Volume Recognition in Video Images for Movement Optimization of Passenger Elevators

Bachelor Thesis

for the course of study

Applied Computer Science

at the

Baden-Wuerttemberg Cooperative State University Stuttgart

by

Felix Stegmaier

03 September 2018

Project Period:	12 Weeks
Student ID, Course:	6079153, TINF15A
Company:	DXC Technology, Böblingen
Supervisor:	Julia Baumhauer, Executive Master of Arts
Scientific Supervisor:	Prof. Dr. Bernd Schwinn

Erklärung

Ich versichere hiermit, dass ich meine Bachelorarbeit mit dem Thema *Volume Recognition in Video Images for Movement Optimization of Passenger Elevators* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Stuttgart, 30. August 2018

Felix Stegmaier

Declaration of Authorship

I hereby declare:

- that this paper with the title *Volume Recognition in Video Images for Movement Optimization of Passenger Elevators* is my own work and
- that I have not used any other sources or assistance than declared here.
- that I have not submitted the paper as a whole or in part for an degree at any university or institution before.
- that I have not published this paper before.
- Furthermore I confirm, that the presented electronical version of this paper is identical to the printed version.

I am aware, that an incorrect declaration will be followed by legal measures.

Stuttgart, August 30, 2018

Felix Stegmaier

Abstract

Titel	Volume Recognition in Video Images for Movement Optimization of Passenger Elevators
Autor:	Felix Stegmaier

Fahrstühle bewegen Menschen – jeden Tag. Die Fahrwege einer modernen Personenfahrstuhlanlage hängen ab von den Personen, die in jeder Etage auf ihre Beförderung zu unterschiedlichen Zielen warten sowie den Personen im Aufzug, die bereits auf dem Weg zu einer Zieletage sind. Aktuelle Systeme versuchen durch Angabe des Ziels beim Rufen des Aufzuges die Wege der Kabine für eine gleichverteilte und im Mittel schnelle Beförderung zu planen und Wartezeiten zu verringern sowie die Auslastung der Anlage zu optimieren.

In dieser Arbeit werden Möglichkeiten ergründet, diesen Prozess durch die Verwendung von Kamerasystemen in den Kabinen und vor den Einlässen zu unterstützen, indem durch Personen- und Objekterkennung die Anzahl der Passagiere und Wartenden sowie deren Platzbedarf und der Freiraum in der Kabine analysiert wird. Der Fokus liegt dabei auf der Analyse der benötigten und freie Volumina. Dies umfasst eine Implementierung eines entsprechenden Bilderkennungssystems und eine modellhafte Darlegung eines Fahrstuhlsystems, das dieses verwendet.

Das entwickelte Bilderkennungssystem verwendet mehrere Kameras und nutzt die Hintergrundsubtraktionstechnik um die Position von Personen und Objekten in der Kabine zu erfassen. Ein shilhouettenbasierter Volumenschnitt wird durchgeführt um das eingenommene Volumen der Anwesenden zu erkennen. Die so gewonnenen Daten können in einem angepassten Steuerungsalgorithmus für Aufzugsysteme genutzt werden, der großen Objekten in der Kabine Vorrang gewärt, um diese schneller zuzustellen. Da diese Objekte die Kabine für andere Fahrgäste blockieren, kann dies die Anzahl der beförderten Objekte erhöhen, was mit einer durchgeführten Simulation untermauert wird. Eine solche Modifikation der Kotrollalgorithmus kann für Umfelder sinnvoll sein, in denen Passagiere und Objekte um Beförderung im Aufzug konkurrieren.

Abstract

Title	Volume Recognition in Video Images for Movement Optimization of Passenger Elevators
Author:	Felix Stegmaier

Elevators move people – every day. The movement of a modern passenger elevator system depends on the persons on each floor waiting to be transported to different destinations as well as the passengers inside the elevator already on the way to their terminal location. Recent systems try to optimize the movement of the elevator by utilizing destination dispatch controls on each floor in such a way that travel waiting times for passengers are minimized and the utilization of the elevator is maximized.

This paper explores possibilities to support this process by the usage of camera systems within the cabin and in front of the entrances which analyze the amount and spatial needs of people waiting for the elevator as well as currently occupying it. The system employs techniques from the object and person recognition from the field of computer vision. The focus of the automated analysis is the spatial volume needed to transport the passengers. The work done for this paper includes an implementation of an appropriate image recognition system and an exemplary outline of an elevator system that uses such a system.

The developed visual recognition system uses a multi-view camera set-up with the techniques of background subtraction to register the positions of passengers and objects in the elevator cabin. The technique of silhouette-based volume intersection is used to detect the volume that these entities take up. The data gathered by this system can be used for an elevator scheduling algorithm which gives priority to the delivery of large objects in the cabin, which blocks the space for other passengers. A conducted simulation yields that this approach can improve the amount of delivered objects, which can be beneficial in an environment where passengers and cargo objects need to be moved likewise.

Preliminary Notes

Gender Neutrality

This paper is written in gender neutral language. Any person is addressed using the singular *they*. For example, instead of "The user changes *his* account settings." this paper writes "The user changes *their* account settings."

Trademarks

Designations of third-party vendors are used in this paper. These designations may be trademarks or registered trademarks. Wherever the author is aware of such trademark, the designation will be written with an initial capital letter.

Formatting

Italic text is used to emphasize new technical terms, names of persons, product names when they are first used and to highlight general keywords.
Bold text is also used to emphasize terms of interest.
`Monospace text` is used to denote variables, function names and computer commands and reference them from listings.
Italic Symbols (e.g. A) are used in formulas and are also used to reference to those symbols.

