Professional Bachelor Applied Information Technology

State classification of elevator doors to assist emergency detection in elevator networks

Ignace Jordens

Promoters:

Oliver Krauss MSc — University of Applied Sciences Upper Austria

dr. Johan Cleuren MSc — PXL University of Applied Sciences and Arts

Bachelor paper Academic year 2018-2019

Professional Bachelor Applied Information Technology

State classification of elevator doors to assist emergency detection in elevator networks

Ignace Jordens

Promoters:

Oliver Krauss MSc — University of Applied Sciences Upper Austria

dr. Johan Cleuren MSc — PXL University of Applied Sciences and Arts

Bachelor paper Academic year 2018-2019

Acknowledgements

After an education of five years, which was mainly focused on IT, I ended up in Austria to conduct my final internship combined with writing my bachelor paper.

I could not deliver this paper without the help of certain individuals, who supported me the past five years and kept believing in my capabilities.

First and foremost, I would like to thank the entire research team AIST, for continuously supporting me during my internship and for their guidance and patience. Particularly Oliver Krauss, who was my mentor and point of contact during this time. I am very grateful for his support and that he gave me the possibility to conduct my internship in a very interesting project. Rainer Meindl, another member of the AIST team, also gave me continuous feedback on my delivered code which I am very grateful for. Connected to this, I would also like to thank Johan Cleuren for the continuous guidance while writing this paper.

Of course, I was not able to arrive at this point of my life without the knowledge I acquired during the past years. Therefore, I would like to thank every teacher at PXL who contributed to the fact that I have the opportunity to finish my education with so much knowledge and insight into my specific area of expertise.

I would of course like to thank my parents and my girlfriend for supporting me during my absence and to give me the possibility to go to Austria for a semester.

Special thanks go to my old teachers of the Sint-Franciscus College, Maarten Kempeneers and Lori Nys, who contributed to the foundations of my knowledge in IT and are still always available if I have any problems or questions. I also want to thank Daan Vankerkom for getting me into programming and assisting me during all these years.

My sincerest gratitude to everyone who contributed to let me become the person I am today.

Abstract

EDEN, which stands for emergency detection in elevator networks and is a project of the AIST research group at the University of Applied Sciences Upper Austria, aims to use sensors and cameras to automatically detect emergencies in elevators, evaluate them, put them into context and take appropriate actions. This bachelor's thesis tackles the classification of the status of an elevator door. In order to classify certain emergencies correctly, it is vitally important that the classification system knows all the involved parameters. A certain situation can have different interpretations if all parameters are considered. The current state of the door is in this case a very important parameter.

The EDEN project uses an Intel RealSense D435 camera as a device to capture images and depth information. These images and their corresponding depth information are analysed by the project, which is written in C++ and uses the OpenCV framework for computer vision.

A first part of this paper is to research the usages of the provided depth and RGB information by the camera to detect the status of an elevator door. In the research, the different possible approaches are discussed. The most feasible approaches are elaborated in a proof of concept. The first step of the detection of the door status is localising the door itself using depth and RGB information. This is followed by the extraction of the floor, which is achieved by using edge detection and extraction techniques. With the location of the door and the floor known to the application, the status of the door can be determined. In order to correctly classify this status, the research focuses on different methods to detect the status and strategies to reduce noise interference which is caused by either the recording equipment or objects blocking the vision of the door.

The second part of the paper focuses on testing the capabilities of the Intel RealSense D435 camera, more specifically the accuracy of the depth information it can provide. To comply with certain ISO-standards, an elevator car cannot have a height difference of more than 20 millimetres compared to the outlying floor. The research investigates if the D435 camera can detect such a small height difference while still maintaining a visual overview of the entire elevator car, so the application can still detect occurring emergencies.

Table of contents

List of figures

List of tables

List of abbreviations

AIST	Advanced Information Systems and Technology
EDEN	Emergency Detection in Elevator Networks
GARDEN	GARDEN Approach for Reflection Resistant Distress Detection in Elevator Networks
CREATION	Continuous Emergency Adaption and Test Integration Network
KIT	Kit for Imaging Testdata
FH OÖ	Fachhochschule Oberösterreich
GDPR	General Data Protection Regulation
OpenCV	Open Source Computer Vision Library
RAM	Random Access Memory
RGB	Red Green Blue
RMS	Root Mean Square
SDK	Software Development Kit
ROI	Region of Interest

Glossary

Bounding shape	A geometric shape which contains all the points of a specific collection of points.
Classified object	An event which occurred in the elevator and has been analysed by GARDEN, returns a classified object. This object contains certain labels like ‘person’ or ‘bag’.
CREATION	The implemented server-side software part of the EDEN project.
Disparity map	The apparent pixel difference or motion between a pair of stereo images.
GARDEN	The implemented client-side software part of the EDEN project.
GARDEN pipeline	A combination of all the different small steps, conducted in GARDEN. The steps are executed sequentially, hence in a pipeline manner.
Image Averaging	Enhancing the correctness of a pixel value, by using its neighbouring pixel values
RMS error	Root Mean Square error is used to measure differences between values. The RMS represents the square root of the differences between predicted values and observed values.
State machine	The part of GARDEN which combines all the extracted information from a frame into a classified object.
Stereo images	A depth image which was generated by combining two images with different view angles, looking at the same object.
View Client	An existing software application by View Promotion GmbH which acts as a conversation partner for GARDEN and handles received messages, sent by GARDEN, accordingly.
View Server	An existing software application by View Promotion GmbH which acts as a conversation partner for GARDEN and handles received messages, sent by GARDEN or the View Client, accordingly.

Introduction

Today, the safety of the general public is more important than ever. While the safety measures during these uncertain times are kept on an all-time high, a next important case is to improve the safety long-term. Railway stations and airports are guarded and surveyed by the military and private security organisations, who continuously look for possible emergencies. However, as they are all just human, they cannot see everything or be everywhere.

The EDEN project focuses on a small part of this growing issue, it aims to improve the safety of elevator users. For the EDEN project, a camera is installed in an elevator that is connected to an application. This application continuously analyses the frames it receives, in order to detect emergencies that can occur in elevators. These frames consist of both a regular image and an image showing the depth.

A detection of that sort is not an easy task for an application, since it must consider all the involved parameters. One of these important parameters is the current status of an elevator door, since this can influence the classification of a certain event.

A very important question in this case is if the data received from a camera can be used to provide a status for a door. Moreover, can this be of assistance to the classification of emergencies in an elevator?

As a first step the entire project, its goal and its structure are discussed, together with the purpose and usage of the door status. There is also a brief overview of the used technologies and frameworks.

Continuing the research part, the possible algorithms to tackle the occurring problems are examined and compared with each other. The conclusion of this research brings forward some methods and algorithms forwards, which then are used to elaborate on a solution for the problem, as a proof of concept.

The research group Advanced Information Systems and Technology at the University of Applied Sciences Upper Austria conducts the research and execution of the EDEN project. Regarding this project, they also collaborate with the company View Promotion GmbH.

I. Traineeship report

1 About the company

1.1 University of Applied Sciences Upper Austria

The Fachhochschule Oberösterreich (FH OÖ) is a University of Applied Sciences located in the province of Upper Austria. Their company structure, which is shown on their website [1], is displayed in figure 1.

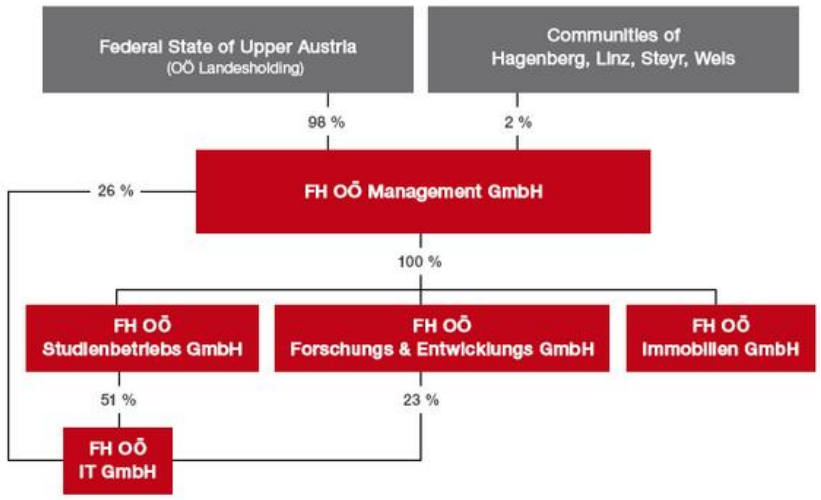

Figure 1 Company structure FH OÖ

The FH OÖ has a research and development department, which is seen in figure 1 as 'FH OÖ Forschungs & Entwicklungs GmbH'. This department exists out of different research groups, specialised in their own area of expertise. The research groups are part of the FH OÖ organisation and have employees which are mainly alumni, professors or students of the university.

1.2 Advanced Information Systems and Technology

The research group Advanced Information Systems and Technology (AIST), which is one of the research groups at the FH OÖ, researches software systems in many different areas. The main areas which AIST focuses on are communication, assistive and analysing systems and the integration for assistance of humans.

Looking at the ongoing projects displayed on their website, their current focus lays on projects using techniques like Virtual and Augmented Reality, mobile systems and autonomous devices. Projects to improve the healthcare industry are also part of their curriculum. Current and previous cases like REPO and WIRE are for example focused on the improvement of the interoperability between radiologists and digitalising their workflow. [2]

Everything considered, this suggests that many of the projects at AIST are focused around the well-being of the public. This can either be like the health platform, which was mentioned earlier, but it can also include the small things in life, like using Data Mining and Natural Language Processing to identify all the menu texts of restaurants in a certain area.

With this background in mind, it is no surprise the Emergency Detection in Elevator Networks (EDEN) project arrived on their doorstep, as the indirect goal of the project is to improve the general well-being of society and the public, by aiding them in a specific way.

As such AIST, or any research team for that matter, cannot be compared to a typical company, since they are not really selling any products but develop concepts, algorithms, and standards. They test their research by making prototypes.

2 EDEN - Emergency Detection in Elevator Networks

2.1 Goal

"The EDEN project aims to use sensors and cameras to automatically detect emergencies in elevators, evaluate them in order to put them into context and take appropriate action. [...] The planned developments should create a safe environment, as well as to help people with fears [sic] of elevators." [3] EDEN is funded by the Austrian Research Promotion Agency and is in collaboration with an industry partner named VIEW Promotion GmbH.

Figure 2 Logo EDEN

In other words, EDEN provides an architecture and framework combined with specific algorithms which enable an automated detection of use-cases which require human attention.

A first area of interest are situations where human lives are in danger. This mostly concludes detecting unconscious persons.

A next step in the detection focuses more on the situations, which can endanger elevator users. Some example cases in this area are forgotten bags, objects blocking the entrance, detection of weapons or technical defects on the elevator itself.

Once EDEN detects an emergency, it should notify specific instances, responsible for this occurrence. This means certain cases should be treated differently, according to the elevator type and place. In the case of an elderly person fainting in the elevator of a retirement home, the system should take different actions compared to when it detects a person holding a knife in an airport.

All these different types of cases can and should not be implemented on a developer level. Therefore, the EDEN project is split into different parts, each with their own responsibilities. So, EDEN can be seen as a general overhead framework combining the sub lying parts, called GARDEN, which stands for GARDEN Approach for Reflection Resistant Distress Detection in Elevator Networks and CREATION, which stands for Continuous Emergency Adaption and Test Integration Network.

In addition to these two underlying parts, EDEN also includes a connection to external servers of the project partner VIEW Promotion GmbH. These servers contain information about the elevator itself as a device. This includes for example the elevator movement. Since these parts are not of notable importance to the further elaboration of the internship, there will be no detailed explanation.

Figure 3 is a visual representation of the EDEN framework.

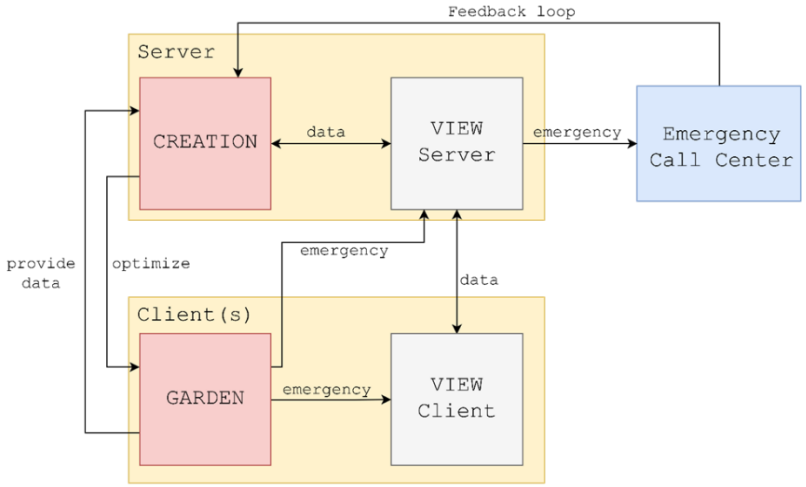

Figure 3 EDEN architecture

2.2 GARDEN

GARDEN is the client-side architecture of the EDEN infrastructure. It processes images and sensor data to detect and classify emergencies. GARDEN can thus be seen as a separate application, with its own closed ecosystem, running in or near the elevator itself.

GARDEN is based on a pipeline concept. The first so-called element of this pipeline is a data producer, like the Intel RealSense camera, more on that later. The data captured by this device is sent to the next element in the pipeline, which modifies or extracts data from it, makes it available to other elements and sends it to the next element. This approach makes the entire pipeline very modular and easy to change, since the configuration of this pipeline is extracted to a JSON file. If there is a need to change the order of the elements or change input parameters, no code must be recompiled, and everything can just be configured within the configuration files.

The client side then, is responsible for the gathering of input data and its processing. As previously mentioned, it modifies and extracts data from input images provided by the RealSense camera. Once a frame is received, GARDEN conducts pre-processing on the images to reduce noise and imperfections in the data. In a next step, GARDEN differentiates possible persons and objects in the elevator itself, assisted by a background subtraction algorithm, which cancels out detections which are part of the elevator itself.

The parts of the image where there are likely to be people or objects, are fed to a classifier algorithm. In addition to the extracted parts of the image, this algorithm uses information about the state of the doors and the elevator's movement. With all this information, the classifier will differentiate objects from people and label them with the right case, for example "Unconscious Person" or "Forgotten Bag". The elaboration of the classification within GARDEN, named "State Machine" is conducted by Rainer Meindl and will be published as his master's thesis [4].

Taking everything into consideration, GARDEN conducts many processing steps in order to classify an image to an emergency event. The result of this process is a known emergency or event, which can be further processed by the next element in the EDEN architecture.

GARDEN is written in the C++ programming language.

2.3 CREATION

CREATION is the server-side architecture of the EDEN infrastructure with the aim to optimise GARDEN. To achieve a feedback loop it collects user-feedback when an emergency is classified incorrectly, in addition to the correct classification.

A secondary function of this part of EDEN is continuously testing GARDEN on its correctness, with predefined test data. If CREATION notices a mismatch in the classification, it can easily change GARDEN's parameters, since they are extracted to a JSON configuration. This provides a continuous feedback-loop, which ensures the correctness of GARDEN's classification and initialisation, without having to recompile existing code.

GARDEN communicates with CREATION by generating messages using protocol buffers and sending them via a message queue.

"Protocol buffers are a flexible, efficient, automated mechanism for serialising structured data – think XML, but smaller, faster, and simpler. You define how you want your data to be structured once, and then you can use special generated source code to easily write and read your structured data to and from a variety of data streams and using a variety of languages." [5]

CREATION is deployed as a microservice architecture with services written in C++, Python and Java.

2.4 Hardware

To capture everything happening in an elevator, GARDEN connects with a camera which provides a real-time video stream. This camera is an Intel RealSense D435 and streams both RGB and depth information. [6]

The RealSense camera captures frames with a resolution up to 1280x720 and enhances its depth data by illuminating objects with an infrared projector.

This camera is hung in the elevator, opposite the door, to have a clear vision of everything going on inside, as shown in figure 4.

Figure 4 Camera location

To access the data stream of the camera, GARDEN implements the Intel RealSense SDK 2.0. [7] Via this toolkit, which has a C++ implementation, GARDEN can access the data feed of the camera and change built-in parameters and filters.

The implementation of this interface and usage of the RealSense SDK is already existing within the GARDEN project and is seen as a usable framework for further implementations.

The Intel RealSense camera also provides build-in filters and post-processing functionalities as a first step in noise cancellation. The most important one to mention is the hole filling filter, which reduces the amount of unknown depth.

3 Internship elaboration

3.1 Technologies

To give a clear understanding of the entire project, the needed technologies to achieve it are briefly explained in this chapter.

3.1.1 OpenCV

“OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library.” [8] GARDEN uses the library to provide real-time computer vision algorithms. OpenCV can for example be used to track moving objects or recognise faces.

GARDEN uses OpenCV version 3.4.1 as its core algorithm, for both pre-processing and classification. The pre-processing concludes matrix transformations, blurring, dilating and eroding. To assist classification, it uses contour detection, feature calculators and implementations for neural networks.

3.1.2 C++

As previously stated, GARDEN is written in C++ version 17. The application needs to be very memory and processing efficient because it must run on low tier hardware while computing a lot of data. Additionally, OpenCV itself is written in C++ so its primary interface is also in this language. This makes C++ the perfect tool for the job.

3.1.3 Google Test

According to Wikipedia, Google test is a unit testing library for C++, based on the xUnit architecture [9]. Within the GARDEN application this framework is used to write Unit and Integration Tests. It provides features such as test discovery, assertions, user-defined assertions and report generation [10].

3.1.4 Versioning

The EDEN project uses Git as the versioning tool for the entire architecture. As a central repository server, a self-hosted GitLab server runs on the servers of the project partner View Promotion.

To comply with the company standards, the developers should commit their new code at least multiple times a day, so there is a nice overview of completed steps. Each day the developers should push their code to the remote branch as a safety measure and to keep other developers on the same branch up-to-date.

The used branching strategy is feature branching. This means that every new issue or feature, should be developed on its own branch, derived from the master. [11]

When a feature is completed, the corresponding branch is rebased on the master to maintain a clean history.

3.2 Data processing

As already mentioned, the GARDEN application is a pipeline concept, so it is completely dependent on the configuration of this pipeline. This configuration is extracted to a specific JSON file.

At the start-up of the application, this file is read and interpreted in order to enable the correct classes and elements at runtime.

The JSON configuration defines all the separate elements. An element defines a step to be taken during runtime. Only the classes which are requested by the JSON configuration, get executed and all classes which are used in the pipeline derive from an interface DataProducer.

In order to correctly implement this interface, each subclass should have a constructor which excepts JSON values, a method which produces or extracts the data and a method which checks if the producer can be executed.

An element in the configuration file defines the class to be called, the class of the input parameter, the class of the output parameter and the parameters for the constructor.

An element also has a corresponding connection, which defines the element providing the input and the target element for the output.

