prediction" + str(i) + ".jpg")
__alternat_path = os.path.join(Image_recognition.__folder, "not_recognized",
                              "no_prediction" + str(i) + ".jpg")

# try except bc no prediction can be made if there is no image
try:
    prediction = self.model.predict(__neutral_path) # letting self.model predict
except:
    # append to textfile
    f = open(__file_path, "a")
    f.write("Image " + str(i) + ":\n\tNo image and therefore no prediction. \n")
    f.close()

    # maintenance routine
    print("no image -> no prediction")

    # deleting previous file at this location
    self.__remove_doc(__output_path)
    self.__remove_doc(__alternat_path)

    # skipping this iteration
    continue

__objects_detected = len(prediction.predictions) > 0 # Object detected and
__model.confidence>=40

if __objects_detected:
    # append to text file
    f = open(__file_path, "a")
    f.write("Image " + str(i) + ":\n\tObjects detected! \n")
    f.write(str(prediction))
    f.close()
    __det += 1
    # saving image (with prediction) to location
    prediction.save(__output_path)

    # deleting previous file at this location
    self.__remove_doc(__alternat_path)
    self.__remove_doc(__neutral_path)

    print("obj. det.")

else:
    # append to text file
    f = open(__file_path, "a")
    f.write("Image " + str(i) + ":\n\tNo objects detected. \n")
    f.close()

    # deleting previous file at this location
    self.__remove_doc(__output_path)

    # for maintenance routine
    shutil.move(__neutral_path, __alternat_path)

    print("No obj. det.")

time.sleep(2)
return __det

```

8.1.7 alert_email_final.py [108]

```

# description:
#   In this class an e-mail is sent with the image(s) of the person that has fallen
#   (indicated with colored frame)
#   In addition the confidence level is appened to the body (plain text)

__author__ = "Victoria Hoffmann"
__email__ = "victoria_hoffmann@gmx.ch"
__credits__ = "The PyCoach" # https://youtu.be/g\_j6ILT-X0k?feature=shared

# import
import smtplib
import ssl
from email.message import EmailMessage
from email.mime.image import MIMEImage
from email.mime.text import MIMEText
from variables_final.p_key_final import *
from variables_final.var_final import *
import os

class Alert_email:
    # variables (same for every object)
    __email_sender = 'raspberr.pi4@gmail.com'
    __email_password = my_dictionary['email_aut']

    # constructor
    def __init__(self, situation):

        # situation
        if situation == 'real':
            dictionary = situation_real
        else:
            dictionary = situation_test

        self.situation = dictionary['situation']
        self.email_receiver = dictionary['email_receiver']
        self.subject = dictionary['subject']
        self.body = dictionary['body']

        # object created
        self.em = EmailMessage()
        self.em['From'] = Alert_email.__email_sender
        self.em['To'] = self.email_receiver
        self.em['Subject'] = self.subject
        self.em.set_content(self.body)

    def __attach_images(self, image_path, i, a):
        # if an image exists at that location

        if (os.path.exists(image_path)):
            with open(image_path, 'rb') as image_file:
                self.image_data = image_file.read()
                self.image = MIMEImage(self.image_data, name=a + 'prediction' + str(i) +
'.jpg')
                self.em.add_attachment(self.image)

    def send(self):
        # attach at most 3 images
        for i in range(1, 4):
            # detected, always attached
            image_path_pred =
'/home/victoria/virt/robot_final/alerting_final/documents_email/recognized/prediction' + str(
i) + '.jpg'
            self.__attach_images(image_path_pred, i, "a_")

            # attaches not detected in test runs but not in real
            if self.situation != 'real':
                # not detected
                image_path_no_pred =
'/home/victoria/virt/robot_final/alerting_final/documents_email/not_recognized/no_prediction'
+ str(
i) + '.jpg'
                self.__attach_images(image_path_no_pred, i, "no_")

```

```

        # attach at most 1 text file, always attached
        self.file_path =
'/home/victoria/virt/robot_final/alerting_final/documents_email/textfile_recognition/predictio
n_precision.txt'
        if (os.path.exists(self.file_path)): # if a text file exists at that location
            with open(self.file_path, 'r') as text_file:
                self.text_data = text_file.read()
                self.text_message = MIMEText(self.text_data)
                self.em.attach(self.text_message)