Contents

List of Figures

List of Tables

List of Listings

List of Acronyms

2D	2 dimensional
3D	3 dimensional
API	application programming interface
BDSG	Bundesdatenschutzgesetz
BSD	Berkeley Software Distribution
GDPR	General Data Protection Regulation
MOG	Mixture of Gaussians
RGB	red-green-blue
SI	International System of Units
UML	Unified Modeling Language
YAML	YAML Ain't Markup Language

1. Introduction

The usage of elevator constructions to lift goods and people dates back to a long history and gained significant importance since the industrialization. Nowadays modern high-rise buildings would not be sustainable without elevation systems capable to handle large amounts of business people on their way to and from the office. The issue of optimizing the throughput of passengers and reducing their waiting time is of everyday economic interest since it is critical for the business people using it to use their time most efficiently. Therefore movement optimization for elevators is under active academic research with direct impact for manufacturers. Such optimizations of the control mechanism depend on many parameters including timings of the system, the traffic patterns of the passengers and the kind of input mechanisms to detect passengers and terminal destination.

In this thesis, a possibility to enhance this control mechanism by utilizing computer-based camera vision is proposed. Currently, the general trend of computer vision is emerging in many fields with real-world applications and techniques of varying complexities are employed to support a broad range of use cases. In industrial systems, camera data is used to control and inspect production processes. When monitoring solutions for public or private places are considered such use cases include among others, surveillance, tracking, smart home, and statistical applications. Especially in „smart“ environments, video data is combined with other sensory data to gain insight into behavioral patterns and trigger actions in that environment.

Here the possibilities are explored to which extend computer vision can be used to improve the movement control of elevators by estimating the free spacial volume in the elevator cabins and how much space the passengers waiting for the car would take up. This could allow better utilization of the cabins and an improved passenger-to-car mapping with reduced waiting times for passengers.

1.1. Context and Motivation

This bachelor thesis is conducted in the *Digital Service Innovation* team at *DXC Technology*. The team focuses on helping clients to develop novel, digitized business models. The team is actively involved in the *Startup Autobahn* program organized by *Plug and Play*. The program provides a business platform to connect technology start-ups and global industry corporations by running a 100-day cycle in which demo projects can be conducted to test out how the start-ups can work with the companies and how their technologies can be used.

The idea for this thesis evolved in this context as DXC got interested in a start-up that deals with computer vision and found a possible client in the elevator industry. However, this first engagement did not lead to a joint project. But still, DXC is keen to follow the idea and find new partners and clients to collaborate with. Nonetheless, this project is relevant for the team even though there is no immediate financial benefit associated with it, but rather it opens up opportunities for new partnerships and client engagements when suitable technology partners are found. According to Unger [Ung15, p. 4] the four „biggest“ companies in the elevator business are *Otis, Kone, Schindler and Thyssen* which therefore are potential clients to benefit from the ideas proposed in this thesis.

For academia, this research is interesting because elevator systems typically only utilize 60-70% of their theoretical capacity in one ride [Ung15, p. 194]. When looking at the whole-day average, this figure might even be lower considering empty ride. Using the additional passenger information generated by a vision system capable of detecting the volume of passengers and other cargo might hold the opportunity to improve this efficiency further.

1.2. Goals and Boundaries

In order to set a clear focus of this thesis, concrete questions that shall be answered are helpful. Those provide guidance in the process of determining the methods employed in later investigation steps. In general two main research questions are identified:

- Which computer vision techniques can be used to detect persons and objects within an elevator car and in front of the lift and estimate their spacial volume?
- How can the information acquired by a system that uses such methods constitute to the optimization of algorithms that control the movement plan for elevators?

To contribute to this general questions, it is useful to also evaluate some secondary questions:

- The volume of what kind of other objects are except for humans are interesting for this topic?
- Under which circumstances is it useful to integrate such a system into an elevator system?
- To which extent can the usage lead to improvements in utilization, waiting time, ride time and economic effort?
- Which of the named optimization goals are targetable by this kind of analysis and are they desired from a passenger perspective?

In order to answer those questions, an explicit plan to follow is laid out in chapter 3 which is executed and implemented in chapter 4. This will also determine which analysis, implementation and evaluation actions are taken. In general, an experimental approach is used, where an exemplary implementation of the volume detection system is performed which is tested on real-life sample footage. An analytical approach is used to derive the possibilities to integrate the volume data into the scheduling of an elevator.

Additionally, to keeping the project size manageable, it is necessary to exclude aspects from the investigation explicitly. Especially, it is of no interest to track and identify individual persons and analyze their behavior. Even though some kind of surveillance could be established this way, it is not desired for this thesis. Furthermore, considerations about the market potential for the solution and how it can be targeted to customers are performed. Also, no long-time analysis of passenger traffic is performed based on the volume measurements. This includes that no machine learning techniques are applied.

1.3. Structure of this Document

This thesis is structured into multiple chapters that first build a general understanding of the problem and the technologies needed to solve it and leads to a final implementation.