The order of an element's appearance in the file also resembles the order in which it will be requested in the application at runtime.

This actual JSON configuration is shown in Appendix A. At first, there is a general collection of all the pipeline elements with their corresponding connections. The second block represents a 'PIPELINE_ELEMENT', while the third block represents a 'CONNECTION'. Figure 5 is a visual representation of the pipeline.

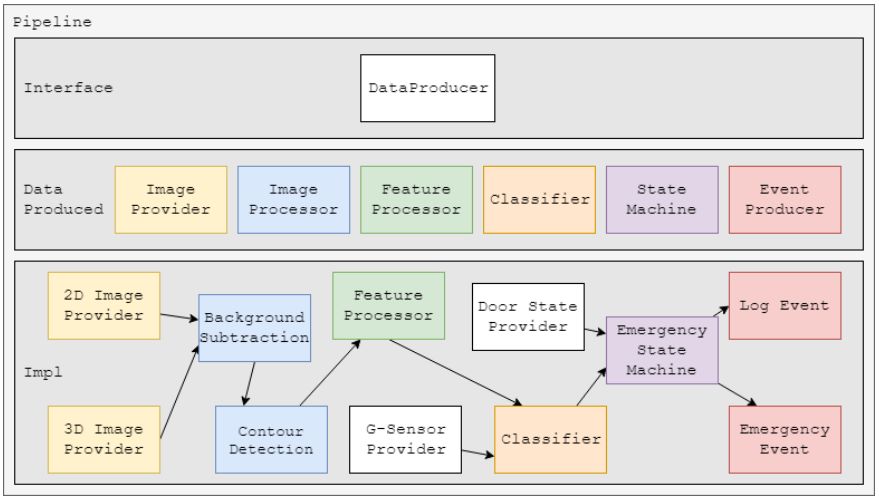

Figure 5 JSON pipeline visual representation

All elements inherit from DataProducer in the interface layer. The 'Data Produced' layer defines the more specific interfaces for each type of provider. The implementation layer visualises the flow of the data and all the major elements in the pipeline. The pipeline starts with the capture of frames and sends them to the processing steps. These steps process the image even further and extract data from them. An example of this processing is the Door State Provider, which aims to use previously extracted data from the image like the contours, together with the image itself to detect the status of the door. This data is saved into the current frame and sent to the Emergency State Machine. Using all the extracted information, the State Machine classifies the current status in the elevator correctly and creates a corresponding event. The exact working of this part of GARDEN is described in the master thesis of Rainer Meindl [4].

4 State classification of elevator doors

4.1 Usage of door information

As previously stated, GARDEN's goal is to classify emergencies and take appropriate actions. In order to classify these emergencies correctly the system should be aware of all possible parameters, in order to assist the classification. One of these very important parameters is the state of the elevator doors. When for example a person is detected in one frame, but disappeared in the next one, this could suggest a bug in the system. When the system knows that the door was closed in both frames, the certainty of a failure is more prominent, since a person cannot just disappear from a closed elevator.

The state detection of the elevator doors will thus assist the "State Machine" in correctly classifying the detected scenarios. A second use of knowing the status of the door is detecting technical issues. If for example the door is not closing anymore or only partly opening, this can suggest a defect.

4.2 Computer vision versus existing sensors

Using computer vision to detect the status of the door might sound like reinventing the wheel; normally the system can be linked to the elevator itself and request the current door state.

In a few cases, this is a valid statement. The system should be able to read out the parameters of the elevator. Unfortunately, this is not always possible or preferable. First-hand experience of the project partner View Promotion suggests that these sensors are one of the first things to go wrong in elevators. They are located inside the elevator, a machine that is not cleaned often, and they become dirty over time. This leads to false read-outs or sensor failure.

The second most likely thing to go wrong in elevators concerns the motors of the door. If the door is closing while somebody is standing in the doorway, the door will hit that person and feel an obstruction. Due to the blocking force, the door will open again. This is mechanically a very demanding task for the motors, which causes them to fail sooner than expected.

To conclude the previous statements, the sensor data from the elevator itself is useful but the level of accuracy is highly dependent on the elevator itself. The system must be able to read the data, which requires a different implementation for each type of elevator and working sensors. However, most importantly, the elevator should have those sensors installed in the first place.

Using computer vision, the detection can theoretically be implemented in any type of elevator. Furthermore, the system can detect a person blocking the door ahead of time and send this data to the elevator, which eliminates the wear on the door motors.

Disadvantages of using the computer vision are mostly the development cost and the accuracy. This accuracy is highly dependent on the number of people in the elevator. If the vision of the door is obstructed or completely blocked, the system cannot detect the status correctly.

Nevertheless, the advantages of the system outweigh the disadvantages, since there are easy workarounds for this when the system considers all parameters.

II. Research topic

1 Research questions

As previously cited, GARDEN is connected to an Intel RealSense camera, which captures a stream of both RGB and depth information. Within the application, the functionality is expanded by implementing the Door State Provider, as explained previously.

Before the start of the implementation of this functionality as a proof of concept, there are certain issues which need to be resolved.

A first issue is the definition of a 3D model of the elevator. The application cannot just load an image and tell by itself where the door is and if it is open. This logic must be programmed dynamically and be applicable for different types of elevators. Combined with this logic, all the parameters should thus be variable and interchangeable. The definition of the model should be done before the actual GARDEN pipeline starts running. A second question to be answered, related to this, is the extraction of this model to the pipeline.

As the next step, the application uses this model to check if the door is open or closed. Issues regarding this step are the definition when something is open or closed and most importantly, how to handle incorrect or disturbed data and combining this with a confusion model.

The final part of the research is an investigation for the feasibility to detect the height difference between the floor of an elevator and the floor of a building, with the Intel RealSense camera. Since this gap can legally be 20 millimetres to the utmost.

Because the transmission of the frames provided by the Intel RealSense camera occurs in real-time, the Door State Provider must handle the input immediately. As this provider is an important step which is executed whenever a new frame is received, it is of great significance that the Door State Provider is as optimised as possible so there are no significant delays while detecting the states.

In conclusion, the following main question can be stated.
How can the data from the Intel RealSense camera be used in the provision of the door status, to be of assistance to the classification of emergencies in an elevator?

This main question can be separated in the following questions:

- How can the elevator door and the floor be separated using depth and RGB information?
- How can the status of the elevator doors be extracted using depth and RGB information?
- How can an Intel RealSense camera improve the safety of an elevator user and an elevator's compliance to specific ISO-standards?

2 Research method

In order to achieve the optimal results for this research, it starts with the search for possible methods to extract the location of the door and the floor of an elevator and the way in which this can be transferred or merged into a 3D model. Firstly, a conceptual model is created to pinpoint the preferred approach in order to find the required parts. This means distinguishing the different panels of the elevator in the depth data to see if it can be used.

Once it is certain that the different panels in an elevator can be distinguished visually, the search for methods to do this in OpenCV starts. Using existing papers, the OpenCV documentation and OpenCV tutorials, a list of possible approaches is created. The research continues by comparing the methods found with regard to their speed, accuracy, scalability and in general, whether they work and return the expected result. A final step in this part is comparing the methods for acquiring input data for this training method, as it should be run before the actual pipeline.

When the application continues to the actual detection of the door status, the first part of the elaboration focuses on which possible approaches there are available to check the status of the door and comparing these again for accuracy, speed and fault tolerance. This part is then expanded with ways of reducing of disruptions in the data.

The research regarding the height difference is conducted as follows. The specifications of the Intel RealSense camera are of major importance, since the device itself should be able to detect height changes within a 10 millimetres threshold. Furthermore, the clarity of the height difference will be substantially harder to determine when the detection target is further away. Additionally, since the camera uses lasers to measure depth, reflecting surfaces do provide additional noise to the data. When the parameters meet the criteria, a visual conclusion is conducted if the camera is to be used for this appliance. This part of the research starts by recording the training data and is followed by a visual difference inspection, to be finalised by either a proof of concept or an alternative method.

The approach to each research question is the following. First, a comparison is made between all possible methods to achieve the requested output. In the next step, the best option is selected. If there is only one option, a proof of concept is made using it. If there are multiple options, the speed, correctness and flexibility are taken into account and compared, to select the best method for this application.

All the measurements regarding the computation time and the speed of certain algorithms are executed on a desktop with an Intel Core i7-6700k CPU, 16 GB of RAM, Intel Z170 Express Chipset, a NVidia GeForce GTX 1070 GPU and a Windows 10 x64 operating system.

3 Research elaboration

The goal of this research is thus to create an approach to find and label the different panels in an elevator. These panels are for example the floor, the door or the walls. The individual parts must be separated, so each individual element can assist the further classification of the door state.

The input of the application is, as said earlier, a video stream with RGB and stereo frames. Once GARDEN receives this data, the application executes certain logical steps before it can tell which part of the elevator is located at a certain spot. As the application does not reason on its own where each part is found, this logic must be implemented before GARDEN can detect the status of the door.

There are different possible approaches to implement the search and classification of each section into the GARDEN project. The following chapters describe the possible different approaches to tackle this for both the door and the floor. Once the methods to acquire these parts are researched, the focus changes to methods that can be used for the actual state detection.

The research only discusses the possible approaches using OpenCV, to reduce its scope.

3.1 Finding the door

3.1.1 Theoretical model

Finding the elevator door is not as easy as it might sound, it can occur in different places in the frame. Since the installation location of the camera is established, as shown previously in figure 3, the conclusion is that the door is on either the right side or the left side of the frame.

Initially, this seems like quite a big narrowing factor of the search area. However, if the camera changes position in the future, the usage of this approach renders the entire implementation useless. Hence, the focus is on finding the door in any circumstance. If the input data from the RealSense camera is analysed, the following data can be observed. Examples of RGB frames from a closed and an open elevator can be seen in figure 6 and figure 8 respectively.

Figure 6 Image closed elevator

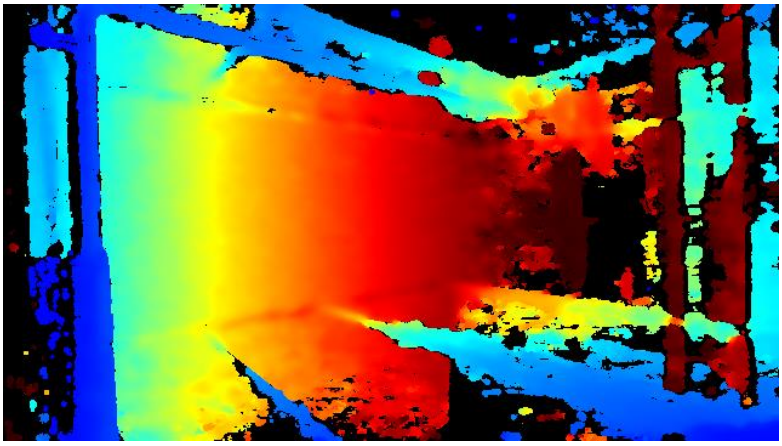

Figure 7 Image closed elevator depth

Examples of depth frames from a closed and an open can again be seen in figure 7 and figure 9 respectively. Every pixel in this frame corresponds to a point. Every point in the RGB frame carries the colour value of the corresponding pixel, while every point in a depth frame stands for the distance of that pixel to the camera.

Figure 8 Image open elevator

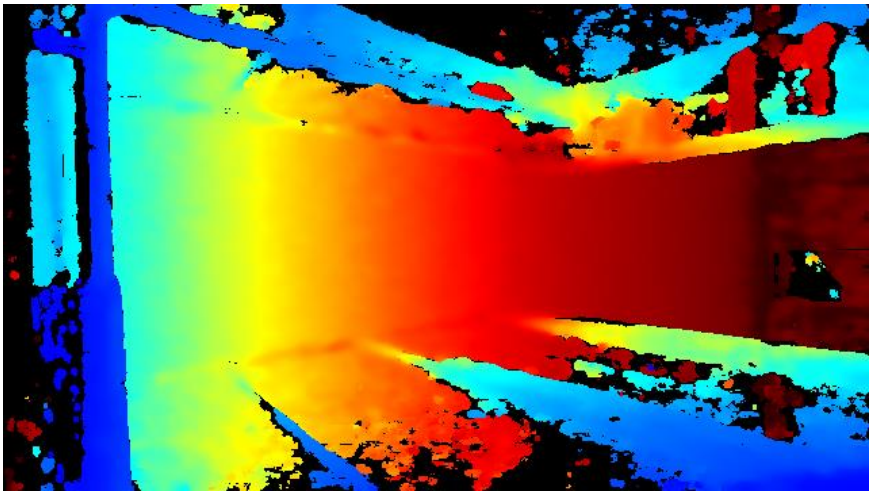

Figure 9 Image open elevator depth

The pictures show that the human eye can observe the door quite easily in both types of frames. So, a door can be observed in both an RGB as a depth frame. To indicate it concerns a door, recognition points are unconsciously used to undeniably suggest the type of object. A door is de facto a door, which is just common knowledge. The goal is to transfer this knowledge to the GARDEN application using computer vision.

3.1.1.1 Depth information

As said in the previous paragraph, the door is visible in both its closed and open state. Nevertheless, if the application only uses the depth frame from for example the closed elevator, it cannot pinpoint the door location.

A logical first step to try this, is stating that the panel furthest away from the camera would be the one where the door is located. However, as the position of the camera is variable, this solution cannot be used. Therefore, the location of the door cannot be found using just the one frame of a closed elevator.

If thereafter the depth frame of an open elevator is analysed, it can be noted that the outlining of the door is more prominent in this image, as shown in figure 10.

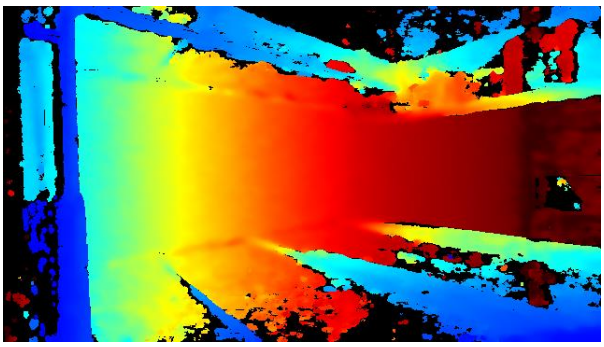

Figure 10 Image open elevator depth

The deepest part, shown in this image by a dark red colour, is everything outside of the elevator. Since the door is the only moving part in the elevator, the deepest points can be collected. Once all the points have been acquired, a bounding box can be drawn around them. According to a paper written for the International Workshop on Document Analysis Systems, a minimum bounding box is an enclosing box for a point set, with the smallest area, wherein all the points lie. [12] This bounding box then theoretically defines the door.

There is however still a problem with this approach. The deepest points are queried for this example, but how does the application define said points as “deepest points”? When is a point deeper than the original door location?

To acquire these results, the application can use both the depth frames from an open and a closed elevator. If the frames are compared to each other, a difference in depth is measured between them.

As an example, the theoretical points X, Y and Z are in both frames. The points X and Y are located on the location of the door, while point Z is located on the wall next to the door. If the application computes the difference of the frames, it notices a change in depth for X and Y, while the depth of point Z stays constant. This suggests that the first two points define the door.

The conclusion, summarised in a step-by-step plan is therefore:

- Load a frame of a closed door.
- Load a frame of an open door.
- Measure the difference in depth between the frames.
- Extract the points where a difference is observed.
- Create a bounding box for the extracted point set to extract the door.

3.1.1.2 RGB information

To achieve the same result with only RGB information available, the method used in the previous paragraph can also be of use here. The values of the points in an RGB frame represent the colour, so the return value of the method is a set of numbers representing the colour of the certain pixel.

If again the application measures the difference between the frames of an open and a closed elevator, only the points where the door is located changed colour, so they can be extracted.

A second approach using both frames is to imitate the human eye. If we look at a door, we see the object and its shape. We distinguish this shape and send it to our brain, which classifies the object as a door. If an application uses this logic, it must detect the shape of an object, which is defined by contours and edges. So, in order to achieve an outlining of a door, the application can use either frame.

To find a door in an image, the application must first find the edges of a surface or object. So in this case, it has to look for all the edges in the image. As an RGB frame does not really highlight edges sufficiently for extraction, the frame must be manipulated in order to easily extract the edges. The image must be converted to a sort of blueprint in which the edges stand out, so the application can analyse them. An example of this conversion is demonstrated in figure 11 and figure 12.

Figure 11 Closed elevator RGB frame

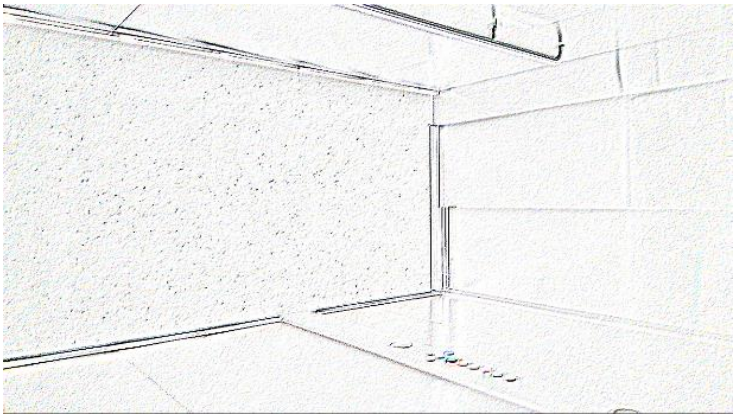

Figure 12 Closed elevator blueprint

When the edges are extracted, they can be used to gather the required information. The biggest panel is probably the floor. Panels which start big but get smaller are probably the walls, due to the perspective of the image. A pane which is divided in multiple smaller panes, is probably the door since it is made from different sliding pieces.

A final method to define the door, focuses on a very specific aspect of this element, movement. A door is normally the only element in an elevator which moves. This means that if the application analyses a video of the elevator and maps all the moving elements, the conclusion is that the sector in which the most movement is recorded, indicates the location of the door.

3.1.1.3 Elaboration with OpenCV

To elaborate these concepts, the OpenCV framework offers different methods and algorithms to tackle the problem step-by-step. The next chapters focus on the specific elaborations for each discussed method, coupled with the corresponding OpenCV methods, to achieve the expected result.

The methods are compared based on their theoretical capacity, produced values, accuracy and applications within GARDEN. The actual implementation of the chosen methods into the GARDEN pipeline is elaborated in the “Proof of concept”.

3.1.2 Intel RealSense Provider

Both methods to elaborate the door detection mainly rest on the use of the Intel RealSense camera. This camera provides the application with the depth and RGB frame of both the open and the closed door.

The first step the application must take to load both of these frames, uses already implemented code within the GARDEN project. The element within the project, responsible for this, is called the RealSenseImageProvider.

This provider is a class called upon in the GARDEN pipeline, which has the responsibility of connecting the RealSense D435 camera with the application. Once the camera is connected, the class requests the frames recorded by the device and sends them to the other GARDEN components in real time. A second source of input for this provider can be an already recorded stream of data, saved to a file.