        __context = ssl.create_default_context() # more security

        # logging into server and sending email
        with smtplib.SMTP_SSL('smtp.gmail.com', 465, context=__context) as smtp:
            smtp.login(Alert_email.__email_sender, Alert_email.__email_password)
            smtp.sendmail(Alert_email.__email_sender, self.email_receiver,
self.em.as_string())

        # for maintenance routine
        print('email sent')

```

8.1.8 p_key_final.py

```

# passwords
__author__ = "Victoria Hoffmann"
__email__ = "victoria_hoffmann@gmx.ch"

my_dictionary = {
    # Roboflow Model
    'api_rob': '*****',

    # email login
    'email_aut': '**** * * * * *'
}

```

8.1.9 var_final.py

```

# dictionaries for different situations
__author__ = "Victoria Hoffmann"
__email__ = "victoria_hoffmann@gmx.ch"

situation_real = {
    'situation': 'real',
    'email_receiver': 'vifeho04@gmail.com',
    'subject': 'ALERT - FALL DETECTED',
    'body': """
        This is an alert!

        A person on the ground has been detected!
        Enclosed are the images along with the corresponding confidence level.
        Your immediate review and prompt follow-up actions are required instantaneously.

        Sincerely,
        ~robot

        -----

        """
}

situation_test = {
    'situation': 'test',
    'email_receiver': 'raspberr.pi4@gmail.com',
    'subject': 'e-mail maintenance (test)',
    'body': """
        This e-mail is intended for maintenance purposes only.

        It includes the documents from three folders:
        - recognized: pictures where an object has been identified
        - not recognized: pictures where no object has been identified
        - textfile_recognition: a textfile with information about the pictures... i.e.
confidence

```

```

        Sincerely,
        ~robot

        -----

        """
}

situation_general = {
    # keypad
    'col': [32, 36, 38, 40],
    'row': [12, 16, 22, 18],

    # led
    'redPin': 37,
    'greenPin': 35,
    'bluePin': 7,

    # movement
    'Motor1Ena': 29,
    'Motor1A': 31,
    'Motor1B': 33,
    'Motor2Ena': 11,
    'Motor2A': 13,
    'Motor2B': 15,
}

```

8.1.10 ultrasound_2_direction.ino

```

// In this program the distance between the ultrasound sensor and the nearest object are calculated via soundwaves
// The calculation is made for 2 ultrasound sensors
// Based on the calculation the direction in which the robot should move is decided
// Numbers stand for the different directions for them to be easily converted back to integers by the Raspberry Pi
// 100 -> stop
// 200 -> right
// 300 -> left
// 400 -> forward
//
// author: Victoria Hoffmann
// email: victoria_hoffmann@gmx.ch

// initializing pins
// right
int triR = 7; //trigger-pin
int echR = 6; // echo-pin
long tR = 0; // time (from emitting till receiving)
long dR = 0; // distance

//forward
int triF = 2; //trigger-pin
int echF = 3; // echo-pin
long tF = 0; // time (from emitting till receiving)
long dF = 0; // distance

int action=0;

// setup
void setup()
{
    Serial.begin (9600);
    pinMode(triR, OUTPUT); // Trigger-Pin is an output
    pinMode(echF, INPUT); // Echo-Pin is an input

    pinMode(triF, OUTPUT); // rigger-Pin is an output
    pinMode(echF, INPUT); // Echo-Pin is an input

```



```
}

void loop()
{
  //right
  //emitting soundwave
  digitalWrite(triR, LOW);
  delay[5];
  digitalWrite(triR, HIGH);
  delay[10];
  digitalWrite(triR, LOW);

  //calculation
  tR = pulseIn(echR, HIGH);
  dR = (tR / 2) * 0.03432; // /2 bc only one way, *0.03432 -> speed of sound

  delay(1000);

  //forward
  //emitting soundwave
  digitalWrite(triF, LOW);
  delay[5];
  digitalWrite(triF, HIGH);
  delay[10];
  digitalWrite(triF, LOW);