- This chapter gives an introduction into the topic and outlines the problem statement which is worked on in the thesis.
- Chapter 2 describes the current state of the art in the fields of elevator control and movement optimization, as well as the basics of image processing that are needed for the conduct of this project. It lays out which methods, technologies, and tools can be used to approach the problems in this project.
- Chapter 3 explains the methods used to conduct research for the project and plans a concept for the solution. An artifact centered approach is used, which plans to build and evaluate two products: one to perform the visual volume measurement and one to show the effectiveness of integrating this data into the scheduling of an elevator.
- Chapter 4 describes how the proposed solutions are implemented and tested. The solution is reflected upon from a functional and economic perspective. The details of the implementation planned in the chapter beforehand are laid out and the artifacts are tested according to the set criteria.
- Chapter 5 gives a short summary of the conducted method and the outcomes of the project. The findings are discussed and the execution of the project is reflected upon. Finally, an outlook into future developments in the field and the opportunities and follow-up actions for DXC Technology and the Digital Service Innovation Team are outlined.

2. Fundamentals

In this chapter, the state of the art of the technologies and concepts used in this thesis are outlined. A broad overview over object detection using computer vision and the control of elevator systems is given. This forms the basis to explore the problem further and find possible solutions in the chapters afterward.

2.1. Computer Vision

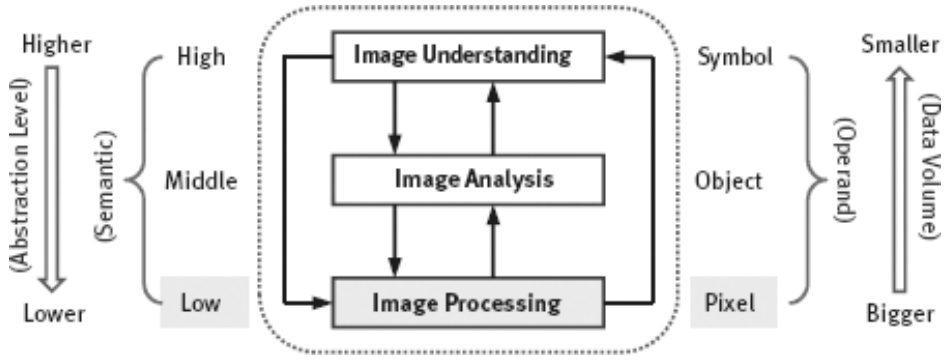

Figure 2.1.: Levels of image engineering. Reprinted from Zhang [Zha17b, Chapter 1]

The field of computer vision tries to establish methods and algorithms that extract useful information out of image data to be used by machines. The field has a broad range of applications and is influenced by various subjects such as mathematics, physics, and computer science. Therefore Zhang [Zha17b] coined the term of *image engineering* to localize the methods that deal with obtaining data from images. Zhang [Zha17b] defines three abstraction layers that these methods operate on (see figure 2.1). *Image processing* deals with the „manipulation of an image to produce another (improved)

image“ [Zha17b]. It includes image capturing, representation and reconstruction. *Image analysis* is concerned with the extraction of higher level data from the image which described the objects and features present in the image. In order to gain domain knowledge about the content of the image, this data can be processed by the means of *image understanding*. This data can then be used to make decisions about the objects in the image. Since the methods of each layer operate on data of the lower layer, a complex computer vision system employs methods from all three fields.

2.1.1. Image Representation

Image data can be captured with different systems which produce image data of various dimensions. It consists of information that can be visualized in any manner, such as still picture in color or grey-scale, a video, or a volumetric scan [Zha17b, Chap. 1]. In general, an image can be thought of as a function $f : D \to V$, where D is the domain of the location of a point in the image and V is the domain of the properties of the real world represented at this point. Typical examples for D are two dimensional images where the points are of the form (x, y), three dimensional images with points of the form $(x, y, z) \in \mathbb{R}^3$ or videos with points of the form $(x, y, t) \in \mathbb{R}^3$ where t is the time dimension. Typical examples for V are grey-scale images where the luminosity is described as a value in $\mathbb{R}$ or a colored image where the color is described as a red-green-blue $(r, g, b) \in \mathbb{R}^3$.

For digital images, V and D are discrete with upper and lower limits. In two dimensions an image point at a discrete position $(y, x) \in \mathbb{N}^2$ is called a *pixel*, in three dimensions a point $(x, y, z) \in \mathbb{N}^3$ is called a *voxel*. In order to store such an image on a computer a *matrix representation* can be chosen. Examples: For two dimensional images this matrix is element of $V^{X \times Y}$, where X and Y describe the width and height of the image [Zha17b, Chap. 1]. For three dimension $V^{X \times Y \times Z}$ can be used, where XYZ describe the length of the image in each spatial direction. A discrete (digital) two dimensional red-green-blue (RGB) color image can be represented as a matrix $F \in \mathbb{N}^{X \times Y \times 3}$. The value of a pixel can then be obtained using the function $f_F : \mathbb{N}^2 \to \mathbb{N}^3; (x, y) \mapsto f(x, y) = F_{x,y}$.

2.1.2. OpenCV Computer Vision Library

OpenCV (Open Computer Vision Library) is a free (as in freedom) and open-source library, which collects many common functionalities for advanced computer vision. The Berkeley Software Distribution (BSD) license allows it to be used in private, academic and commercial contexts. It is written in performant C/C++ and offers application programming interface (API) bindings for C++, Python and Java on Windows, Linux, Mac OS, iOS and Android [Ope18d]. It features algorithms that operate on all three layers of image engineering as defined by Zhang and can be used to implement complex visual systems. Its documentation is extensive and offers practical examples for a lot of problems it solves.