3.1.2.1 Frame delivery

Because the camera transmits frames in real time, this can cause some minor issues which are important to understand the project. If the data would just be streamed, the frames would occur in the order they were generated by the camera. So, if the D435 records frames one, two and three, then it would only be normal that when the stream is received the frames are also in that exact order. This is then also how the provider delivers the frames to the GARDEN pipeline.

The camera provides these frames with a steady speed of 60 frames per second for depth information and 30 frames per second for RGB information. This deficit is resolved by the provider, by combining both streams in a delivery of 30 frames per second. This means that GARDEN receives a frame every 33.33 milliseconds.

GARDEN requests a new frame at the beginning of the pipeline. The next frame is requested when the pipeline is at its end and restarts. The special occurrence with this approach is, if the duration of the entire pipeline is longer than 33.33 milliseconds, that the frames generated by the camera during

this time are not used. This means that the provider does not keep track of missed frames, which is important to know for further approaches in calculations and research.

When a file is used as an input for the provider, GARDEN provides an option to use all the frames in said file. The drawback of this is that the frames are saved to the memory of the application, so they are allocated to the random-access memory (RAM). As the input files recorded by the RealSense camera are quite large, the buffer is mainly used for testing purposes. The upside of this approach is of course that all the frames can be processed.

3.1.2.2 Data delivery

The RealSenseProvider class converts a frame, received by the D435 camera, to a datatype that can be used within OpenCV. This datatype is a self-made class named Image. This class combines and saves all the data regarding the current frame. This can range from matrices to vectors of points and so on. The RealSenseProvider is the first class which adds data to the Image.

The RGB and depth values from the received frame are converted by the provider to an OpenCV-type named Mat. This type resembles a matrix. The matrix has a number of rows and columns, which represent the resolution of the incoming frame. This means that a frame recorded in the resolution 1280x720, is saved to a matrix with 1280 columns and 720 rows.

A matrix can also have a different number of channels. This is the case when the application saves RGB data, which has 3 channels. Concretely this means that every element of the matrix contains a vector of 3 values with the datatype ‘unsigned char’. These three values represent the RGB colour space, so each value in the vector is a number between 0 and 255 which indicates the colour strength.

The depth matrix has only one channel of data. The value of an element in this matrix is just a floating-point number of datatype ‘float’ and it represents the distance in meters from the camera to this pixel.

As a conclusion, it can be stated that each element of a matrix represents a pixel in the image, regardless of the datatype.

3.1.3 Static frames using depth information

To determine the location of the door using depth information, as previously stated, a distinction must be made between a frame of an open door and a frame of a closed door.

A problem with this approach is the high level of distortion within the frames and recordings. If two frames are compared wherein the elevator is in a 'closed' status, it can visually be noted that certain depth areas are divided differently. This difference can be explained by the D435 camera and its usage of lasers to enhance the depth information combined with the elevator having reflective surfaces. The camera also uses stereo images to define depth, but this does not entirely get rid of the noise.

An example of the noise can be seen in figures 13 and 14, figure 14 is the frame immediately after figure 13. Note the difference between the two frames, which is highlighted in figure 15.

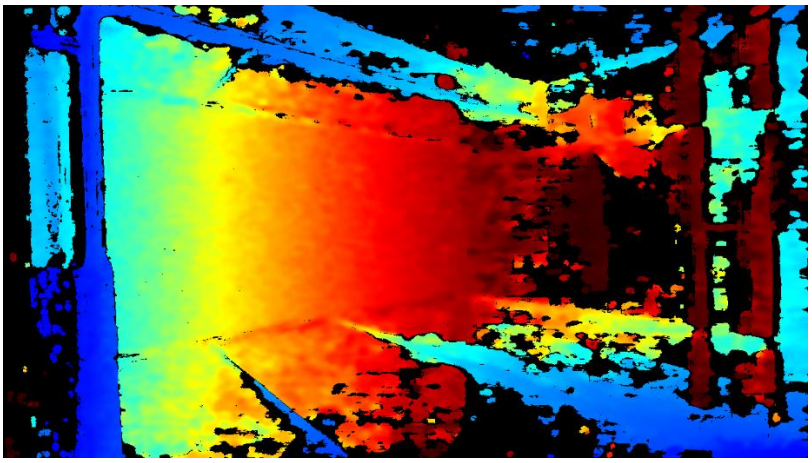

Figure 13 First depth frame

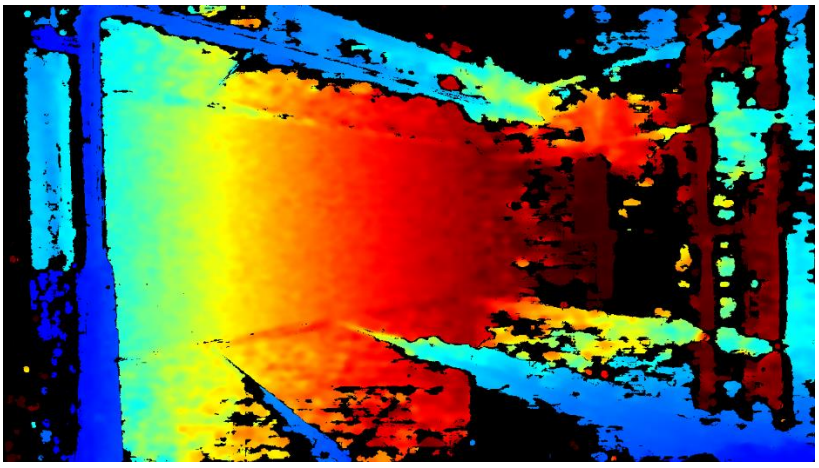

Figure 14 Second depth frame

As said in the theoretical model, the ideal way to find the door was subtracting a frame of a closed elevator with a frame of an open elevator to define the points where there is a difference in depth. Theoretically speaking, if two depth frames of a closed elevator are subtracted from each other, there should not be a measurable difference in depth. Unfortunately, due to the noisy data, there is a difference when the two images are subtracted from each other. This highlights many unnecessary and unwanted changes in depth.

In figure 15 those points are highlighted. A subtraction was made between the two images which were shown earlier, as an addition to this, only points where the difference was more than 0.5 meter are highlighted.

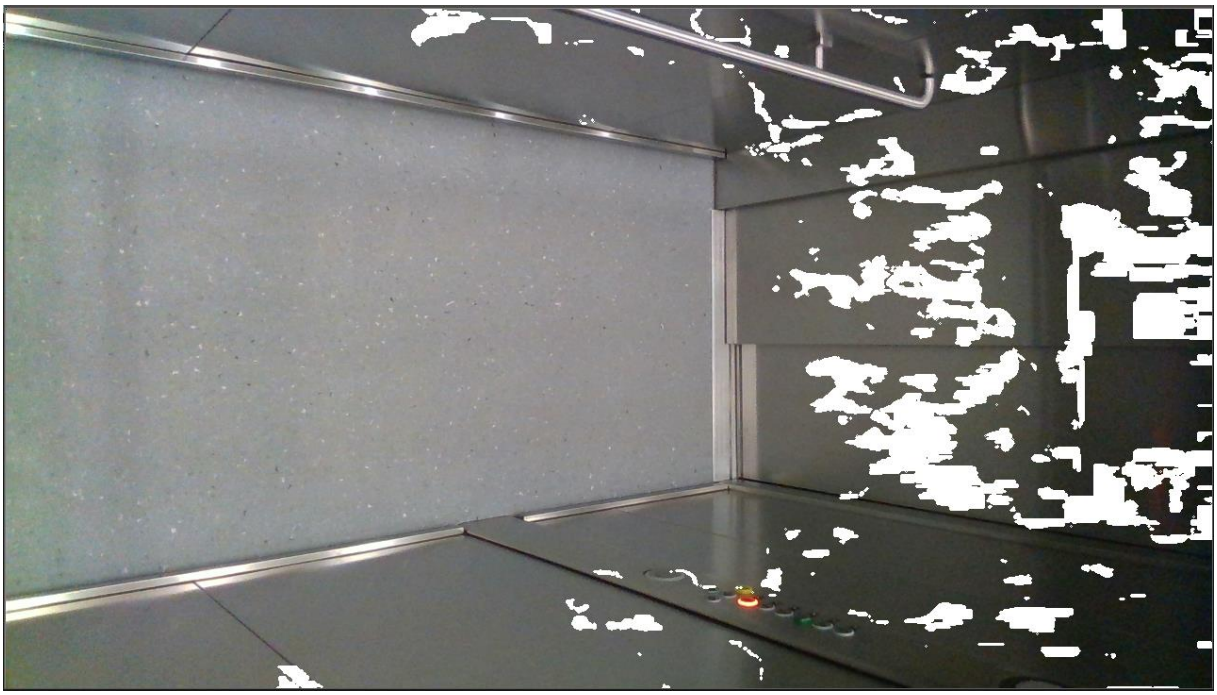

Figure 15 Depth difference

3.1.3.1 Image averaging

A solution for this problem is of course adding noise reduction filters like Gaussian Blur or general Image Averaging. [13] These algorithms are available to use in OpenCV and seem like the perfect solution. Unfortunately, these methods require a 3-channel RGB image or a 2-channel grayscale image and will not work with the one channel floating point matrix. However, this solution can be modified to meet the needed requirements.

The median filter also uses Image Averaging and works as follows. It requests the pixel value of 1 channel of the RGB image. Now it also requests the values for that channel of the surrounding pixels. Using all these pixels, the algorithm calculates the median pixel value for that channel and sets that value as the new one for the requested pixel. [14]

This theory can also be applied to the depth matrix. The pixels where the depth is unstable or unknown, can be filled with the median values of the surrounding pixels. However, if the input data is examined again, it can be stated that the fluctuations in the depth data have a surface which is too large. So, using the surrounding pixels to cancel out noise is not desirable, as these surrounding pixels also have the same inconsistency issues.

What can be stated if the depth images are considered again, is that certain areas in for example the first frame have depth information, but this depth information is missing in the next frame.

Due to this, it can be stated that calculating the median not for the surrounding pixels but calculating it for the same pixel over a range of frames, is the solution to reduce noise in the depth data.

If a series of frames is used as input for both statuses (open and closed), then a median image can be calculated from this range of frames wherein the noise is significantly reduced. Once this image is calculated for both states, the difference can be computed. The median image can be seen in figure 16.

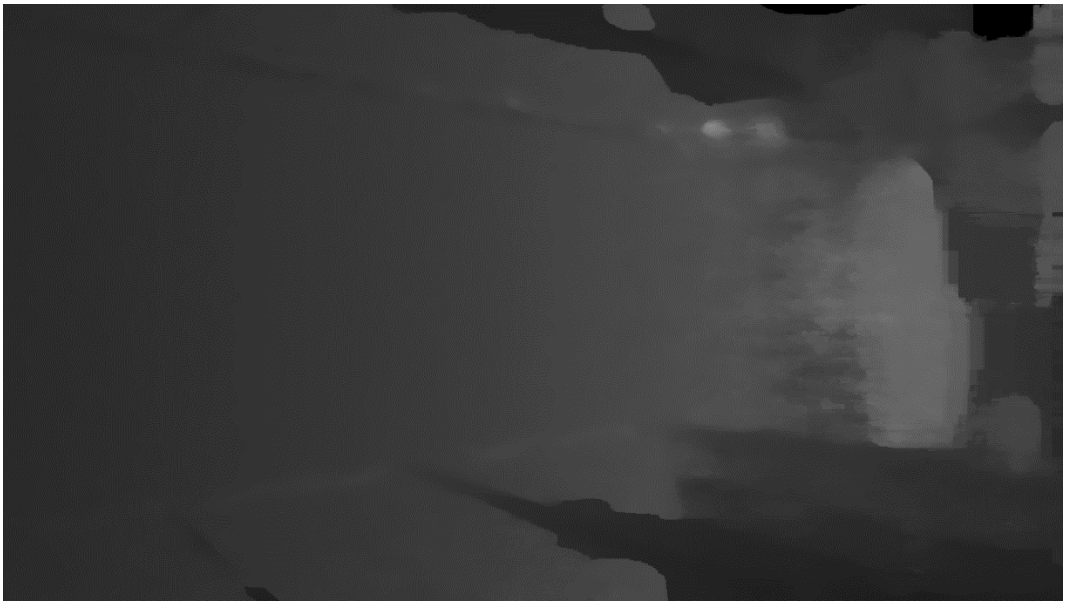

Figure 16 Median depth image

3.1.3.2 Optimising the calculation of difference in frames

The input frames have a resolution of 720 by 1280 pixels. This means that each frame consists of 921 900 separate pixels. Comparing the depth value of each pixel creates a lot of unnecessary overhead in the application. Regarding performance, it is an improvement to divide the image in parts, squares for example. Using a built-in OpenCV-algorithm, an average depth for this square can be calculated in a very performant way. This lets the application compare the average depths of each separate square instead of each separate pixel, which theoretically improves runtime drastically. If for example the frame is divided by squares of 25 by 25 pixels, it only has to compare 1 475 elements instead of 921 600. The result can experience some minor deflections, but this is preferable compared to a long runtime.

To support this statement, a comparison is made between the two mentioned methods. In the first instance, the method compares the depth of each point with the value 1. If the depth is higher, a white circle is drawn on that point. In the second instance, the frame is divided in squares of 25x25 pixels. Subsequently, the average depth of all the points in a square is calculated and compared with the value 1. Again, if it is higher, the square is drawn in white.

In the table underneath, the results of this test are represented.

Table 1 Difference computing depth

	Average time in milliseconds (ms)
Points	414,965 ms
Squares	7,495 ms

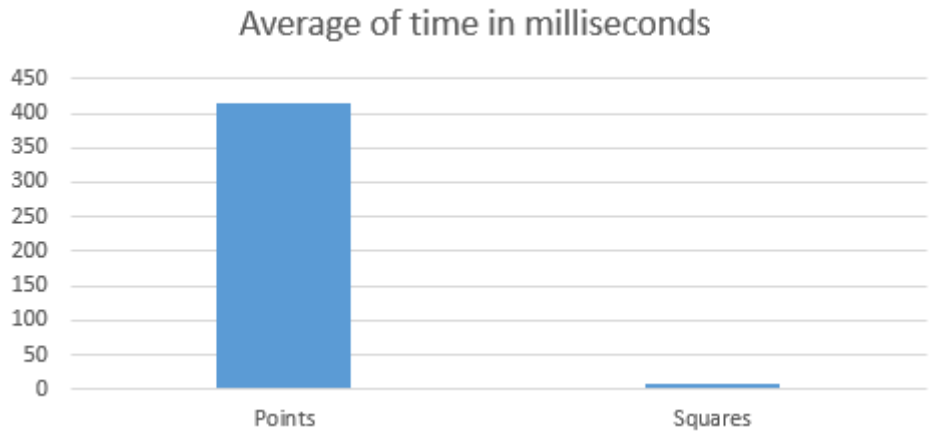

Figure 17 Difference computing depth graph

As seen in the results, the usage of the squares is significantly faster than the usage of the individual points, so this method is the preferred approach.

To define the door with this method, all squares where the depth was significantly different compared to another frame can be requested and joined into one bounding rectangle.

3.1.4 Static frames using RGB information

To define the location of the door using RGB information, the focus must be laid on algorithms which depend on optical recognition in images. The usage of such algorithms is discussed in this chapter.

3.1.4.1 RGB information

As previously cited the application can use the same algorithm and method used for detecting the door with depth information, with RGB information. More specifically, comparing an RGB frame of a closed and an open elevator and calculating the difference between them. The RGB values which differ the most, are noted as a result.

The advantage of using RGB values is of course the lack of noise compared to the depth values. That is, however, what is noticeable to the human eye if two images of a closed elevator are compared. Nevertheless, if a comparison in RGB values is made with these two frames, the results are not what is expected. If the area's wherein the RGB information is significantly different are highlighted, the following result is obtained as shown in figure 18.

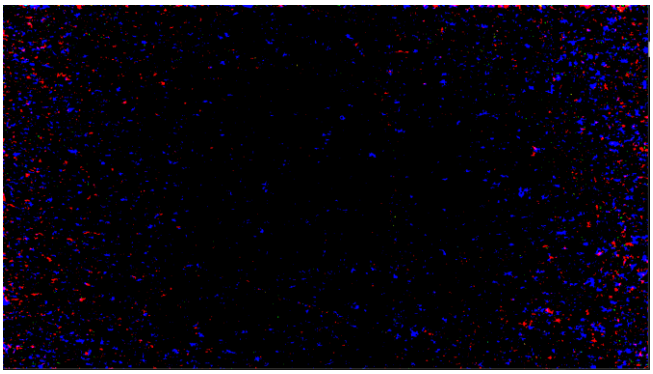

Figure 18 RGB difference closed elevator

If a comparison is made between an open and a closed elevator, using RGB images, the result is significantly different, as shown in figure 19.

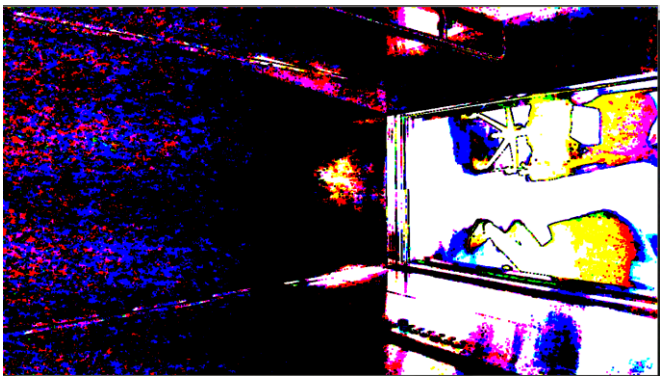

Figure 19 RGB difference open elevator

However, this result is far from optimal because due to reflection, there are also significant changes in the areas surrounding the doorway. An extra challenge using this technique is that the lighting in each situation or on each floor behaves differently, which causes additional noise. Because of all these reasons, it can be stated that the depth information is more stable than the RGB information in order to determine the location of the door.

3.2 Edge detection in RGB images

If humans see a door on a picture, our brain normally immediately classifies it as a 'door' and recognises the object. If that same door is drawn by an artist with only lines on a paper, our brain is still able to recognise the drawn object as a door. The brain uses these lines to visualise the objects. More information about this subject can be found in the paper written by Bilge Sayim and Patrick Cavanagh [15]. An example of recognising a door in a drawing is demonstrated in figure 20.

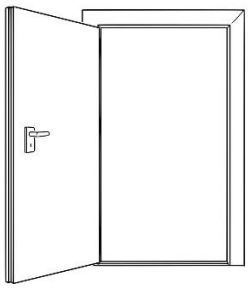

Figure 20 Drawing door

If the brain can detect an object using lines, an application should be able to do the same by using existing computer vision algorithms provided by OpenCV.

3.2.1 Canny algorithm

Canny edge detection is a technique to extract structural information from different vision objects, in order to simplify the analysis of images by drastically reducing the amount of data to be processed, according to John Canny [16].

To summarise the paper, it can be stated that the algorithm uses an input image, finds the intensity grades of this image while applying a non-maximum suppression in combination with hysteresis, to receive the strong edges in that image. An example of this is elaborated in a paper by the Indian Institute of Technology [17].