  //calculation
  tF = pulseIn(echF, HIGH);
  dF = (tF / 2) * 0.03432; // /2 bc only one way, *0.03432 -> speed of sound
  delay(1000);

  // deciding direction which will be printed in serial port

  if (dR<=15 && dF<=15){ //back
    action=100;
  }else{
    if (dR>40){ //has to turn right
      action=200;
    }else if(dF<50){ // has to turn left
      action=300;
    }else{ // forward
      action=400;
    }
  }
  Serial.println(action);
}
```

8.2 Extensions

8.2.1 Language processing model

This language processing model is designed to differentiate between situations requiring a call for immediate assistance and situations where the person feels fine but an email should be sent regardless.

Call:

1. (\$determiners:d) (\$body:b) (\$verb-feeling:v) \$negation:n \$positive:p
- My Leg is not okay
2. I (\$verb-feeling:v) \$negation:n \$positive:p
- I am not healthy
3. (please) (I) (\$verb-feeling:v) (\$bad-adv:b) \$verb-body:v2 (\$bad-adv:b2)
- I continue to be severely pained
4. (please) (I) (have) \$bad-adv:b (\$verb-body:v) (\$determiners:d) (\$body:body)
- I have severely wounded my neck
5. \$verb-body:v (\$determiners:d) \$body:b
- I dislocated my shoulder
7. (\$determiners:d) \$body:b (is) \$verb-body:v
- My knee is throbbing
8. (please) (I) (\$verb-help:v) (\$determiners:d) \$object-pos:o (please)
- Please I rely on an emergency service
9. \$exclamations:ex
- Help me
10. (please) (\$determiners:D) \$body:b (\$verb-feeling:v) (\$bad-adv:b2)
- My body suffers
11. (please) (I) (\$verb-help:v) (\$determiners:d) \$help:v2 (please)
- I require medical attention
12. (please) \$verb-alerting:v (\$determiners:d) (\$object-pos:o) (please)
- Please notify my relatives
13. (I) (have) \$verb-action:v (\$bad-adv:b)
- I have collapsed

Message:

1. I (\$verb-feeling:v) \$positive:p
- I feel fine
2. (\$determiners:d) (\$body:b) (\$verb-feeling:v) \$positive:p
- My knee is okay
3. (please) (I) (\$verb-feeling:v) (\$bad-adv:b) \$negation:n \$verb-body:v2 (\$bad-adv:b2)
- I feel no throbbing
4. (I) (have) \$negation:n \$verb-action:v (\$bad-adv:b)
- I have not slipped badly
5. (please) \$negation:n \$verb-alerting:v (\$determiners:d) (\$object-pos:o) (please)
- Please don't alert the paramedics
6. (please) (I) \$negation:n (\$verb-help:v) (\$determiners:d) \$help:v2 (please)
- I don't require first aid
7. (please) (\$determiners:D) \$body:b (is) \$negation:n (\$verb-feeling:v) (\$bad-adv:b2)
- My head doesn't hurt badly
8. (please) (I) \$negation:n (\$verb-help:v) (\$determiners:d) \$object-pos:o (please)

- *I don't need nine one one*
- 9. (*\$determiners:d*) *\$body:b* (*is*) *\$negation:n* *\$verb-body:v*
 - *My foot is not in pain*
- 10. (*I*) *\$negation:n* *\$verb-body:v* (*\$determiners:d*) *\$body:b*
 - *I haven't twisted my ankle*
- 11. *\$negation:n* *\$exclamations:e*
 - *No need to worry*
- 12. (*please*) (*I*) (*have*) *\$negation:n* *\$bad-adv:b* (*\$verb-body:v*) (*\$determiners:d*) (*\$body:body*)
 - *I haven't severely injured my head*
- 13. (*please*) (*I*) *\$negation:n* (*\$verb-feeling:v*) (*\$bad-adv:b*) *\$verb-body:v2* (*\$bad-adv:b2*)
 - *I don't feel injured*
- 14. *\$negation:n*
 - *Don't*