2.1.3. 2D Object Detection

Convolution Kernels

A key concept in image processing is the application of a *convolution matrix* to an image, which is also known as a *kernel*. It is a small matrix with the same dimensions as the image it is applied to. For 2 dimensional (2D) images 3×3 or 5×5 kernels are used. To apply the kernel to the image, they are convoluted. This is done for each pixel individually by „overlaying“ the kernel with its center onto it. Then the sum of the element-wise multiplication of the kernel and the overlaid area of the image is calculated in order to determine the resulting pixel value in the output image [Ope18c]. For a 2D image $F \in \mathbb{N}^{X \times Y}$ the convolution $H \in \mathbb{N}^{X \times Y}$ using the kernel $K \in \mathbb{N}^{M_i \times M_j}$ is calculated with the formula in figure 2.2:

$$H_{x,y} = \sum_{i=0}^{M_i-1} \sum_{j=0}^{M_j-1} F_{x+i-a_i,x+j-a_j} K_{i,j}$$

Figure 2.2.: Application of an 2D convolution kernel. Reprinted from the OpenCV Dev Team [Ope18c]

Convolution kernels can be used to find local features in an image or apply general local operation. This includes blurring with the Gaussian filter, and edge detection with the Sobel filter in X and Y direction or with the Laplace filter (see figure 2.3). It can

also be used to sharpen an image. The use of convolution kernels is not limited to 2D images, the same method also works for higher dimensional images, when appropriate kernel matrices are used.

$$G = \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}; G_x = \begin{bmatrix}1 & 0 & -1\\ 2 & 0 & -2\\ 1 & 0 & -1\end{bmatrix}; G_y = \begin{bmatrix}1 & 2 & 1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{bmatrix}; D^2_{xy} = \begin{bmatrix}1 & 1 & 1\\ 1 & -8 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Figure 2.3.: Examples for kernels, left to rigth: Gaussian blur filter, Sobel filter in X direction, Sobel filter in Y direction, Laplace filter

Background Subtraction

Background subtraction used to detect areas of interest in a video. Especially in a static camera setup, everything that moves can be considered the foreground and everything that stays static is the background. To do so, a model of the background without any moving objects is needed. It acts as a reference frame to compare the current frame of a video against.

If an image is available, that only shows the background of a scene, *static background subtraction* can be used. In this method, the current frame of the video and the background are blurred and the element-wise absolute difference between the images is calculated. If the absolute difference for a pixel is higher than a specified threshold, then the pixel is considered part of the foreground. This method uses the given background image as a static model for the background and does not update the model if the background changes.

When no pure background image is available or if the background can change over time, e.g. by different lighting, the background model needs to be regularly adapted. Piccardi [Pic04] presents an overview of techniques used to create such a model, whereas here only a subset is presented:

- **Running Gaussian average** For each pixel I in the grey-scale video image a running average μ_t is calculated, that is updated in each frame by the formula

$\mu_t = \alpha I_t + (1-\alpha)\mu_{t-1}$, where α is an weight that determines how fast the model is updated. The variance σ_t is calculated similarly. A pixel I_t is considered foreground if $|I_t - \mu_t| > k\sigma_t$ holds, where k is a parameter that determines the threshold sensitivity of the algorithm.

- **Mixture of Gaussians (MOG)** Instead of assuming a single value likelihood distribution for each pixel, multiple Gaussian distributions are used in this method introduced Stauffer and Grimson [SG99]. When using K distributions this enables the distinction of K types of background and foreground, which improves the detection rate in outdoor scenes where e.g. leaves and branches of trees might be present. For each of the K distributions the peak amplitude $\omega_{i,t}$, average $\mu_{i,t}$ and variance $\sigma_{i,t}$ are stored. The distributions are ranked by the ratio $\omega_{i,t}/\sigma_{i,t}$, since more narrow distributions are assumed to belong to a background classification. The distributions are checked in order of their rank to accept a pixel value when $(I_t - \mu_{i,t})/\sigma_{i,t} > 2.5$ holds, where I_t is the current pixel value. If ω_t of the first accepting distribution is higher than a threshold, the pixel is considered background. The distribution parameters are updated using a running calculation (as above) when a new pixel value is an accepted by it. If no distribution accepts the value, the distribution with the lowest ranked is replaced by a new distribution centered around the new value with a high variance. Improved forms of this algorithm exist, like [CMV11], and are currently used.

Since video images can contain pixel level noise, it is useful to convert color video to grey-scale and to blur the images before applying the background subtraction.

Edge Detection

Edge detection is the process of finding edges of objects in a grey-scale image. An edge is present where ever there is a sharp change in luminosity along a curve in the image. It is important to note that edges can also occur within the are of an object and must not necessarily describe its outline. There are two major approaches to edge detection.

First the application of the *Laplace filter kernel* (see D^2_{xy} in figure 2.3) yields an approximation of the second order derivative of the image. The second order derivative describes the divergence of the gradient of the image and therefore is capable of finding the „turning point“ in the luminosity since the second order derivative in this points is zero. It is common to combine this operation with prior application of a Gaussian blur [Sze10, pp. 237f.].

The second approach is called the *Canny algorithm* [Can86]. In order to find edges, first, two partial first-order derivatives of the image are approximated by applying the Sobel kernel in X and Y direction respectively (G_x, G_y in figure 2.3). The sharpness (or strength) of the edge in each point can then be described as the element-wise euclidean norm $G = \sqrt{G_x^2 + G_y^2}$. The angle of the normal of the edge curve in this point can be found by the element-wise two-argument inverse tangent of the two partial derivatives: $\Theta = atan2(G_x, G_y)$.