The canny algorithm is available as a method within the OpenCV framework [18]. According to its documentation and the official tutorials by OpenCV, the usage of the algorithm is not completely as described by John Canny [16].

3.2.1.1 Gaussian blur

In order to successfully use the OpenCV implementation, the image must first be blurred using the Gaussian filter. As described in the paper of mister Canny, this is part of his algorithm. In OpenCV, an additional method must be used, since the Gaussian filter is not part of the Canny method. The usage of this additional method is recommended in an OpenCV tutorial. [19]

The Gaussian filter (or Gaussian blur) is a type of image-blurring and noise reduction filter that uses a Gaussian function. This function expresses a normal distribution in statistics and is used for calculating the transformation to apply to each pixel in the image, as defined by Wikipedia. [20]
The in-depth implementation and working of this filter is documented by Elhanan Elboher and Michael Werman [21].

3.2.1.2 Grayscale input

To receive the best possible result for the edge detection, OpenCV themselves recommend using a grayscale image as input for the Gaussian blur method [19]. Converting a RGB image to grayscale in OpenCV is done using the function 'cvtColor', with the parameter 'COLOR_BGR2GRAY' [22]. The result of this method is an OpenCV matrix, which defines the grayscale image.

3.2.1.3 Canny edge detection.

The flow of the edge detection is thus as follows. First the RGB image provided by GARDEN must be requested. This RGB image is converted to a grayscale image, which is blurred using the Gaussian filter. The result of the filter is used as input for the Canny algorithm.

An example of this entire workflow is shown in figures 21 and 22. Figure 21 is the input image, while figure 22 is the result after the Canny algorithm.

Figure 21 Input image closed elevator

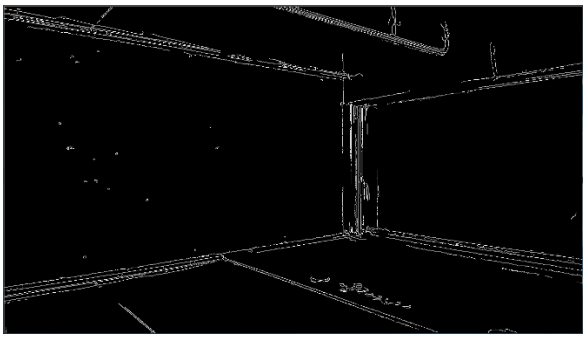

Figure 22 Input image after Canny algorithm

3.2.2 Hough transform

As defined by Wikipedia, the Hough transform is a feature extraction technique used in image analysis and computer vision. The purpose of the technique is to find lines in an image. Later this was expanded with the detection of arbitrary shapes. [23]

So, what Hough transform does is detecting lines in images and extracting them. Like Canny, Hough is also part of the OpenCV framework, this means the algorithm can be used within the GARDEN project.

According to the documentation [24], OpenCV uses two different implementations of Hough, the standard Hough transform and the probabilistic Hough transform. Both algorithms have some similarities, like how they represent the found lines.

If a line is found, it is presented connected to the origin (x: 0, y: 0) of the image. The line is expressed using the Polar coordinate system, which uses parameters ρ (rho) and θ (theta). Hence, a line equation can be written as follows according to the documentation [24].

$$y = \left(-\frac{\cos(\theta)}{\sin(\theta)} \right) * x + \left(\frac{\rho}{\sin(\theta)} \right)$$

The origin of the image is always in the left-hand upper corner in the OpenCV coordinate system, which means lines are displayed as shown in figure 23. More information about the coordinate system can be found in the documentation [25].

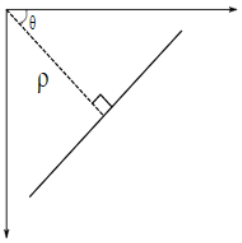

Figure 23 Hough coordinate system display

OpenCV uses the polar coordinate system [26], this system has its origin in the left-hand upper corner. Since this coordinate system is different than the Cartesian system [27], which is more commonly used in physics and mathematics, this will influence the way the program handles the coordinates.

3.2.2.1 Standard Hough transform

According to the OpenCV documentation [24] [25], using the standard Hough transform, the result of this algorithm is a vector (collection) of the ρ and θ values. Rho is measured in pixels, while theta is measured in radians. An important parameter in this OpenCV algorithm is the 'threshold', which defines the number of 'votes' a line must have before it is returned as a detected line. In the example [25] the standard Hough transform is applied on a Sudoku game, they achieve the following result as displayed in figure 24. In OpenCV this algorithm can be used with the function 'HoughLines'. [18]

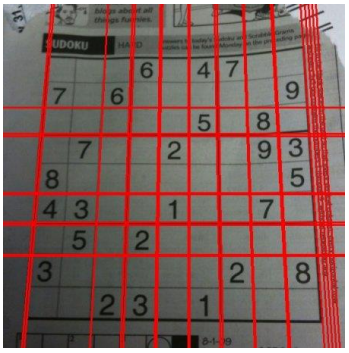

Figure 24 Hough standard example

3.2.2.2 Probabilistic Hough transform

As defined by J. Matas the probabilistic Hough transform is a more efficient implementation of the Hough transform algorithm. It gives only the extremes of the detected lines. It does not use all the points, but only a random subset which is sufficient for line detection [28]. The working of the probabilistic approach is documented on the OpenCV website [24] [25]. The most important parameter for this algorithm is 'maxLineGap' which defines the maximum allowed gap between line segments, to treat them as a single line. In OpenCV this algorithm can be used with the function 'HoughLinesP'. [18]

In the example in the OpenCV documentation [25], the same Sudoku game is used in combination with the probabilistic version of Hough. The result can be seen in figure 25.

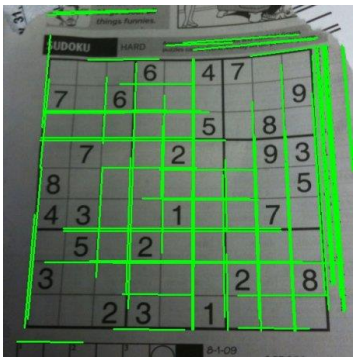

Figure 25 Hough probabilistic example

3.2.2.3 Conclusion

If the results of both algorithms are compared, it can be noted that the line detection using the probabilistic approach is 'cleaner'. There is less overhead with this approach, the lines do not continue over the entire frame. However, this algorithm has, in comparison to the standard approach, a lot more 'open spots', where no lines are detected.

Since lines in an elevator need to be detected, the algorithm cannot afford to have blank spaces in the found lines. Otherwise the panel which must be detected will have gaps. These gaps can of course be filled accordingly, but this creates overhead again.

3.3 Finding the floor

In the previous chapter, the possibilities to detect edges in images are discussed. Once these edges are available, they can be used by an algorithm in order to detect shapes. One of the shapes to be detected is the floor of an elevator. The detection of the floor is an important part of the GARDEN project, since this area can be used in order to generate a 3D model of the elevator.

To achieve this, the lines in an image of the elevator must be detected using the Hough algorithms. The expected input is displayed in figure 26 while the expected detected area is highlighted in figure 27.

Figure 26 Floor detection input image

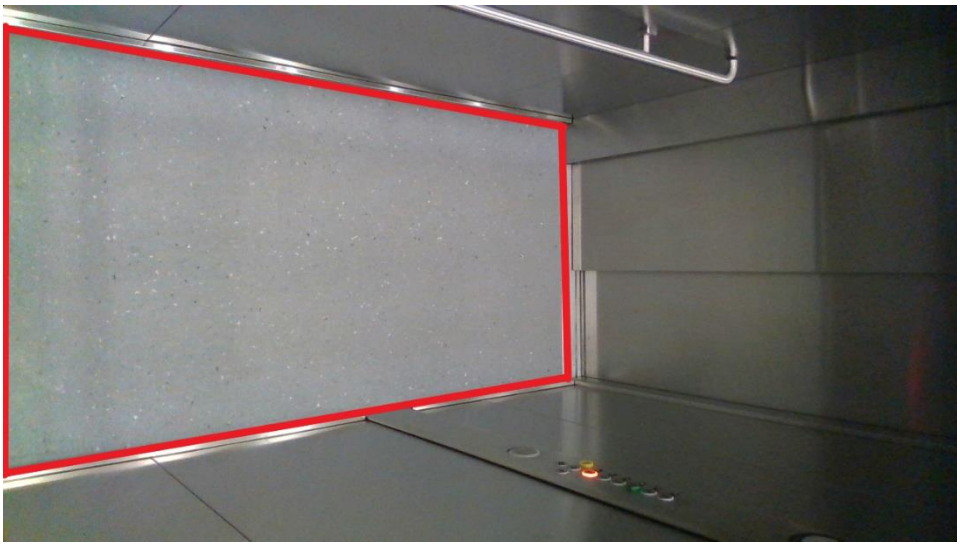

Figure 27 Floor detection expected result

3.4 Resolving the door status

In order to correctly resolve the status of the door, the application must first know where the door is located. A method to approach this was discussed earlier, so the following chapter is written using the fact that the door location is already known.

In this chapter, the focus is laid on detecting the status of the door combined with resolving the issues the application must overcome in order to correctly classify the status of the object, in various scenarios. The following questions are answered in this chapter.

- Which computer vision methods or connected logical steps must be taken to define a door status like ‘open’ or ‘closed’?
- How can computer vision express the status of a door in a percentage defining how ‘open’ or ‘closed’ a door is?
- How can existing algorithms within GARDEN be used to filter out noise from persons?
- How can movement of objects assist the classification of a door status?
- Can optical recognition algorithms be used to detect a door status?

3.4.1 Interpreting depth information

The application must be able to detect whether a door is open or closed. Since GARDEN is connected to the Intel RealSense D435 camera, it has access to the recorded depth information. This information can be proven useful to detect the status of a door.

3.4.2 Depth information to object status

In figure 28, a depth frame of a closed elevator can be observed. In figure 29, a frame of the same elevator can be seen. In both frames, a specific point at the location of the door is highlighted.

Figure 28 Closed elevator with detection point

Figure 29 Open elevator with detection point

The depth data for this point can be requested, in order to define a difference between the two frames. In the 'closed' state of the elevator the point has a depth of about 2 metres, while in the 'open' state it has a depth of about 5 metres. Because a difference in depth can be noted when comparing the frames, this can be translated into the application using OpenCV. As previously stated, the depth image is provided by GARDEN as the OpenCV-type 'Mat'. According to the documentation, this type within the framework gives the application the possibility to request a specific value for a point in the frame [29] and thus request the depth information for the specific point.

To conclude, the application can therefore determine if the door is open or closed by analysing the depth information. If the depth of a defined point is closer, the door is probably closed compared to open when it's further.

3.4.2.1 Fixed states

The state of the door can be determined using the difference in depth of a certain point located on the door. Unfortunately, this approach has a few drawbacks.
A first drawback is of course the situation wherein an elevator user walks through the door and pollutes the depth data of the specific detection point. More on this issue later.

A second drawback is that the approach of using a fixed-point, results in only being able to detect the states 'open' or 'closed'. Once the movement of the door passes the specific point, it will immediately change its state. Depending on where the point is located on the door, it will always give false information for one of the states. In addition to this, there can be distorted depth data in the recording. If this distortion is located on the point used to define the state, this creates false readouts.

3.4.2.2 Dynamic percentage

Using only one point or area to detect the door state results in only knowing if the door is either open or closed. This approach cannot be used to determine how far the door is opened or closed. To implement a solution for this using the pixel values, a few detection points can be set up over the width of the detected door. If for example 10 detection points are set up, the door status can be expressed as a percentage. If 7 detection areas detect a closed door and 3 detect an open door, it can be concluded that the door is only open for 30%. An example of such a detection area can be seen in figure 30. A problem with this approach is again the possible noise in the detection points and persons distorting the depth data.

Figure 30 Multiple detection points

3.4.3 Depth averaging

A solution to make the classification of the door status more dynamic is enlarging the detection area. If the application uses the entire location of the door for its calculations, the error rate reduces because a lot more depth data is used for the classification. Combining this with the previous algorithm means keeping track of all the points at the door location and comparing their depth between each other, in order to classify the door as 'open' or 'closed'.

However, as mentioned earlier, comparing each point to new values every frame requires a lot of computation power which causes a lot of delay in the calculations. Dividing the door area in squares is also a possible solution for this problem. But as the area which is checked by this algorithm consists mainly of the door, this means that the entire door's depth will change accordingly. There are almost no parts of this detection area which do not represent the depth of the door. In the previous approach, the detection area was split into rectangles of which the mean depth was calculated. But because the detection area in this case is the door itself, calculating the mean of this area suffices.

To test this method, the door is manually defined and extracted as a detection area as seen by the highlighted area in figure 31.

Figure 31 Detection area door

To calculate the average depth of the selected area, the OpenCV-algorithm 'mean' is used. According to the documentation [30], this algorithm requests a matrix and calculates the average value of all the points inside. It calculates the average separately for each channel and returns a one-channel vector (OpenCV-type 'Scalar') with the average values inside. Because a depth matrix only has one channel, the first element of the vector is the requested result. Using this function on the three states of the elevator, the following values are returned.

Table 2 Average depth values calculated by cv::mean() method

	Average depth value
Door fully closed	0.62
Door fully open	0.75
Door halfway open	0.68

As seen in the results, the average depth values can represent the current state of the door. Using this approach, the status of the door can also be expressed as a percentage. This means the application is no longer bounded to two states 'open' and 'closed' but can calculate that the door is 80% 'open'. This can be achieved by setting the average depth value for the 'open' and 'closed' states as upper and lower boundaries. To calculate the percentage, the value of the current mean is compared to the set boundaries and expressed as a percentage. This formula can be used to achieve the required goal.

3.4.4 Blocked visibility

A recurring issue in the previously discussed methods to detect the status of a door is the distortion or pollution of depth data by persons entering or leaving the elevator. If a person is standing between the camera and the door, there is no possibility to track the part of the door which is behind that person. This does not mean, however, that the application cannot determine the door status.

Calculating the status of the door using of the 'mean' algorithm works as expected in an empty elevator, but this can change if there are people inside. Because there is an object closer to the camera now, the average depth information is less deep than the average actual depth of a closed elevator. Using the same set up and method of calculation as in point 3.4.3, the average depth of the closed elevator is again 0.62, while the average depth of the same closed door with a person blocking the view is 0.51.

In the case of a closed elevator, this would mean that the average depth is smaller than the defined 'closed' border, which can be easily cancelled out. However, when the door is open, this distortion in the data can suggest the door is closed because the average depth information suggests so.

Therefore, it is of great importance that the distorted depth data is cancelled out. A preferred approach is extracting the parts of the frame where the person is standing from the calculations. A first step for this is finding the person in the frame and drawing a line around them.

Within GARDEN, there is an already implemented functionality for this matter. Using the contours algorithm of OpenCV, GARDEN can detect persons in an elevator and draw a containing polygon around them. This is documented in the architecture description [31] and the bachelor thesis of Cornelia Lederhilger [32]. For the scope of this project, the focus is put on the available information returned by this method.

According to the documentation, the method returns a set of points which defines the contours in the entire frame. An example of a contour can be seen in figure 32.

Figure 32 Example extracted contour GARDEN

If the application then requests these contours and subtracts them from the door detection area, the noise of the persons is removed. The average depth can then be calculated using the door detection area minus the overlapping persons. To put this received result into perspective, the contours must also be subtracted from the upper and lower boundaries for the depth, in order to determine the exact 'open' or 'closed' percentage. The elaboration of this is handled in the proof of concept later.

3.4.5 Optical recognition

3.4.5.1 Theoretical approach

According to Nazanin Hashemi there is a method called template matching, which searches for similarities between objects in images using certain mathematical algorithms. [33] The OpenCV framework defines template matching as "A method for searching and finding the location of a template image in a larger image". [34] Template matching essentially uses a predefined image and tries to match that one to a new input image.

An example of this is given by OpenCV in a tutorial [35] and can be seen in figure 33. The image of the dog's head is the template to be found, while the image with all the dogs is the input data. The result of a template matching method is then the location of the found template in the input image.

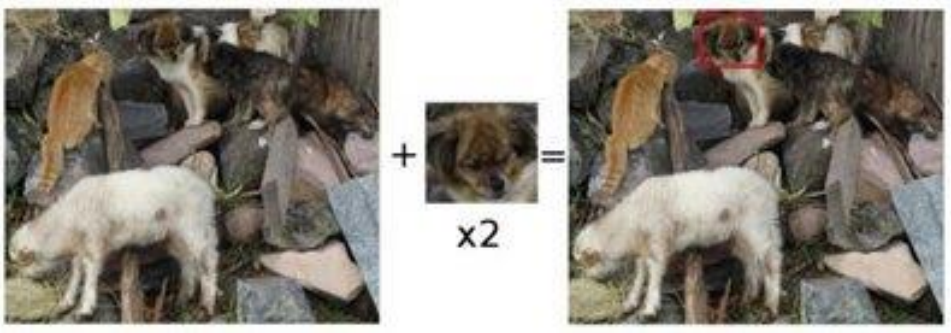

Figure 33 Template matching example

Using the OpenCV method 'matchTemplate' this result can be achieved. According to the documentation [36] the input for this method is the source image converted to a matrix and the template image converted to a matrix.

3.4.5.2 Functional test

To test the feasibility of this method, a picture of a closed elevator floor was loaded into the application as a template to search for. Using the 'matchTemplate' method and a recorded video stream of the elevator, the accuracy of this method was tested.

When the elevator is empty and the door was closed, the door was easily found by the OpenCV method, as shown in figure 34.

Figure 34 Template matching door closed

However, when the door is open, the method still returned a detected result area but now it detects the floor as the result, as shown in figure 35. A probable reason for this would be that the used input image for template matching was a closed elevator door, which is reflective and thus reflects the floor. The same problem is present when people are inside the elevator, as seen in figure 36.

Figure 35 Template matching door open

Figure 36 Template matching door people inside

3.4.6 Movement detection

To assist door status classification, movement can also be used. If a door is opening for a moment after which there is no movement anymore, it suggests that the door is open. The same can be said for the closing motion.

To detect movement in images the concept of optical flow can be used. Wikipedia defines optical flow as "[…] the pattern of apparent motion of objects, surfaces and edges in a visual scene caused by relative motion between an observer and a scene". [37]

An algorithm to detect such an optical flow in computer vision is researched by Bruce D. Lucas and Takeo Kanade. Their approach is a low-cost image registration technique to find an optical flow between two images, as described in their paper. [38] The method and algorithms defined by Lucas and Kanade are transferred into the OpenCV framework. The documentation of the framework states "[The function 'calcOpticalFlowPyrLK'] Calculates an optical flow for a sparse feature set using the iterative Lucas-Kanade method with pyramids". [39]

According to the OpenCV documentation [39] and a tutorial by the creators of the framework [40] the function needs certain parameters before it can be executed. Since optical flow is calculated between a pair of frames, there are two frames needed to execute the algorithm. However, a detection of features to track must also be executed for both frames. These features are the anchor points which the Lucas-Kanade algorithm uses to calculate optical flow between frames.