- 15. *False alarm*
 - *False alarm*

Verb-alerting	<i>alarm, alert, call, communicate, dial, email, inform, message, notify, phone, reach out to, ring, signal to, speak to, summon, telephone, text, warn.</i>
Verb-help	<i>call for, demand, depend on, desire, have to have, invoke for, need, rely on, request, require, seek, want.</i>
Verb-feeling	
Verb-body	<i>ache, aches, aching, agonized, bad, bleed, bleeding, bleeds, break, broke, broken, bruised, crushed, cut, damaged, disabled, dislocated, distressed, dying, fractured, gave out, give out, gives out, handicapped, hit, hurt, hurting, hurts, in agony, in discomfort, in distress, in pain, injured, limping, miserable, numb, numbed, pained, paralyzed, pulled, scarred, scraped, sensitive, sore, sprained, strained, suffering, swell, swells, swollen, tear, tears, tender, throbbing, tore, torn, tortured, twist, twisted, twists, wounded.</i>
Verb-action	<i>am lying, am lying on the floor, am lying on the ground, collapsed, fainted, fallen, fell, found myself on the floor, found myself on the ground, lost balance, lost my balance, sagged, sank, sank to the floor, sank to the ground, slipped, staggered, stumbled, toppled, toppled over, tripped, tumbled.</i>
Body	<i>ankle, ankles, arm, arms, back, back of head, back of my head, body, bone, chest, collarbone, ear, ears, elbow, elbows, eye, eyes, feet, finger, fingers, foot, forearm, forearms, forehead, hand, hands, head, heel, heels, hip, jaw, knee, knees, leg, legs, neck, nose, palm, palms, pelvis, rib, ribcage, ribs, shin, shins, shoulder, shoulders, spine, tailbone, temple, temples, tendon, tendons, thigh, thighs, thumb, thumbs, toe, toes, upper arm, upper arms, wrist, wrists.</i>
help	<i>aid, assistance, bandage, bandages, checkup, evaluation, examination, first aid, help, medical attention, medication, pain relief, rescue, support, treatment.</i>
object-pos	<i>ambulance, anybody, anyone, caregiver, child, children, clinic, contact, contacts, daughter, doctor, emergency contact, emergency services, friend,</i>

	<i>healthcare provider, husband, neighbours, nine one one, paramedic, paramedics, personal assistant, relative, relatives, rescue team, somebody, someone, son, wife.</i>
Exclamation	<i>accident, emergency, fall, help me, injury, need to worry, oh no, pain, please, urgency, worries, worry.</i>
Determiners	<i>a, an, my, the.</i>
Negation	<i>barely, by no means, cannot, can't, did not, didn't, do not, does not, doesn't, don't, hardly, has not, hasn't, have not, haven't, is not, isn't, never, no, no way, none, not, not at all, not ever, not in any way, not in the faintest, not in the least, not in the slightest, stop, under no circumstance, wrong, wrongly.</i>
Bad-adv	<i>Awfully, badly, dreadfully, horribly, miserably, negatively, regrettably, severely, terribly.</i>
positive	<i>all good, alright, awesome, brilliant, comfortable, contempt, content, excellent, fabulous, fantastic, fine, fit, good, great, happy, healthy, intact, marvellous, okay, optimal, perfect, safe, sound, splendid, steady, superb, survived, terrific, unhurt, uninjured, undamaged, well, whole, wonderful.</i>

8.2.2 Autonomous phone control