Blob Detection

A *blob* is a collection of positively marked image points in a binary image, which are located near to each other and therefore form a coherent group. One can distinguish between two types of blobs: One is a fully connected blob, where only those image points are considered to belong to the same blob, that are direct neighbors. The other is an area of image points which might only be loosely connected. It consists of multiple connected areas, which are close to each other are considered to belong to the same blob.

The method to find fully connected blobs is called *connected component labeling*, and two fundamental methods exist to perform it: *label propagation algorithms* and *label equivalence solving* [He+17]. In the label propagation approach, the image is each image point is scanned and labeled. When a new, unlabeled point, which is positively labeled in the binary image, is encountered, it is given a new label and all of its neighbors are given the same label. When an already labeled image point is encountered, the label is copied to each of the positively labeled neighbors. In the label equivalence solving approach, first, every image point, that is positively labeled in the binary image, is given a unique label. In the next step, it is solved which labels are equivalent and belong to the same blob by unifying labels of neighboring image points. Both

approaches yield an image with the same domain dimension as the binary image but have pixel values in $\mathbb{N}$, where the value designates the number of the blob that a pixel belongs to.

The method to find partially connected blobs is based on grouping connected blobs together that are close to each other. The algorithm used for this is the following [Ope18a][Mal15]:

1. Find the contours present in the image and calculate their centers. The contours can be found by following the border (edge) of an area and thereby enclosing a local positively marked area of pixels [SS85].
2. Group these centers together based on a threshold for the distance for their location to form a loosely connected blob.
3. For each of these groups, find its overall centers and estimate area and radius of the blob.

The blobs can then be enumerated based on their centers and a unique label is given to each blob and the image point that belong to it.

2.1.4. 3D Scene Reconstruction

By utilizing images from multiple perspectives of a single or multiple objects and their surroundings reconstructing the geometrical surface of these objects is possible. Depending on the number of images and perspectives, this can be performed with varying detail and completeness if the reconstructed scene.

Camera Projection and Calibration

Since camera lenses use a perspective projection to produce images, there are several considerations needed when working with 3 dimensional (3D) scenes that are captured. The points in the 3D scene can be described as coordinates in the *world space*. In the perspective projection, these points are seen from a single point in the camera, which is the sensor in a real camera. To perform the projection the position of the point seen on the *image plane* is calculated. The image plane is the section of the scene

that is seen by the camera in the distance of the focal length. Figure 2.4 shows how this projection can be visualized. Note that the Z axis is pointing into the real world space in the direction the camera is looking to. Since this projection is dependent on the focal length of the camera and the principal point, which is seen in the center of the image, it can be described as the intrinsic camera matrix A in the equation in figure 2.5. When world space and camera space coordinates are not the same, i.e. the camera is rotated or translated, there exists an additional rotation and translation matrix $[R|t]$ that describes these external projection parameters.

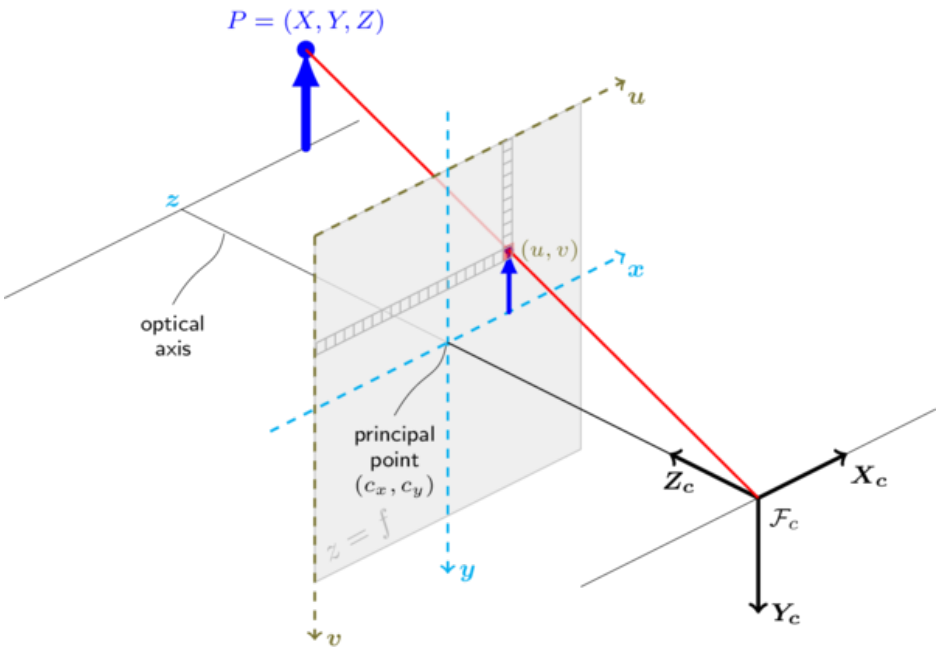

Figure 2.4.: Camera model to illustrate perspective projection of a 3D scene onto the image plane (grey). Reprinted from the OpenCV Dev Team [Ope18b]

$$sm' = A[R|t]M' \quad \Longleftrightarrow \quad s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Figure 2.5.: Camera calibration with internal and external parameters. Reprinted from the OpenCV Dev Team [Ope18b]

The equation in figure 2.5 depicts how a point in the world coordinate system can be mapped to the image plane of a camera. The following list explains the mentioned coefficients:

- s is the scale factor for image, such that the z axis of the image plane is 1
- (X, Y, Z) is the 3D point in world space
- (u, v) is the projected point in the image plane
- A is the internal camera matrix
- (c_x, c_y) is the principle point in the world space seen at the image center
- f_x, f_y are the focal lengths in pixel units
- R is the external rotation matrix
- t is the external translation vector

The internal camera matrix A can be calculated from multiple images of an object with known structure and dimensions, called a *calibration pattern* OpenCV Dev Team [Ope18b]. For example, a chessboard can be used, which is photographed from different positions and angles. The corners of the chessboard squares are used determine how a straight line is projected in the camera in order to approximate the internal camera matrix.