As suggested by OpenCV, these features or detection points can be obtained using the function 'goodFeaturesToTrack'. This method is defined in the documentation [18] and uses the findings of Shi and Tomasi in their article 'Good features to track' written for the 'Computer Society Conference' [41]. Once the features are acquired, they can be used for the Lucas-Kanade algorithm, which returns a vector containing a collection of points which define the movement between the corresponding detected features in the pair of frames. This suggests that if a detection point in frame one moved to another location in frame two, the returned collection of points will determine the direction of movement for that pair of features. A visual example of the return value of the Lucas-Kanade method can be seen in figure 37. [42]

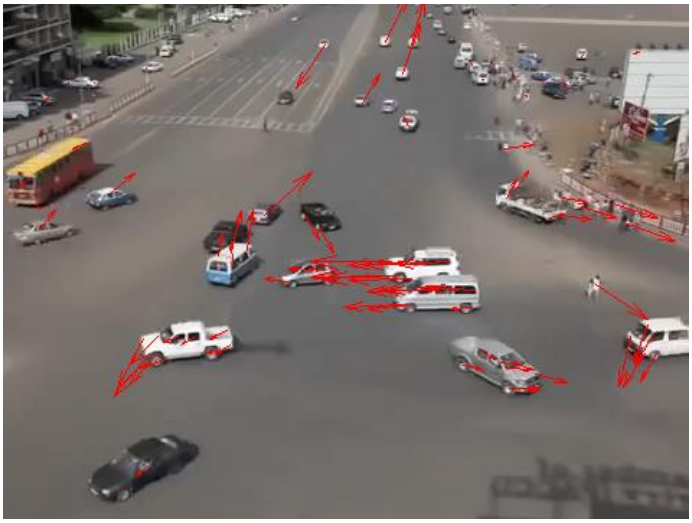

Figure 37 Example Lucas-Kanade

3.5 Elevator levelling

3.5.1 Purpose

To guarantee the safety of elevator users, several ISO-guidelines are composed which elevators must adhere to. These guidelines determine for example the maximum speed or stopping force an elevator can have. One of those guidelines also defines the margin wherein an elevator has to level out with the floor of the outlying building or shaft. This specific guideline is noted by the European Committee for Standardization in the finalised version of document prEN 81-20, which defines the standards for elevators which transfer goods or persons.

Chapter 5.12.1.1.5 states the following. “The stopping accuracy of the car shall be ± 10 mm and a levelling accuracy of ± 20 mm shall be maintained. If, during e.g. loading and unloading phases, the value of 20 mm is exceeded, it shall be corrected.” [43]

If an elevator does not meet these requirements, it can endanger the elevator users. If there is a difference of 2 centimetres between the floor of the elevator and the floor of the outside, a user can stumble over this hump.

In addition to this, since it concerns a fixed guideline, the elevator must adhere to this and is designed to do so. If the elevator fails to minimise this gap, it can suggest a mechanical failure with the elevator itself, which has to be brought to the attention of mechanics as fast as possible.

In summary, the following accuracies must be detected.

- The stopping accuracy of the elevator car cannot deviate more than 10mm
- The levelling accuracy of the elevator car cannot deviate more than 20mm

3.5.2 Technical capabilities and limitations Intel RealSense camera

As previously said, GARDEN is connected to an Intel RealSense D435 camera, which records both RGB and depth information. This camera also has its limits of course.

The maximum resolution this camera can record for depth information is 1280x720 pixels at a speed of maximum 90 frames per second and with a field of view of 85.2 degrees horizontal and 58 degrees vertical. The minimum distance at which the camera can record depth is 0.11 metre. The maximum depth it can record is 10 metres. For the RGB sensor the video can be streamed at a resolution of 1080x1920 pixels and at a framerate of 30 frames per second with a field of view of 69.4 by 42.5 degrees. [6]

According to the documentation from Intel themselves about the camera, a well-calibrated copy should have a maximum theoretical deviation of 0.1 millimetre regarding the depth root mean square (RMS). [44]

For the application of this technology within the GARDEN project, this is the perfect deviation. As the minimum difference to be detected is 10 millimetres, it seems there is enough breathing room for error to notice such a difference.

Unfortunately, this is only the perfect theoretical model. The documentation continues the statements with the following equation. [44]

$$Depth\ RMS\ error(mm) = \frac{Distance(mm)^2 * Subpixel}{focal\ length(pixels) * Baseline(mm)}$$

$$where\ focal\ length(pixels) = \frac{1}{2} * \frac{Xres(pixels)}{\tan(\frac{HFOV}{2})}$$

For the D435 camera, the following parameters are used in the equation. 'HFOV' is set to 65 (degrees), 'Xres´ is set to 1280 (pixels), the 'Subpixel' is 0.08 and the 'Baseline' is 50 (millimetres).

The result of this calculation is shown the following graph, displayed in figure 38. [44]

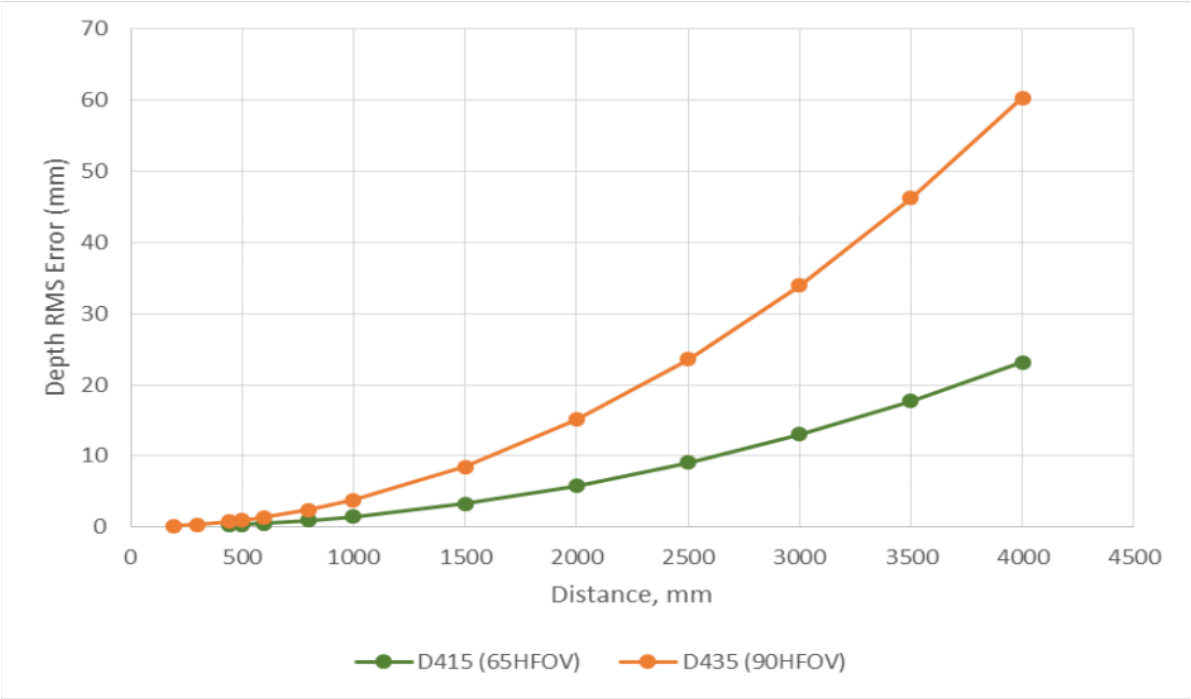

Figure 38 Graph RMS error D435

From this graph it can therefore be derived that the depth increases exponentially compared to the distance. As the D435 camera is used within the EDEN project, the values are represented by the orange line.

3.5.3 Measuring levelling accuracy

The deviations considered, the theoretical possibility to detect a height difference of 10 or 20 millimetres can be looked at. As shown previously, this mostly depends on the position of the camera inside the elevator.

If the camera is placed as described before, namely in the most outer corner across the door, then this suggests the following theoretical results. For the elevator which is used to record test scenarios for the EDEN project, it means that the real distance from the camera to the floor outside of the elevator is around 4 to 4.5 metres. The fault tolerance of the D435 camera at this distance exceeds 60 millimetres, which suggests that the small height difference of 10 or 20 millimetres is not detectable from this position.

The project partner View Promotion GmbH has, as a possible follow-up and possible new permanent camera position, suggested a second camera position in the elevator. The camera is placed on either side wall of the elevator in the upper border, right across the control panel.

Figure 39 New camera position

In figure 39, the old camera position is highlighted in the black colour. The new position is shown by the red colour. In figure 40 the camera view from the new position is demonstrated.

From this position the door is still partially visible while the distance to the floor outside is also smaller. This distance is now around 2 to 2.5 metres. Hereby the fluctuation in RMS error for the depth narrows down from 15 millimetres to 25 millimetres. With these theoretical values the GARDEN application can in theory check the partial compliance with the suggested ISO norms.

Figure 40 New camera position RGB image

The entire situation is shown in the figures below. In figure 41 you can see the normal situation for the elevator. In figure 42 a levelling error is recreated. The new camera position is highlighted with number 2 in both figures. The camera at position 1 is the old location.

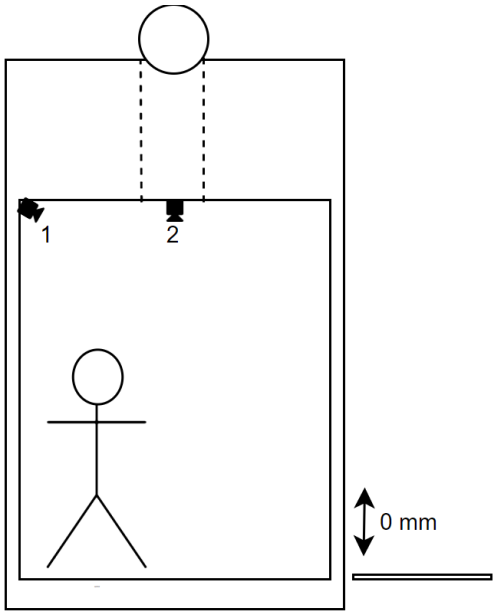

Figure 41 Normal elevator levelling

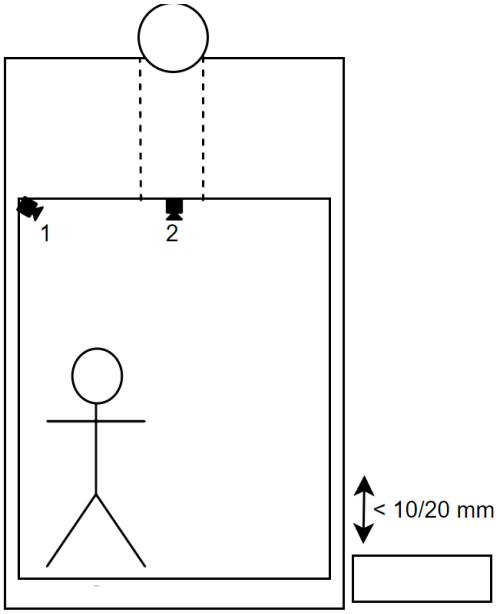

Figure 42 Incorrect elevator levelling

3.5.4 Disparity shift

There are certain functions within the Intel RealSense Software Development Kit (SDK) which are responsible for enhancing the received depth information. One of those functions is changing the 'disparity shift'. To put this functionality into perspective, the coordinate system should be considered. This system has three axes namely X, Y and Z. The Z-axis resembles the depth.

As previously stated, the camera has an RMS deviation from a certain distance. To reduce this deviation, the application can apply the 'disparity shift'. According to Anders Grunnet-Jepsen the disparity shift changes the maximum and minimum values of the Z-axis. These borders are called 'MaxZ' and 'MinZ'. By applying the 'disparity shift', both values change simultaneously. By changing both values using this functionality, the theoretical accuracy of the depth information should be improved within a certain range. [44] [45]

To test if the disparity shift has actual influence on the precision of the depth data, a test scenario is created to measure this in controlled conditions. The test setup is as follows.
A square object is placed on exactly 2.5 metres from the camera, in a perfectly straight line. During the different tests, the object or the camera are not moved. A visual representation of the setup is shown in figure 43.

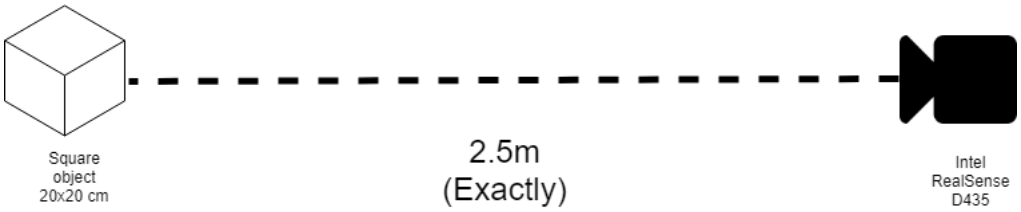

Figure 43 Disparity shift test

The Intel RealSense D435 camera is configured in all scenarios with a baseline of 50 millimetres, a resolution of 1280x720 and a horizontal field of view of 90 degrees.

The purpose of the test is to measure the accuracy of the depth information in different scenarios. For every frame, the depth value of 10 pixels on the square are recorded. In every following frame, the depth values of the same pixels are used. From these 10 depth values in every frame, the median is calculated and noted, to reduce noise and outliers. Per test scenario, 25 frames were recorded, which combines to 250 values being analysed for each scenario.

From the found medians in each test case, the collective median, the standard deviation and the variance are calculated to compare with each other.

In the first test, the distance to the object is exactly 2.5 metres and the setting 'disparity shift' on the RealSense camera is set to the value '7'. In the second test the distance is the same, but the disparity shift is set to '0'. This means that the shift will not be applied during runtime, so the 'MinZ' is set to the lowest possible value and the 'MaxZ' is set to infinity.

The choice to use value '7' for the 'disparity shift' was made using the equations, found in a paper written at the Sensors conference in Basel [46]. These equations are the following.

$$MaxZ = focal\ length\ (pixels) * \frac{baseline\ (mm)}{disparity\_shift}$$

$$MinZ = focal\ length\ (pixels) * \frac{baseline(mm)}{disparity\_shift + 126}$$

$$where\ focal\ length\ (pixels) = \frac{1}{2} * \frac{Xres(pixels)}{\tan(\frac{HFOV}{2})}$$

These formulas [46] are used with the following values regarding the D435 camera. The 'Xres' is set to 1280, the 'HFOV' is set to 90 and 'baseline' is set to 50.

For a 'disparity shift' of value '7' this results in a 'MinZ' of 0.24 metres and a 'MaxZ' of 4.57 metres. If a higher 'disparity shift' is used, the 'MaxZ' drops significantly. However, since there is a lot of reflection in the elevator, increasing the shift above '7' creates more noise and incalculable depth. Hence why this value is the upper limit for this test.

The tests produce the following results.

Table 3 Disparity shift test results

	Disparity shift = 7	Disparity shift = OFF
Median	2.47 m	2.49 m
Standard deviation	0.0263	0.0379
Variance	0.0007	0.0014

What this table tells us is that, on average, the 'disparity shift' does not really influence the final result. The average result with 'disparity shift' is even more divergent from the actual distance, compared to the measurements without the shift. In general, these results must be taken into another context. As previously shown, there is also an RMS deviation at this distance.
The actual difference between the two measurements is about 20 millimetres, but the RMS deviation at this distance is around 15 millimetres with these settings. So the results should not be narrowed down to only the distance value.

Therefore, the standard deviation is also an important value in this test setup. The standard deviation in the measurements where the 'disparity shift' is applied is in fact considerably lower compared to the measurements where the function is not applied. The difference between the deviations is 11.6 millimetres. This difference is of great importance to the further elaboration of this functionality.

In fact, the goal of the to be developed algorithm is not to know what the exact distance is between the camera and the floor in a 'normal' scenario versus an 'abnormal' scenario. In contrast, the algorithm must focus on the actual difference and deviation between these two scenarios. If the accuracy of real depth is a bit off, but the correctness while comparing two values is improved, this is a far better application.

For example, if the actual depth is 2.5 metres and the camera detects 2.49 metres on average, but is deviating a lot between the values, this would be bad for measuring the difference between those values. If however the camera detects 2.47 metres on average but all the values are much closer, this is beneficial for the detection since the measuring accuracy is better.
The algorithm prefers steadier data in favour of spread-out data, so it can detect the difference between states more accurately.

To conclude the testing for this concept, a disparity shift of value '7' is preferred above no disparity shift because the deviation in values is less in this test case.

3.5.5 Fluctuation depth data

Because the calculation of the exact difference is a long process, it brings up a few problems.
A first problem is again people leaving and entering an elevator, because they block the direct vision of the floor and pollute the depth data. Due to the new camera position, the door and the floor outside of the elevator are a much smaller part in the frame than in the previous position. This makes the margin of error greater, as somebody leaving the elevator can quickly block the vision of the door with more than 50%. Therefore, it is of great importance that the algorithm is executed when there is no-one blocking the vision of the door.

This brings up the second issue. To get an accurate measurement, a series of values must be recorded and compared. This means that multiple frames should be requested by GARDEN. As earlier described, it can take some time between the requests for frames. As the elevator door is only open for a few moments, there is little time to request frames which meet the requirements. In a real scenario where the door is opening and closing when people leave or enter, about 5 frames meet the requirements and can be used for the calculations.

The little input data combined with the RMS error on that distance can cause inaccurate calculations.
A final possible issue is the type of floor outside of the elevator. For our test cases the floor outside of the elevator is not reflective and made of stone. Because the tests can only be conducted with that type of scenario, a statement about the functionality in other scenarios cannot be given.

3.5.6 Different solutions

A different solution for this issue has already been designed by Oliver Krauss [47]. In this solution, a second camera is placed on top of the elevator. On each level where the elevator stops, barcodes are put up. To measure the levelling accuracy of the elevator, the camera can compare the vision of the barcode with a predefined one to check for a difference. This solution is illustrated in figure 44, wherein position '3' defines the newly placed camera.

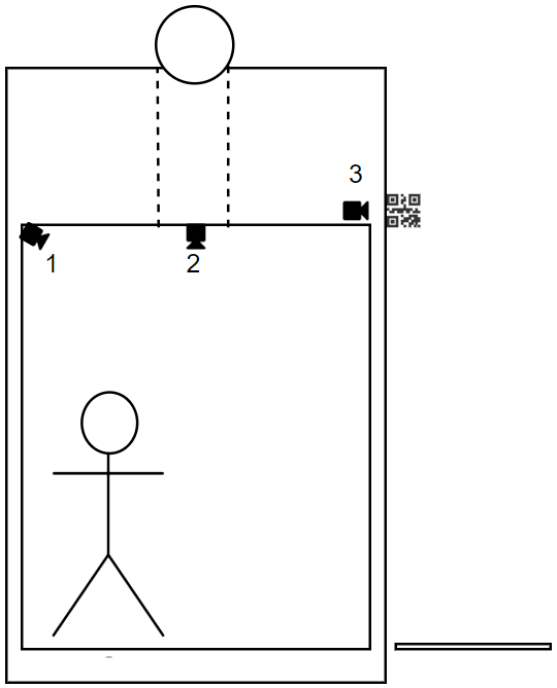

Figure 44 Elevator levelling solution

III. Proof of concept

4 Extraction door and floor

As all the possible methods to find the floor and the door are discussed earlier, the following chapters briefly touches on the actual implementation of this functionality in the GARDEN application.