```

# description:
# This program opens the connection to an Android phone, opens WhatsApp, and calls a contact.
# However for other phones the location where the mouse has to go might be different.
# Therefore the coordinates would have to be adapted
#
# with this code snippet the wanted location of the mouse can be determined
# while True:
#     print(pg.position())
#     time.sleep(1)

__author__ = "Victoria Hoffmann"
__email__ = "victoria_hoffmann@gmx.ch"

import pyautogui as pg
from pynput.mouse import Button, Controller
import time
mouse = Controller

def click_sleep(x, y, sleep_time):
    pg.click(x, y)
    time.sleep(sleep_time)

def swipe_pin():
    pg.click(500, 600)
    time.sleep(1)
    pg.mouseDown(button='left')
    pg.move(0, -200, 0.1)
    pg.mouseUp(500, 200, button='left')

def enter_password():
    coordinates = [(440, 530), (440, 460), (500, 400), (580, 460), (580, 530), (440, 400)]
    for coord in coordinates:
        pg.click(coord[0], coord[1])
        time.sleep(2)

# open phone connection
click_sleep(9, 16,1) #click menue(raspberry)
click_sleep(31, 240,1) #system tools
click_sleep(250, 237, 5) # srcrpy

# Swipe to enter pin
swipe_pin()

# Enter password
enter_password()

# The second part of the code
click_sleep(510, 710,1) #click menue on phone
click_sleep(460, 650,1)#whatsapp
click_sleep(600, 145, 2) # search contact
pg.typewrite("Mama") # enter contact name
click_sleep(460, 200,1) # click on contact
click_sleep(555, 145,1) # call






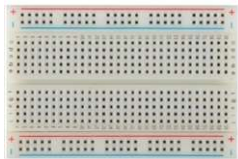
```



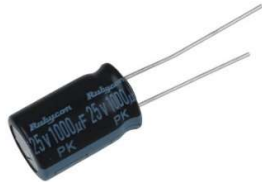


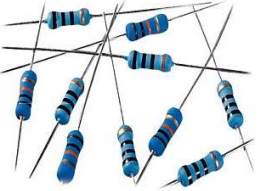
8.3 Materials

8.3.1 Software

- Machine learning software library: OpenCV
- Python library for modification of arrays: Numpy
- Extension of OpenCV: Imutils
- OS: bookworm (not buster or bullseye)
- Annotating images: Roboflow
- Neural network: yolo-v8
- Language: Python, and Arduino

8.3.2 Hardware

<p>Primary device: Raspberry Pi 4B+</p>	<p>Subordinate device: Arduino Uno</p>	<p>MicroSD card (...GB)</p>
 <p><i>Figure 31: Raspberry Pi 4B+, copied from [109]</i></p>	 <p><i>Figure 32: Arduino Uno, copied from [110]</i></p>	 <p><i>Figure 33: MicroSD Card, copied from [111]</i></p>
<p>Raspberry pi camera version 2.1</p>	<p>Switch Membrane Module 1PC</p>	<p>Breadboard</p>
 <p><i>Figure 34: Raspberry pi camera version 2.1, copied from [112]</i></p>	 <p><i>Figure 35: Membrane Switch Module 1PC, copied from [113]</i></p>	 <p><i>Figure 36: Breadboard, copied from [114]</i></p>

4 DC Motors+4 tires	H-Bridge	Capacitor
		
<p><i>Figure 37: 3V-12V DC Motors + corresponding wheels copied from [115]</i></p>	<p><i>Figure 38: H-Bridge, copied from [116]</i></p>	<p><i>Figure 39: 1000mF 25V capacitor, copied from [117]</i></p>
Ultrasound sensor (x2)	RGB - LED	Resistors
		
<p><i>Figure 40: Ultrasound sensor, copied from [118]</i></p>	<p><i>Figure 41: RGB LED, copied from [119]</i></p>	<p><i>Figure 42: Resistors, copied from [120]</i></p>

Soldering board	Robot Chassis	Exhaust fan
		
<p><i>Figure 43: Board for own circuit copied from [121]</i></p>	<p><i>Figure 44: Robot Chassis, copied from [122]</i></p>	<p><i>Figure 45: Exhaust fan, copied from [123]</i></p>
USB cable	USBc cable	Arduino cable
		
<p><i>Figure 46: USB cable, copied from [124]</i></p>	<p><i>Figure 47: USBc cable, copied from [125]</i></p>	<p><i>Figure 48: Arduino cable copied FROM [126]</i></p>
MM, MF, FF cables	power bank	Screws
		
<p><i>Figure 49: MM, MF, FF cables copied from [127]</i></p>	<p><i>Figure 50: power bank copied from [128]</i></p>	<p><i>Figure 51: screws copied from [129]</i></p>

Declaration of independence

Die Unterzeichnete bestätigt mit Unterschrift, dass die Arbeit selbständig verfasst und in schriftliche Form gebracht worden ist, dass sich die Mitwirkung anderer Personen auf Beratung und Korrekturlesen beschränkt hat und dass alle verwendeten Unterlagen und Gewährspersonen aufgeführt sind.

Autorin: Victoria Hoffmann



Herrliberg, 8. Januar 2024

Victoria Hoffmann