The external parameter for rotation and translation can not be found from one camera alone. In a multi-camera setup, they can be found by triangulation of multiple points

in multiple images that show the same scene from different angles. If this triangulation is not possible because the points used for triangulation are not visible from all camera perspective, the rotation matrix and the translation vector can be found out by measuring the real world. The rotation matrix can be calculated by measuring the relative rotation angles of the cameras to each other. The equation in figure 2.6 can be used to determine the combined rotation matrix when the rotation around the yaw, pitch, and roll angles are known. Yaw, pitch, and roll describe the rotation around the Z, Y and X axis respectively. The translation vector can be found by measuring the distance of the cameras to each other.

$$R = R_z(\alpha)R_y(\beta)R_x(\gamma) = \begin{bmatrix} cos\alpha & -sin\alpha & 0 \\ sin\alpha & cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos\beta & 0 & sin\beta \\ 0 & 1 & 0 \\ -sin\beta & 0 & cos\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\gamma & -sin\gamma \\ 0 & sin\gamma & cos\gamma \end{bmatrix}$$

Figure 2.6.: External rotation matrix from rotation along the angles α , β and γ

Furthermore, real lenses introduce optical aberrations and there exists *distortion* in camera images. Distortion is visible in the effect where straight lines appear curved in the image representation. There exists two kinds of distortion that can occur, namely radial and tangential distortion [WCH92, Fig. 2]. Radial distortion makes points appear closer together or further apart depending on their distance to the image center. It is visible as a „barrel“ or „pillow“ effect on rectangular lines. Tangential distortion causes a rotation of the image depending on the distance and angle of a point to the image center. The amount and type of distortion is part of the internal camera parameters. Just as the matrix of intrinsic parameters, the parameters for the distortion can be calculated from images of structures with known dimensions [Ope18b].

Single View Volume Estimation

When the geometric shape of a 3D object is roughly known beforehand and only its exact outline and scale varies, it is possible to estimate its volume from its outline in a single image. The assumption here is that the volume scales with the area, that the

object covers on the image [LHS90] [Wan+17] [Sap+17] . Therefore a reference object with known dimensions is necessary to calculate the scale against. This approach avoids the need to multiple cameras and a 3D calibration. However, it does not generalize to 3D objects of unknown shape.

Point-Based Scene Reconstruction

Multiple images of the same scene from different perspectives enable the reconstruction of the 3D geometry of the scene. In order to do so, a 3D point in the scene needs to be found in multiple perspectives. The point is a local feature which can be recognized and matched across images. Using perspective projection and triangulation can then be used to find out the 3D position of this point.

The simplest form of this kind of reconstruction is *stereo vision*, where a scene is reconstructed from two images taken from known positions [Zha17c, Chap. 2]. Figure 2.7 demonstrates how a 3D point P is perceived in two 2D images as points P_L and P_R. When the relative position of the cameras O_L and O_R are known, the relative position of the point P in 3D space can be found via triangulation. This triangulation is performed for every point that can be matched between the two images in order to create a *depth map*.

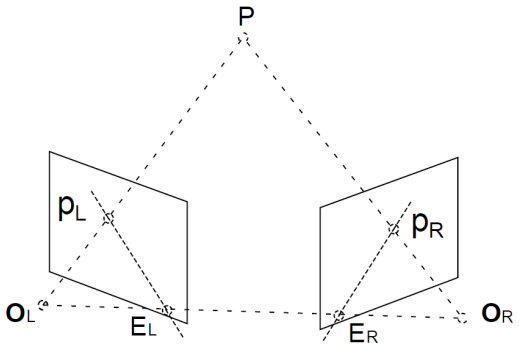

Figure 2.7.: Projection of a 3D Point into two 2D images of different perspective. Reprinted from ZooFari [Zoo09]

However, with stereo vision, it is not possible to fully reconstruct a scene, as for example the back side of objects can not possibly be seen from two perspectives that have points in common. It is required to take images from many more perspectives into consideration to fully reconstruct a scene at every angle. When the camera position for each of those images is known, it is possible to create a depth map for each adjacent perspective pair. The depth maps are then combined to reconstruct the full geometry of the scene.

In order to enhance the capability to create the depth maps, cameras with integrated depth sensors can be used. Examples for such cameras are the (discontinued) Microsoft Kinect [Mic18] and Intel RealSense [Int18]. Figure 2.8 depicts which parts of an object are visible in a 2D situation, and of which parts the depth is actually reconstructable, depending on the perspective of the available cameras. The concept is applicable to 3D scenes similarly.