4.1 Training using depth images

The first part of the training focuses on detecting the floor and the door, using a frame of an open elevator and a frame of a closed elevator in combination with the Canny and Hough algorithms.

4.1.1 Calculating depth difference

In order to calculate the location of the door, a depth difference is made between a median image of a closed elevator and an open elevator, as discussed in the research part. The difference is calculated by dividing the image in squares to reduce runtime. The squares in which a significant depth difference was noted, are highlighted in figure 45.

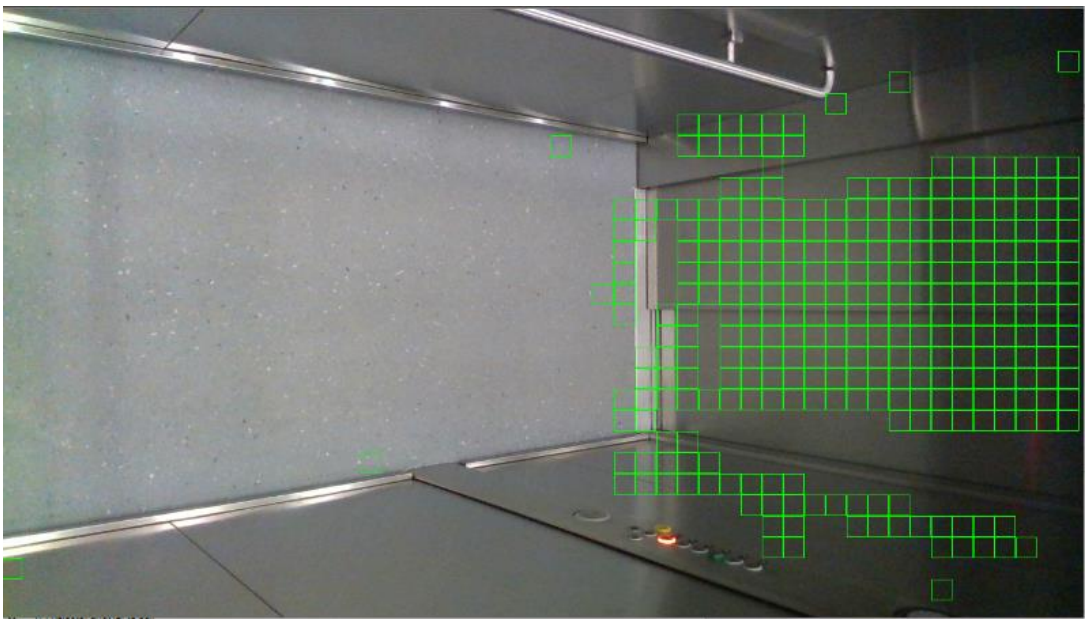

Figure 45 Depth difference elevator

4.1.2 Preparation images

In order to prepare the images for edge detection and extraction, the Canny algorithm must be applied to the images. As said in the research, the images must first be blurred and converted to grayscale for the Canny method to work properly. Once the Canny method in OpenCV is applied on the frames, it gives the following output which can be seen in figure 46.

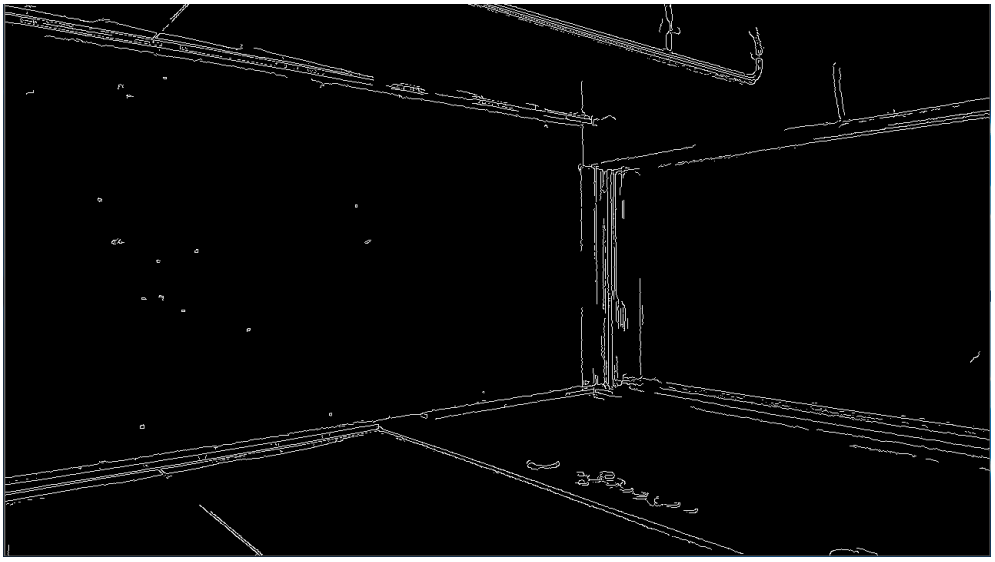

Figure 46 Canny algorithm elevator

4.1.3 Hough Probabilistic lines

A next step in the detection of the door and the floor is calculating the probabilistic Hough lines. The result of this algorithm applied on the image of the elevator is shown in figure 47.

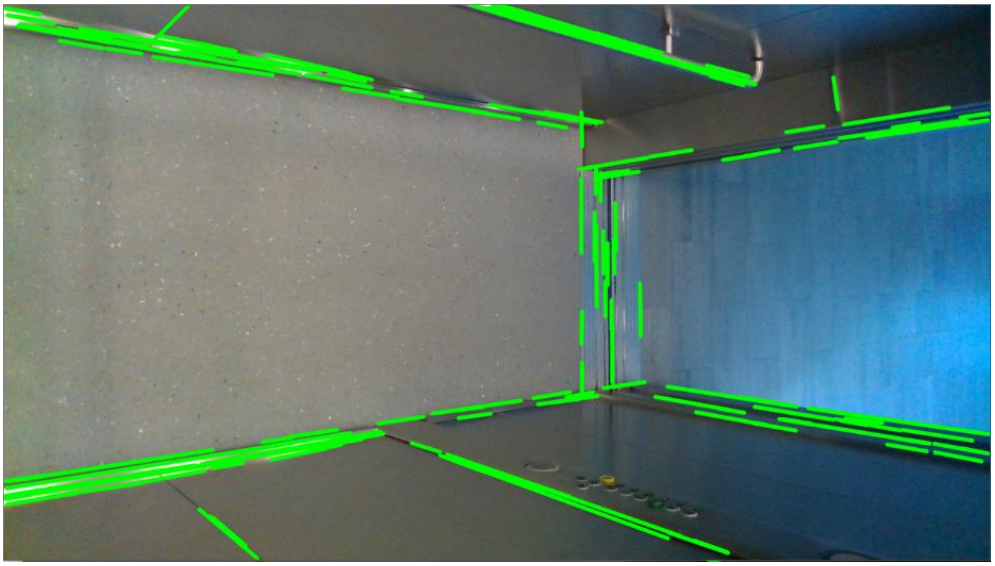

Figure 47 Hough Probabilistic elevator

With this result, the application can combine the calculated Hough lines with the previously calculated depth difference information, to determine the location of the door more precisely.

This results in the removal of detected outliers in the depth difference, as shown in figure 48. The rectangles acquired by this algorithm can now be combined into one shape, which defines the location of the elevator door.

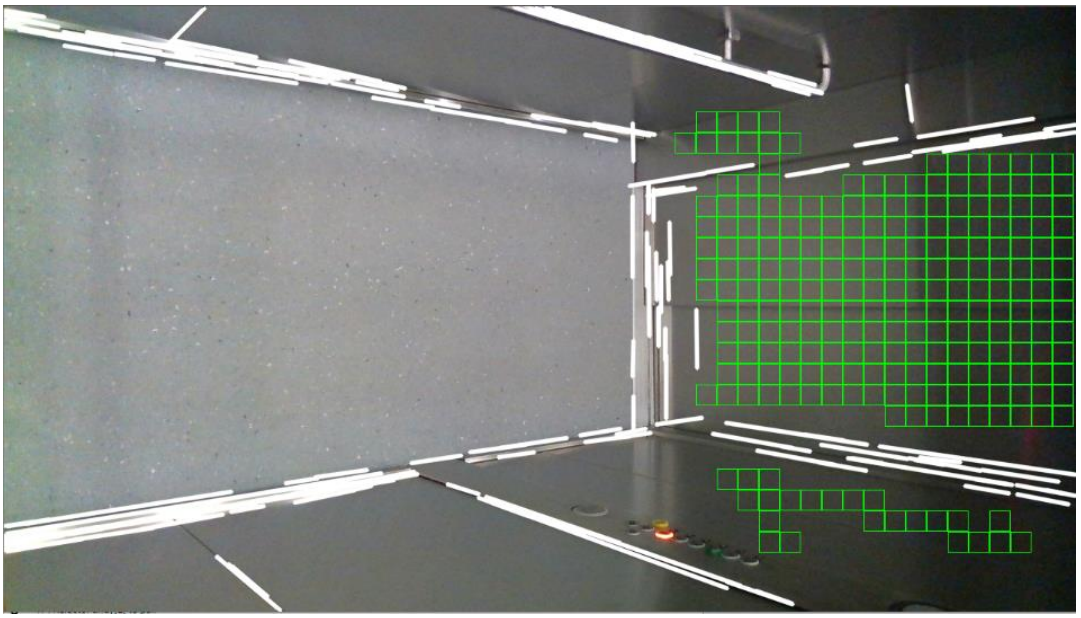

Figure 48 Hough Probabilistic with depth difference

4.1.4 Hough Standard lines

To determine the floor, the standard implementation of the Hough algorithm is used, since the application needs uninterrupted continuous lines for the detection. First, the Hough standard algorithm is applied on the RGB frame of an elevator. Since this results in a few redundant lines, lines with a similar location and angle are filtered out. A noticeable occurrence in this frame is that there is no line detected in the left side of the frame, due to the camera position. Because of this, the application draws an extra line at this location to cope with the camera placement. Once the lines are filtered and the baseline is drawn by the application, the intersection points between these lines are calculated and highlighted. The first iteration of the Hough algorithm is shown in figure 49, while figure 50 shows the filtered lines and the extra drawn line with their intersection points between each other.

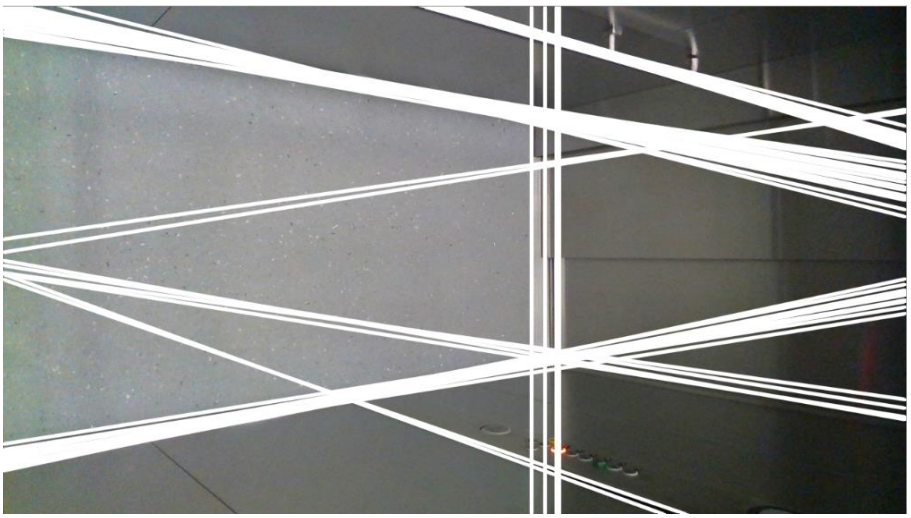

Figure 49 Hough standard result

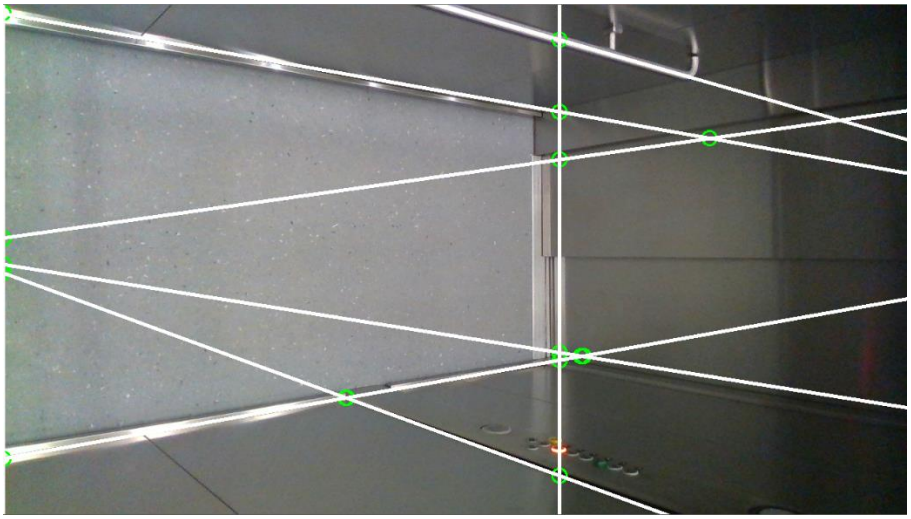

Figure 50 Hough standard result filtered

4.1.5 Intersection points

The found intersection points by the application are the intersecting areas of the lines detected by the Hough standard algorithm, but because these lines continue over the entire frame, there are a few outlying intersection points which must be filtered out. These outliers are filtered out by proximity between each other, distance to the drawn baseline or amount of intersection points per line. Another way to filter the points is using the depth information, since points which are located on the walls or door have a shallower depth than the ones defining the floor.

Once the unnecessary points are filtered, the leftover points are ordered in clockwise direction and converted into a sorted vector which defines a polygon shape which can be seen in figure 51.

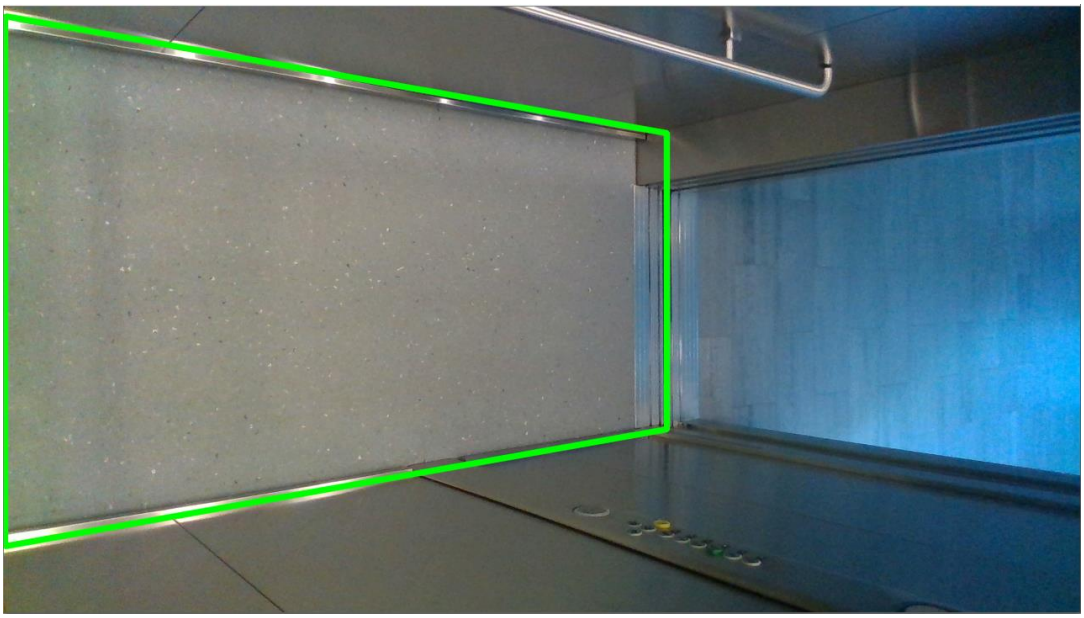

Figure 51 Elevator polygon floor result

4.1.6 Mask creation

As a final step, the application converts the rectangle which defines the door and the polygon which defines the floor to a mask. A mask in OpenCV is a matrix which defines the pixels which can be used for calculations. [48] If an element in the matrix has the value '255', the corresponding pixel can be used for calculations, if the value is '0' it is ignored. When the shapes of the door and the floor are converted to masks, their respective matrices are shown in figure 52 for the floor and figure 53 for the door. The white parts of the image resemble the value '255' and the black parts resemble the value '0'. These masks are written to files so GARDEN can use them again during runtime.

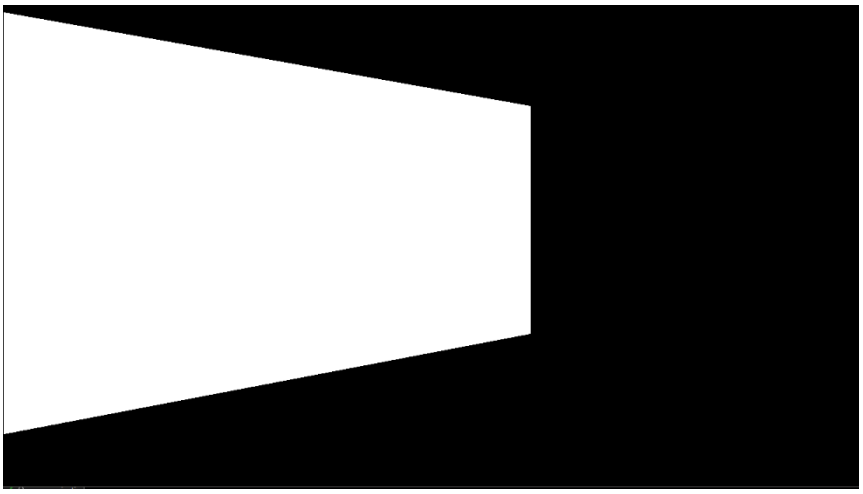

Figure 52 Mask floor

Figure 53 Mask door

5 Door status classification

The following part focuses on the actual detection of the door status in an elevator.

5.1 Available data

In order to classify the status of the door correctly, GARDEN can use certain features within a frame which are already calculated by earlier elements in the GARDEN pipeline. A few of these elements are for example the depth and RGB image, the masks of the door and the floor and the contours and extracted persons in a frame.

5.2 Image: DoorInformation

To quickly recap the operation of the pipeline, a frame is recorded by the RealSense camera and is converted to the class Image. In this class, the captured frames are saved (depth and RGB) in addition to certain extracted data such as the detected contours.