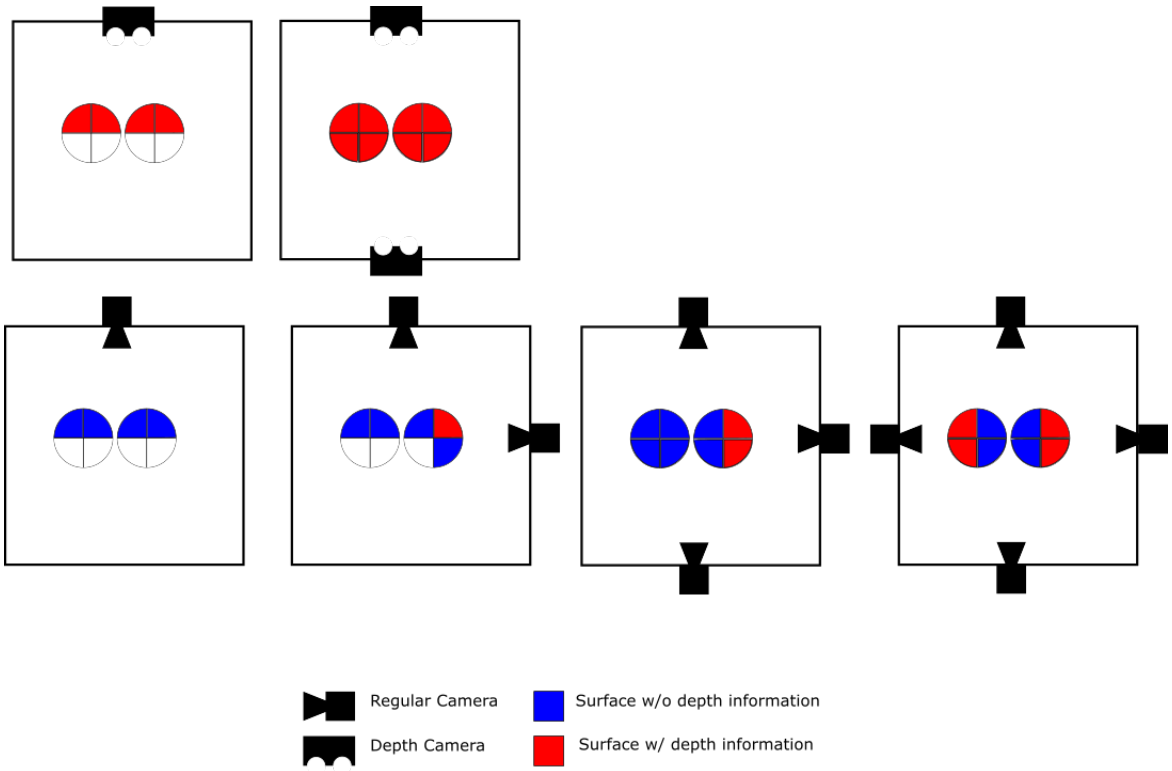

Figure 2.8.: Concept of depth reconstruction in a 2D multi-view situation. Adapted from Sonaten [Son11]

Silhouette-Based Volume Reconstruction

The *silhouette* of an object within an image is intuitively defined as a binary mask describing the areas of the image in which the object is present. The edges of this areas describe the contours of the object. The silhouette can be obtained by image segmentation of the original image [Zha17a, Chap. 2], such as by background subtraction. Depending on the camera perspective different portions of the object are visible.

Only the outline of the object from this perspective is visible in the silhouette. This implies that is is not possible to capture any concave parts of an object.

Silhouettes from multiple perspectives can be used to reconstruct the 3D shape of an object from 2D images of said object. The reconstructed 3D shape does not represent the real geometry of the object but produces the same silhouettes as the real object when seen from the angles it was reconstructed from. The shape that is constructed from the silhouettes is called *visual hull*, a term introduced by Laurentini [Lau94]. Since the silhouette does not include information about concavities, the visual hull also is convex. Furthermore, the silhouette does not include information about parts of the object whose visibility is obstructed from a certain perspective. Therefore the visual hull can not describe the shape of parts of the object, that are not visible in any silhouette.

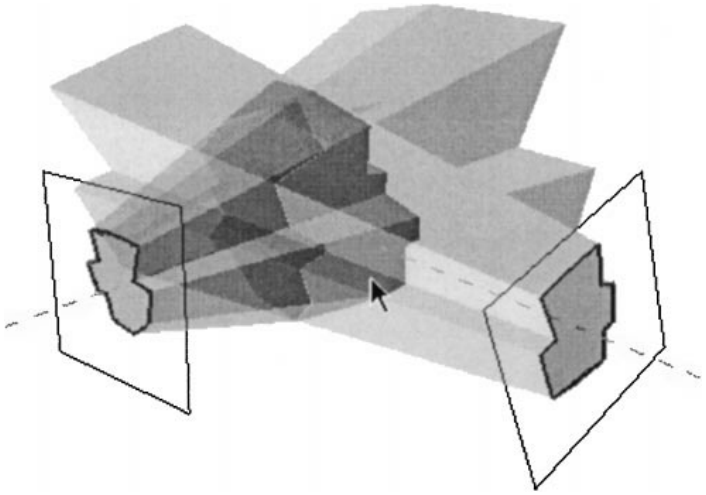

Figure 2.9.: Volume intersection of cones from silhouettes. Reprinted from Bottino and Laurentini [BL01]

Figure 2.9 visualizes how the visual hull is created from two silhouettes. Since the cameras that capture the silhouette are perspective, cones are used for its projection into the object space. The intersection of those cones creates the visual hull. There are several methods to implement silhouette based shape reconstruction:

- **Standard Voxel-Based Method** In this method, first, the space to be modeled is described as a 3D grid of voxels at a constant resolution. To find out which voxels belong to the reconstructed object, each voxel is projected into the

viewport of each camera and it is checked whether it lies in the silhouette from this perspective. Only if it lies inside of all silhouettes it remains part of the finale volume description.