In order to hold all the information of the door status provision together, an extra class 'DoorInformation' is made wherein all data regarding the status of the door is saved. This class is added to the Image and is therefore available to use in the rest of the GARDEN application.

5.3 Door difference provider

The first step in the detection of the door status is the 'DoorDifferenceProvider'. This class is a pipeline element which calculates the average depth difference of the current received depth frame compared to the depth frames of an open and a closed elevator.

5.3.1 Contours

The first step of this provider is to subtract the persons blocking the view of the door from the calculations. In order to achieve this, GARDEN requests the already calculated contours which can be seen in figures 54 and 55.

Figure 54 Input image contour calculation

Figure 55 Output image contour calculation

These contours define the persons standing in the elevator. Once the contours are requested by the provider, they are converted to masks. In this case, the application does not want to use the depth information where a person is standing, so all the pixels wherein a person is standing are ignored by the application. This mask wherein the locations of persons are ignored is combined with the mask of the location of the door. The result of this combination is a mask, which only defines the parts of the door which are not blocked by persons. An example of this can be seen in figure 56.

Figure 56 Combined mask door and contours

Using this mask, GARDEN also calculates how much percent of the door is blocked by persons.

5.3.2 Difference calculation

GARDEN now applies this mask to the current frame and calculates the average depth of the non-filtered areas. GARDEN also applies this mask to the frame of an open and a closed elevator and calculates the respective depths.

The depth of the currently received frame is compared with the depth of the open and closed elevator states. The differences of the current depth with both the depth of the closed and the open state are written to the 'DoorInformation' class.

5.3.3 Floor status calculation

The purpose of the floor status calculation is to detect if the floor of an elevator is empty. The application must be sure if there are no persons or objects inside. To calculate this, the depth of an empty elevator is compared with the current depth. For the calculations, the mask of the floor is applied to both images, so only the floor area is used. If there is no significant depth difference between the current and the empty elevator, the DoorInformation class is updated accordingly.

5.4 Door percentage provider

GARDEN now knows the depth difference of the current state with the open and closed states of the elevator. The purpose of the 'DoorPercentageProvider' is to convert this difference to a percentage.

5.4.1 Definition depth boundaries

During the runtime of the application, GARDEN records the highest and lowest calculated depth differences for each state and saves them. This in order to generate a predictive depth border model per elevator, based on the depth difference information. The saved borders are analysed by a separate thread which runs parallel with GARDEN and writes the acquired results to a file.

5.4.2 Expressing percentage

In a next step, GARDEN converts the received depth difference information for each state to a percentage. Using both the calculated depth boundaries and the known open and closed states of the elevator, it converts the depth difference of the current frame to a percentage difference with both states. The output of this part is for example that in the current frame the door is 80% similar with the 'closed' state and 20% similar with the 'open' state.

5.5 Door movement provider

In order to assist the classification of the door status, movement is also detected using RGB images to check if a door is opening or closing. To detect this, the Lucas-Kanade and FeaturesToTrack algorithms are used, as discussed in the research. The movement is detected using the 'DoorMovementProvider'.

A first definition for this provider is the configuration of the movement direction. In the GARDEN pipeline JSON configuration, the movement direction is coupled to the movement state. For example, if the movement is from left to right, this movement flow can be set to 'opening' or 'closing' in the configuration.

The movement of the door can only be detected by GARDEN when the vision of the door is not blocked too much, since otherwise the movement of the persons interfere with the movement of the door.

An example of how the optical flow algorithm works can be seen in figure 57. The red arrows show the direction of movement.

Figure 57 Lucas-Kanade movement in elevator

In order to cancel out movement by persons, the detected direction of movement can be set in the configuration. Using this camera position, the door movement is from top to bottom. The movement of people leaving and entering is mostly from left to right. This means that GARDEN can cancel out all the movement from left to right, since the door never moves in this direction. This is shown in figure 58, the green arrow resembles the detected movement while the red arrow resembles the ignored movement.

Figure 58 Movement persons removal

5.6 Door state analyser

The final part of the classification of the door status is analysing all the gathered information. Since GARDEN saved all the calculated information into the DoorInformation class, only this class is necessary to calculate the door status.

The actual defining of the door status is completed by the 'DoorStateAnalyser' class. This class combines the difference percentages with the 'open' and 'closed' state with the door movement and all the other available information, to calculate the status of the door. This status can be 'open', 'closed' or 'in between'. Together with the provision of this state, a confidence level is calculated depending on for example the visibility of the door, the movement, the noise of the depth information etcetera. The final output is again saved to the DoorInformation class.

An example of the content of the DoorInformation class can be seen in figure 59. This readout shows that the door is closed, but because the depth information is noisy and the door is blocked, the confidence is low.

```
05/27/19 10:58:29 [ERROR] - Door State: Closed
Visibility Door: Slightly blocked
Door Movement: No Movement
Floor State: People or object Inside
Percentage Closed: 86 %
Percentage Open: 1 %
Average percentage open: 7.500000 %
Percentage Door Blocked: 34.448606 %
Confidence door state: 47.551391 %
```

Figure 59 Content DoorInformation class

6 Elevator levelling

In a previous part of this paper, the capabilities of the Intel RealSense camera were discussed and researched. The result of this research is that theoretically a height difference of 10 or 20 millimetres is measurable with the revised camera position. However, the executed test to acquire this information, was in controlled conditions with ideal objects.

To measure if this approach is sustainable and possible in a real scenario, a small proof of concept is developed. This proof of concept is independent of the GARDEN pipeline and will thus not be implemented in the EDEN project. The only purpose of any code written for the following part is to provide data to check if the required functionality is possible.

The proof of concept uses certain features provided in the GARDEN project, mainly focused around the acquisition of frames from the D435 camera.

6.1 Test setup

6.1.1 Camera position

To conduct the test, the camera is positioned in the middle of the left side panel of the elevator when the user is inside the lift, facing the doorway. The camera is positioned in a 45-degree angle pinched between the wall on one side and the ceiling on the other. The camera is secured with tape, so it will not move during the test. The vision of the camera is shown in figure 60.

Figure 60 Vision new camera position

6.1.2 Detection area

The detection area to measure the depth information is manually narrowed down to the visible floor, in order to get the most clean and accurate measurements.

The detection area is highlighted in figure 61 with a green outlining.

Figure 61 Highlighted detection area

6.1.3 Mock height level difference

Mocking an error in the levelling of an elevator is no easy task. Breaking the elevator's systems to mock this anomaly is not a preferred method of approach.

To cope with this issue, a temporary cardboard floor is constructed, which is placed at the entrance of the elevator. This floor is placed on level 1, while the floor on level 2 is the regular scenario.
The floor is constructed out of cardboard and paper which make up an extra height of 20 millimetres compared to the regular floor. The height is checked with a ruler on various points, to ensure the difference is indeed 20 millimetres. An example of the construction can be seen in figures 62 and 63.

Figure 62 Height level difference floor

Figure 63 Height level difference floor top view

6.2 Amount of test data

This test scenario uses an already recorded stream by the Intel RealSense camera. For this stream, the D435 camera was activated with the settings profile found in Appendix B. Notable in these settings are the resolution of 1280x720 and the 'disparity shift' of value '7'. The 'disparity shift' value was chosen due to the previously conducted research.

To have an equal test scenario over multiple runs, the RealSenseImageProvider class is initialised with the 'buffer' settings. This means that it will load all the frames of the pre-recorded stream and show them consecutively without skipping any frames. This means that in all the scenarios, the frames which are received and computed are always the same.

For all the tests, 1002 frames were recorded. In this set of frames, there are 165 frames where the door is completely open. In the other frames, the door is either moving, closed or there are people blocking the vision. In the open state, there are 75 frames where the floor is in the status 'Normal' which means that there is nothing wrong with the levelling. During 90 frames, the floor is in the status 'Error', in this state there is a height difference of 20 millimetres.

6.3 Test method

For all the different types of input images, the same testing methodology is applied.

First, the algorithm connects with the RealSenseImageProvider. Parameters for this connection are the path to the file and the option to use a buffer. Since the buffer is used, all the frames of the recording are loaded into the RAM memory.

In the next step, the method loops over all the frames in the buffer. Once a frame is received in the form of an Image class, the depth data is requested. In this depth data, the algorithm loops over all the values which are of interest. This concludes all the pixels in our previously defined region of interest (ROI). Per pixel in the ROI, the depth value is requested. This is a floating-point number, which defines the distance from the camera to that point in metres. This value is pushed to a vector.

When all the distances in the ROI of this frame are pushed to the vector, the method sorts the values inside and uses the sorted vector to calculate a median value. This median value is the median depth of the floor for this frame and is recorded to a file. Coupled to this value is the current status of the door. This can be either 'OPEN NORMAL', 'OPEN ERROR', 'CLOSED' or 'PERSONS'.

This process is repeated for every frame in the recording.

The recording is analysed with two different methods of post-processing. Since the input data of the new elevator position is very noisy, the GARDEN application uses built-in post-processing algorithms of the Intel RealSense camera. These algorithms fill-in the depth of uncalculatable areas in the frame. Since there are two major different types of these filters within the RealSense SDK, they are both considered in this proof of concept.

An example of this noisy input data can be seen in figure 64. The black areas represent pixels without depth information.

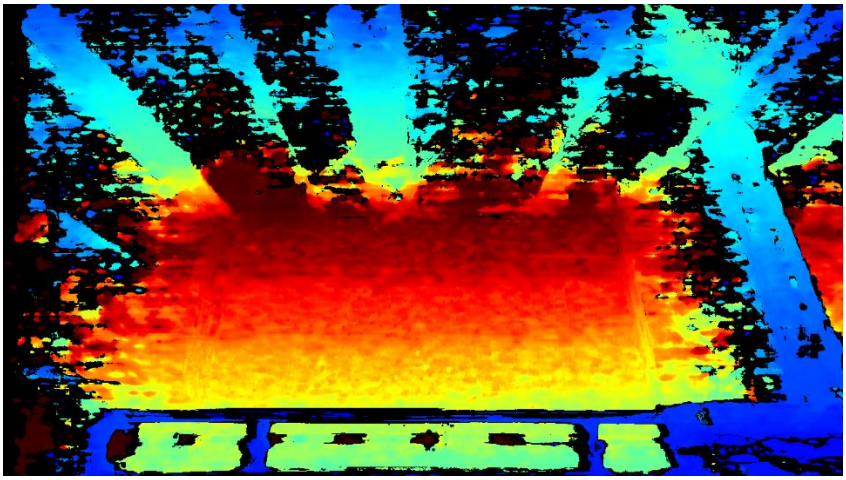

Figure 64 Noisy depth information new camera position

6.3.1 Post-processing differences

The first post-processing algorithm that is used is the standard 'hole filling filter'. This filter has 3 options, namely 'Use Nearest', 'Use Furthest' and 'Fill from left'. As explained by Anders Grunnet-Jepsen, this method of post-processing uses depth information from the background of the frame to fill the missing values. He continues by highlighting that this method of hole filling is known to give a lot of false values since it just copies existing depth data from the frame into the missing areas. [49]

For this test and in GARDEN, the 'Use Nearest' option is selected, which according to the RealSense documentation uses the value from the neighbouring background pixel closest to the sensor. [49] [50] An example of this post processing filter can be seen in figure 65.

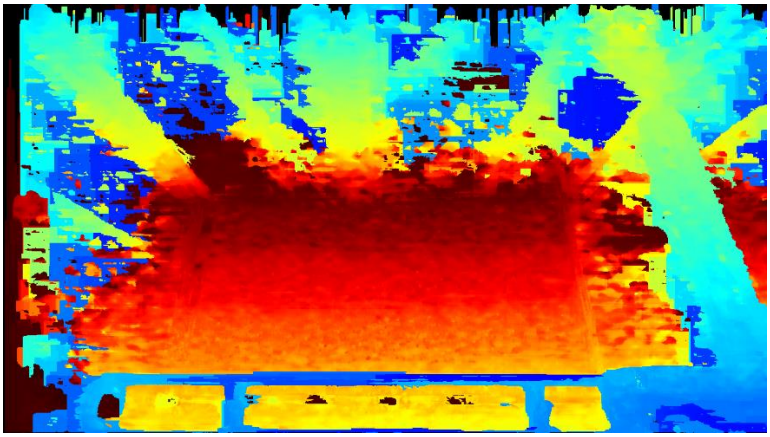

Figure 65 Hole filling filter depth frame

The second post-processing filter that is used is the 'Temporal filter'. According to the paper by Anders Grunnet-Jepsen the temporal filter uses time averaging to improve the depth. This means that if a certain pixel has no depth information, the filter uses previous depth information to cope with the loss of depth and 'predicts' the depth value. [49]

For this test and in GARDEN, the 'Valid in 1/last 8' option is selected. As said in the documentation about the post-processing filters, this option uses a valid pixel value from the last 8 frames if there is no current depth information available. [50] An example of the 'Temporal filter' can be seen in figure 66.

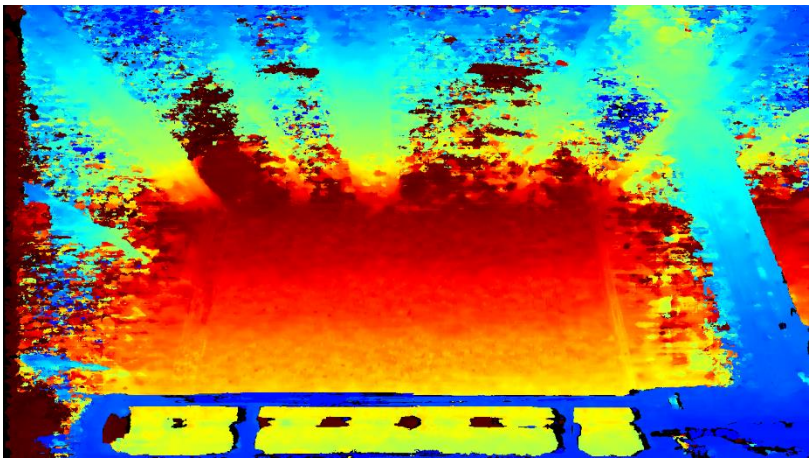

Figure 66 Temporal filter depth frame

6.4 Test results

Using the calculated median values per frame, as mentioned earlier, the results can be compared. To reduce clutter, only the 'OPEN' states are considered. Frames where the door was closed, or people were blocking the vision of the floor are left out.

6.4.1 Hole filling filter

All median values are separated by the different status they represent, per status the average of medians is calculated. These calculated values are represented in table 4 and the graph in figure 67.

Table 4 Depth difference hole filling filter

	Average of median depth (in metres)
OPEN ERROR	2.25196
OPEN NORMAL	2.26864

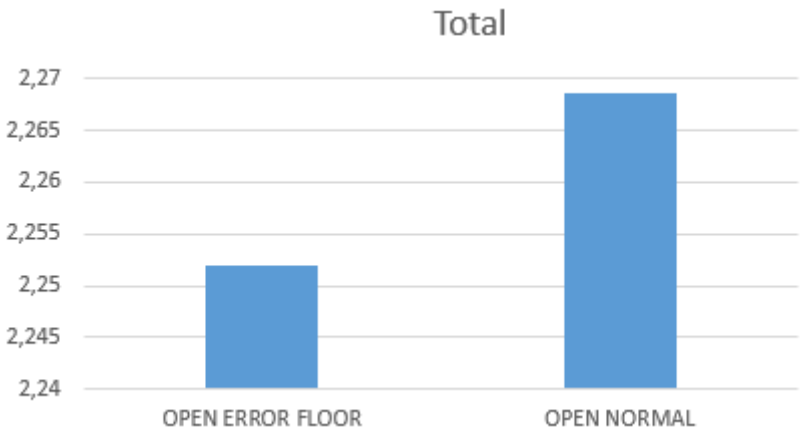

Figure 67 Graph depth difference hole filling filter

What these values show is that the difference between the 'ERROR' status and the 'NORMAL' status is around 16.68 millimetres using the 'Hole filling filter'. This means that the distance to the floor in the 'NORMAL' state, is deeper compared to the 'ERROR' state.

6.4.2 Temporal filter

The same graph and table are also available for the 'Temporal filter'. The calculated values for this test case are represented in table 5 and figure 68.

Table 5 Depth difference temporal filter

	Average of median depth (in metres)
OPEN ERROR	2.25053
OPEN NORMAL	2.26813

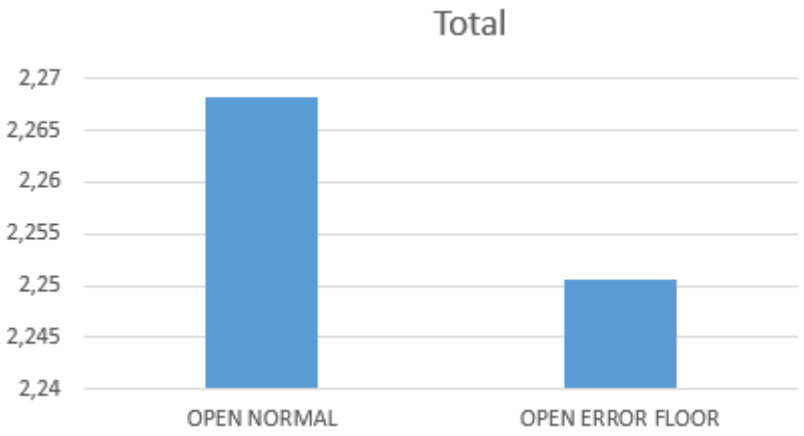

Figure 68 Graph depth difference temporal filter

The 'Temporal filter' shows a depth difference of around 17.6 millimetres. So, this test case also suggests that the 'ERROR' floor is located higher than the 'NORMAL' floor.

6.4.3 Conclusion

As said in the research part regarding the RealSense camera, the RMS error for 2.5 metres is around 25 millimetres. If an average depth of 2.3 metres, as measured in the POC, is used in the equation to calculate RMS error, a value of 13.23 millimetres is obtained.

In the measurements the respective values of 16.68 mm and 17.6 mm are acquired. If these values are compared with the actual height of the object, which was 20 mm, there is only a small difference of about 3 mm. With an expected RMS error of about 4 times as much, this can be classified as a good result. With this in mind, depth information can be of use to detect a levelling error in an elevator, as it is able to detect a height difference greater than 20 mm.

However, to acquire these results, a perfect scenario is used wherein the elevator door stays open longer than usual and wherein the vision of the floor is not polluted by persons entering or leaving. In addition to this, many frames are used to calculate the median and average values. Gathering this amount of good data in a real scenario is doubtful and due to this, using depth information cannot be seen as an optimal solution.

Conclusion

To conclude this paper, it can be stated that using just an Intel RealSense camera is not the preferred nor the most effective way to detect the door status or the levelling of an elevator car. However, the focus of this paper is to research and develop the possibility to tackle these cases using just one Intel RealSense camera, so the system can operate independently from any other hardware or sensors connected to the elevator. If this aspect is brought into the equation, then it can be concluded that the detection of the door status can be achieved using a combination of RGB and depth imagery. In addition to this, the classification of the door status can also be executed when users are partially blocking the vision of the door by subtracting them from the depth image. If the newly determined state is combined with its confidence level, the entire result can be proven useful for the classification of emergencies in elevator networks. Finding the location of the elevator floor and door is also achieved using methods provided by OpenCV and is also based on a combination of RGB and depth information. Since this detection is heavily dependent on the line detection and extraction algorithms and on the quality of the depth information, the accuracy of the result is determined by the parameters of algorithms like Canny and Hough, since the combined output of these methods forms the base of the training. Since the algorithm also uses the depth information, the quality of the depth information is also rather important. If the depth data is too noisy or incalculable, this will certainly affect the results of the training.

To recap the conclusion of the detection regarding the elevator levelling, it can be stated that theoretically and practically speaking the Intel RealSense D435 camera can detect faults in the stopping accuracy of an elevator car. However, the usage of this technique is not feasible in an actual scenario, since there are too many scenarios wherein the detection cannot be executed. This results in an inconsistent and inaccurate result when deployed as a self-sustaining system. In addition to this, a change in camera position is needed to reduce the general RMS error in depth measurements, which also changes the results obtained for the door detection and the GARDEN system in general.

A final note is that using computer vision to detect the door status is an approach which can be used in a variety of situations outside of the project scope. Detecting the status of an elevator door is only the tip of the iceberg, as the general approach can also be applied for doors in trains for example.

Reflection

For my internship and bachelor thesis I had the opportunity to complete them in a research group. This was something that I really wanted, since opportunities to work for a research team at a university are not that common. In addition to this, the suggested project EDEN also looked promising. Because I really like challenges, being able to join a big project wherein they use C++ and OpenCV (both technologies which I had never used before) the project immediately got my attention. This in combination with working abroad in Austria where you are pulled away from all the known and where you have to act independent both on and off the internship, it sounded like a real challenge.

The first weeks in Austria did definitely not disappoint in that aspect. Adapting myself to living alone, without a car and in a different culture was a big change on itself. Combined with being thrown in this big project which uses technologies that were unknown for me, it was a challenge.

My first tasks in the project were mostly to familiarise myself with the codebase and the technologies. At work I ran through the code and the OpenCV framework and tried to connect all the loose ends and configurations to understand everything. In the evenings I mostly used Pluralsight to get a quick course in C++ and the usage of OpenCV. Luckily my colleagues were always very supportive and helped me with the questions I had. After familiarising myself with everything and analysing the planned developments for my part, I started programming. I am someone who learns everything by doing it, so I just started to program small parts like showing images and using the matrixes to have a better understanding of everything.

This approach worked well, since I could quickly familiarise myself with the project and the technologies, which resulted in a first prototype of a door state detection. After this point, everything went quite smooth. If I had troubles, I could always rely on my colleagues for guidance. I also always received valuable feedback, which pushed me in the right direction for development.

In the project I could really focus on researching new technologies and learning to use and understand them in a short time period. I had the opportunity to work very independent and figure things out by myself. This resulted in a good understanding of the things I realised. Because of this I could grow a lot on a technical level. I learned how to use C++, computer vision and versioning with rebasing in the EDEN project. But that aside, I also got the opportunity to learn more about AI, deep learning and neural networks because my colleagues were actively working in that field of expertise.

The internship highlighted some strong aspects of me as a person, like my determination and adaptability to new situations. But it also highlighted some things that I can still improve. One of those things was that when I got stuck, I kept searching for a solution myself instead of quickly asking a more experienced colleague. This was mostly the case when I got problems using OpenCV or linking errors.

I am glad that I got the opportunity to join the EDEN project. It was a joy working with people who have a higher level of education than myself, they taught me so much on a professional and a personal level. They were just the best colleagues an intern could ever wish for.

Bibliographical references

[1] University of Applied Sciences Upper Austria, “Corporate Structure - FH OOE,” University of Applied Sciences Upper Austria, 2019. [Online]. Available: https://www.fh-ooe.at/en/about-us/corporate-structure/. [Accessed 13 4 2019].

[2] AIST, “Projects - AIST,” AIST, 1 4 2019. [Online]. Available: https://aist.fh-hagenberg.at/index.php/en/projects. [Accessed 13 4 2019].

[3] O. Krauss, “Project Eden - AIST,” AIST, 1 4 2018. [Online]. Available: https://aist.fh-hagenberg.at/index.php/en/projects/project-eden. [Accessed 13 4 2019].

[4] R. Meindl, *State oriented emergency detection in elevator networks,* Hagenberg, In progress.

[5] Google, “Developer Guide | Protocol Buffers,” Google, 23 8 2018. [Online]. Available: https://developers.google.com/protocol-buffers/docs/overview. [Accessed 13 04 2019].

[6] Intel, “Buy Intel RealSense Depth Camera D435,” Intel, [Online]. Available: https://store.intelrealsense.com/buy-intel-realsense-depth-camera-d435.html. [Accessed 20 04 2019].

[7] Intel, “Intel RealSense SDK 2.0,” Intel, [Online]. Available: https://software.intel.com/en-us/realsense/sdk. [Accessed 20 4 2019].

[8] OpenCV team, “About,” OpenCV, 2019. [Online]. Available: https://opencv.org/about/. [Accessed 19 04 2019].

[9] W. contributors, “Google Test,” Wikipedia, The Free Encyclopedia, 28 04 2019. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Google_Test&oldid=894466051. [Accessed 24 05 2019].

[10] Google, “Google Test,” GitHub, 18 04 2019. [Online]. Available: https://github.com/google/googletest. [Accessed 24 05 2019].

[11] Atlassian, “Git feature branch workflow,” Atlassian, [Online]. Available: https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow. [Accessed 20 04 2019].

[12] K. L. T. C. Yuan B, “Finding the Best-Fit Bounding-Boxes,” in *International Workshop on Document Analysis Systems*, Nelson, New Zealand, 2006.

[13] G. D. a. L. Cahill, “An adaptive Gaussian filter for noise reduction and edge detection,” in *Nuclear Science Symposium and Medical Imaging Conference*, San Francisco, USA, 1993.

[14] S. P. A. a. E. W. R. Fisher, “Spatial Filters - Median Filter,” 2003. [Online]. Available: https://homepages.inf.ed.ac.uk/rbf/HIPR2/median.htm. [Accessed 13 05 2019].

[15] B. S. a. P. Cavanagh, “What Line Drawings Reveal About the Visual Brain,” 28 10 2011. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3203412/. [Accessed 24 05 2019].

[16] J. Canny, “A Computational Approach to Edge Detection,” IEEE, 6 November 1986. [Online]. Available: https://ieeexplore.ieee.org/document/4767851. [Accessed 11 4 2019].

[17] P. K. Kalra, “Canny,” 23 03 2009. [Online]. Available: http://www.cse.iitd.ernet.in/~pkalra/col783-2017/canny.pdf. [Accessed 24 05 2019].

[18] OpenCV, “OpenCV: Feature Detection,” OpenCV, 18 12 2015. [Online]. Available: https://docs.opencv.org/3.1.0/dd/d1a/group__imgproc__feature.html. [Accessed 24 05 2019].

[19] OpenCV, “OpenCV: Canny Edge Detector,” 18 12 2015. [Online]. Available: https://docs.opencv.org/3.1.0/da/d5c/tutorial_canny_detector.html. [Accessed 24 05 2019].

[20] W. contributors, “Canny edge detection,” Wikipedia, The Free Encyclopedia, 14 04 2019. [Online]. Available: https://en.wikipedia.org/wiki/Canny_edge_detector. [Accessed 24 05 2019].

[21] E. Elboher, “Efficient and Accurate Gaussian Image Filtering Using Running Sums,” 25 07 2011. [Online]. Available: https://arxiv.org/abs/1107.4958. [Accessed 24 05 2019].

[22] OpenCV, “OpenCV: Miscellaneous Image Transformations,” OpenCV, 18 12 2015. [Online]. Available: https://docs.opencv.org/3.1.0/d7/d1b/group__imgproc__misc.html. [Accessed 24 05 2019].

[23] W. contributors, “Hough transform,” Wikipedia, The Free Encyclopedia, 13 02 2019. [Online]. Available: https://en.wikipedia.org/wiki/Hough_transform. [Accessed 24 05 2019].

[24] OpenCV, “Hough Line Transform - OpenCV C++,” OpenCV, 10 11 2014. [Online]. Available: https://docs.opencv.org/3.0-beta/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html#hough-lines. [Accessed 25 05 2019].

[25] OpenCV, “Hough Line Transform - OpenCV Python,” OpenCV, 10 11 2014. [Online]. Available: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html. [Accessed 24 05 2019].

[26] W. contributors, “Polar coordinate system,” Wikipedia, The Free Encyclopedia, 22 05 2019. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Polar_coordinate_system&oldid=898285749. [Accessed 24 05 2019].

[27] W. contributors, “Cartesian coordinate system,” Wikipedia, The Free Encyclopedia, 01 05 2019. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Cartesian_coordinate_system&oldid=895000788. [Accessed 24 05 2019].

[28] C. G. J. K. J. Matas, “Robust detection of lines using the progressive probablistic Hough Transform,” *Computer Vision and Image Understanding,* vol. 78, no. 1, pp. 119-137, 200.

[29] OpenCV, “cv::Mat Class Reference,” OpenCV, 29 08 2018. [Online]. Available: https://docs.opencv.org/3.4.3/d3/d63/classcv_1_1Mat.html. [Accessed 25 05 2019].

[30] OpenCV, “OpenCV: Operations on arrays,” OpenCV, 04 06 2018. [Online]. Available: https://docs.opencv.org/3.4.2/d2/de8/group__core__array.html. [Accessed 25 05 2019].

[31] D. W. C. L. D. S. R. M. D. B. D. D. L. R. I. J. Oliver Krauss, “System Design Document EDEN,” Internal company document - not publicly available, Hagenberg, 2019.

[32] C. Lederhilger, “Objekterkennung in reflektierenden Aufzugsanlagen zur Notfallerkennung mit OpenCV,” Internal company document - not publicly available, Hagenberg, 2019.

[33] R. B. A. A. S. B. G. a. P. F. Nazanin Sadat Hashemi, “Template Matching Advances and Applications in Image Analysis,” Cornell University, New York, USA, 2016.

[34] OpenCV, “OpenCV: Template Matching,” OpenCV, 28 05 2019. [Online]. Available: https://docs.opencv.org/trunk/d4/dc6/tutorial_py_template_matching.html. [Accessed 28 05 2019].

[35] OpenCV, “OpenCV: Template Matching,” OpenCV, 29 08 2018. [Online]. Available: https://docs.opencv.org/3.4.3/de/da9/tutorial_template_matching.html. [Accessed 28 05 2019].

[36] OpenCV, “OpenCV: Object Detection,” OpenCV, 29 08 2018. [Online]. Available: https://docs.opencv.org/3.4.3/df/dfb/group__imgproc__object.html. [Accessed 28 05 2019].

[37] W. contributors, “Optical flow,” Wikipedia, The Free Encyclopedia, 10 02 2019. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Optical_flow&oldid=882672317. [Accessed 2019 05 28].

[38] B. D. L. a. T. Kanade, “An iterative image registration technique with an application to stereo vision,” *IJCAI'81 Proceedings of the 7th international joint conference on Artificial intelligence - Volume 2,* no. Vol. 2, pp. 674-679, 1981.

[39] OpenCV, “OpenCV: Object tracking,” OpenCV, 28 05 2019. [Online]. Available: https://docs.opencv.org/3.4/dc/d6b/group__video__track.html. [Accessed 28 05 2019].

[40] OpenCV, “OpenCV: Optical Flow,” OpenCV, 28 05 2018. [Online]. Available: https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html. [Accessed 28 05 2019].

[41] J. S. a. C. Tomasi, “Good features to track,” in *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, Seattle, USA, 1994.

[42] Zhiyuan, “Lucas-Kanade Tutorial example 2,” MathWorks, 15 12 2014. [Online]. Available: https://de.mathworks.com/matlabcentral/fileexchange/48745-lucas-kanade-tutorial-example-2?focused=e7c36cf7-fca2-e03c-e308-dff7d0b1a0cc&tab=example. [Accessed 28 05 2019].

[43] European Committee for Standardization, *Safety rules for the construction and installation of lifts - Lifts for the transport of persons and goods - Part 20: Passenger and goods passenger lifts,* Afnor Normalization, 2011.

[44] A. Grunnet-Jepsen, “Tuning D435 and D415 cameras for optimal performance,” Intel, 05 2018. [Online]. Available: https://realsense.intel.com/wp-content/uploads/sites/63/BKM-For-Tuning-D435-and-D415-Cameras-Webinar_Rev3.pdf?language=en_US. [Accessed 14 05 2019].

[45] J. N. S. a. J. W. Anders Grunnet-Jepsen, “Best-Known-Methods for Tuning Intel® RealSense™ D400 Depth Cameras for Best Performance,” Intel, 2019. [Online]. Available: https://www.intel.com/content/dam/support/us/en/documents/emerging-technologies/intel-realsense-technology/BKMs_Tuning_RealSense_D4xx_Cam.pdf. [Accessed 21 05 2019].

[46] R. F. L. G. C. S. M. S. F. U. a. Y. V. Monica Carfagni, “Metrological and Critical Characterization of the Intel D415 Stereo Depth Camera,” in *Sensors*, Basel, 2019.

[47] D. W. Oliver Krauss, “Automatische Erkennung der Abstellgenauigkeit in Aufzugsanlagen,” Internal company document - not publicly available, Hagenberg, 2018.

[48] Ashwin, “How to apply mask in OpenCV,” Code Yarns, 20 08 2015. [Online]. Available: https://codeyarns.com/2015/08/20/how-to-apply-mask-in-opencv/. [Accessed 6 06 2019].

[49] A. G.-J. a. D. Tong, “Depth Post-Processing for Intel® RealSense™ D400 Depth Cameras,” 31 May 2018. [Online]. Available: https://www.intel.com/content/dam/support/us/en/documents/emerging-technologies/intel-realsense-technology/Intel-RealSense-Depth-PostProcess.pdf. [Accessed 22 05 2019].

[50] Intel, “librealsense post-processing-filters,” GitHub, 31 05 2018. [Online]. Available: https://github.com/IntelRealSense/librealsense/blob/master/doc/post-processing-filters.md. [Accessed 22 05 2019].

Appendices

A. JSON pipeline configuration

B. JSON configuration camera

A. JSON pipeline configuration

```
{
  "pipelineElements": [
    PIPELINE_ELEMENT,
    PIPELINE_ELEMENT
  ],
  "connections":[
    CONNECTION,
    CONNECTION
  ]
}

{
  "type": "class NAMESPACE::CLASSNAME",
  "fromType": "bool" | "int" | "class NAMESPACE::CLASSNAME",
  "toType": "bool" | "int" | "class NAMESPACE::CLASSNAME",
  "params":{
    "CLASS_SPECIFIC_PARAM": "VALUE"
  }
}

{
  "from": INDEX_OF_PIPELINE_ELEMENT_IN_PIPELINE_ELEMENTS,
  "to": INDEX_OF_PIPELINE_ELEMENT_IN_PIPELINE_ELEMENTS,
  "payloadType": "class NAMESPACE::CLASSNAME"
}
```

B. JSON configuration camera


```
{
    "aux-param-autoexposure-setpoint": "1536",
    "aux-param-colorcorrection1": "0.298828",
    "aux-param-colorcorrection10": "0",
    "aux-param-colorcorrection11": "0",
    "aux-param-colorcorrection12": "0",
    "aux-param-colorcorrection2": "0.293945",
    "aux-param-colorcorrection3": "0.293945",
    "aux-param-colorcorrection4": "0.114258",
    "aux-param-colorcorrection5": "0",
    "aux-param-colorcorrection6": "0",
    "aux-param-colorcorrection7": "0",
    "aux-param-colorcorrection8": "0",
    "aux-param-colorcorrection9": "0",
    "aux-param-depthclampmax": "65536",
    "aux-param-depthclampmin": "0",
    "aux-param-disparityshift": "0",
    "controls-autoexposure-auto": "True",
    "controls-autoexposure-manual": "99286",
    "controls-color-autoexposure-auto": "True",
    "controls-color-autoexposure-manual": "156",
    "controls-color-backlight-compensation": "0",
    "controls-color-brightness": "0",
    "controls-color-contrast": "50",
    "controls-color-gain": "64",
    "controls-color-gamma": "300",
    "controls-color-hue": "0",
    "controls-color-power-line-frequency": "3",
    "controls-color-saturation": "64",
    "controls-color-sharpness": "50",
    "controls-color-white-balance-auto": "True",
    "controls-color-white-balance-manual": "4600",
    "controls-depth-gain": "16",
    "controls-laserpower": "360",
    "controls-laserstate": "on",
    "ignoreSAD": "0",
    "param-autoexposure-setpoint": "1536",
    "param-censusenablereg-udiameter": "9",
    "param-censusenablereg-vdiameter": "9",
    "param-censususize": "9",
    "param-censusvsize": "9",
    "param-depthclampmax": "65536",
    "param-depthclampmin": "0",
    "param-depthunits": "1000",
    "param-disableraucolor": "0",
    "param-disablesadcolor": "0",
```

```
    "param-disablesadnormalize": "0",
    "param-disablesloleftcolor": "0",
    "param-disableslorightcolor": "0",
    "param-disparitymode": "0",
    "param-disparityshift": "7",
    "param-lambdaad": "800",
    "param-lambdacensus": "26",
    "param-leftrightthreshold": "24",
    "param-maxscorethreshb": "2047",
    "param-medianthreshold": "500",
    "param-minscorethresha": "1",
    "param-neighborthresh": "7",
    "param-raumine": "1",
    "param-rauminn": "1",
    "param-rauminnssum": "3",
    "param-raumins": "1",
    "param-rauminw": "1",
    "param-rauminwesum": "3",
    "param-regioncolorthresholdb": "0.0499022",
    "param-regioncolorthresholdg": "0.0499022",
    "param-regioncolorthresholdr": "0.0499022",
    "param-regionshrinku": "3",
    "param-regionshrinkv": "1",
    "param-robbinsmonrodecrement": "10",
    "param-robbinsmonroincrement": "10",
    "param-rsmdiffthreshold": "4",
    "param-rsmrauslodiffthreshold": "1",
    "param-rsmremovethreshold": "0.375",
    "param-scanlineedgetaub": "72",
    "param-scanlineedgetaug": "72",
    "param-scanlineedgetaur": "72",
    "param-scanlinep1": "60",
    "param-scanlinep1onediscon": "105",
    "param-scanlinep1twodiscon": "70",
    "param-scanlinep2": "342",
    "param-scanlinep2onediscon": "190",
    "param-scanlinep2twodiscon": "130",
    "param-secondpeakdelta": "325",
    "param-texturecountthresh": "0",
    "param-texturedifferencethresh": "0",
    "param-usersm": "0",
    "param-zunits": "1000",
    "stream-depth-format": "Z16",
    "stream-fps": "30",
    "stream-height": "720",
    "stream-width": "1280"
}
```