- **Space Carving** This method, introduced by Kutulakos and Seitz [KS99], is concerned not only with reconstructing the volume of an object, but also to color its surface consistently to the images it is reconstructed from. The method also begins with a scene full of voxels. However, not all voxels need to be considered for the reconstruction. Instead, the method operates in iterations, where only the voxels on the surface of the volume are checked. Each voxel on the surface is checked for *photo-consistency* with each silhouette, which means that if it is visible from that perspective, it is correctly classified as background or foreground and has the correct color. All surface pixels that are not consistent are removed iteratively until the reconstructed volume converges.

Since the volume intersection method operates by projecting voxels into each perspective, the internal and external calibration parameters of the cameras for each perspective needs to be known. Furthermore, it is not possible to reconstruct the volume with absolute measures without determining the dimensions of the modeled room first and deriving a scaling factor out of it.

2.2. Elevator Control

When looking at elevator systems that are installed in buildings one property that is noticeable to the passenger, except for the physical properties, is the way the elevators are moved in order to pick up and deliver the passengers to their destination. The algorithms used to control this movement influence how long the passenger needs to wait for a cabin and how long their journey takes. Therefore it affects the perceived quality of the system. However, in buildings with an increased traffic demand, it is necessary to use elevator systems with multiple cabins and more complex control strategies. In this section, the general principles about those control strategies are laid out.

2.2.1. System Components

In order to gain an understanding of how an elevator system is controlled, it is useful to look at the components a general system includes. To do so, the mechanisms involved in a typical elevator ride from a passenger's perspective shall be outlined. Figure 2.10 gives a general overview over the involved logical components and their connections.

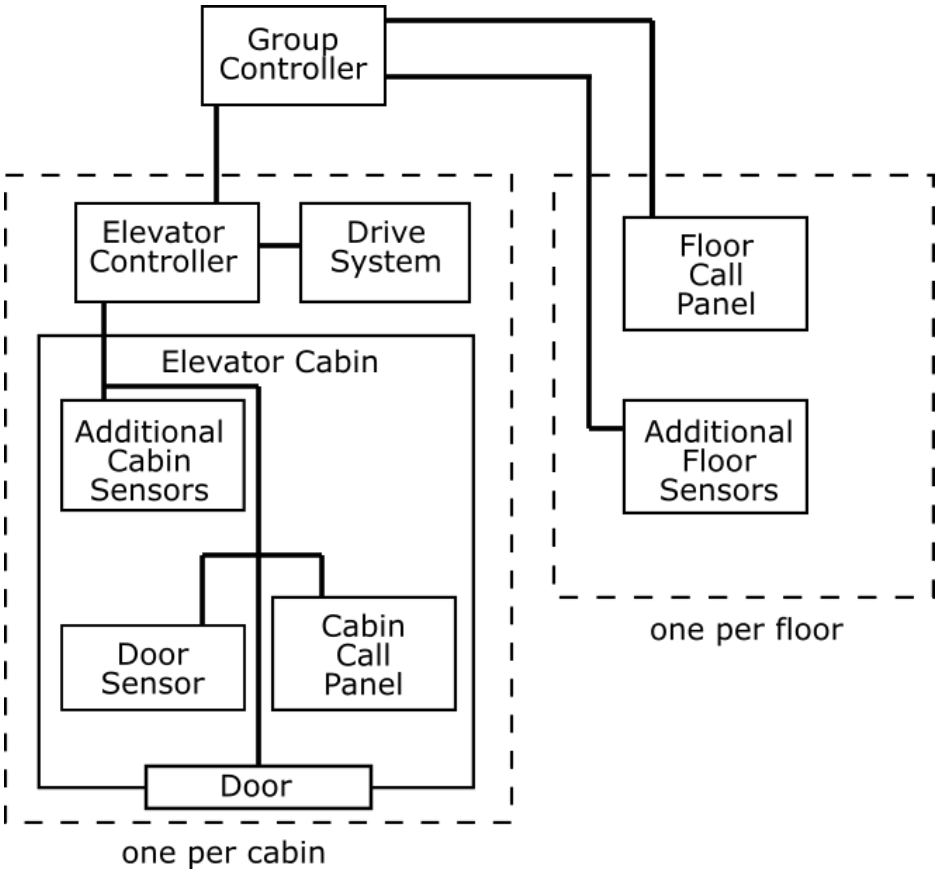

Figure 2.10.: General components of an elevator control system. Adapted from Wang et al. [Wan+16, pp. 4, 16] and Siikonen [Sii97, p. 10]

When a passenger wants to take the elevator from one floor to another, they press a button on the *floor call panel* and issue a *hall call* (or *reservation call*, *landing call*) to the system [Sii97, pp. 6–10]. This hall call signals to the system that an elevator needs to pick them up. The button pressed might be a simple push button, but can also be more complex and capture information about the direction or destination of travel [Ung15, pp. 89–93]. Even before the passenger presses the panel button, *addi-*

tional sensors on the floor can detect their intent to use the elevator and gather more information about them. This can include among others camera systems [Lin+11], badge scanners, or „smart home“ installations [KLB14]. In a multi-elevator system, the hall call is registered by the *group controller* and stays active until it is deleted when a cabin picks up the passenger. The group controller *schedules and dispatches* an elevator move to the respective floor and serve the call. In the case of a single-elevator system, no group controller is necessary. Different scheduling algorithms can be used to determine which cabin should serve which hall call next. The dispatch signal is picked up by the *elevator controller* of the respective cabin, which then instructs the *drive* system to move the elevator along the shaft to the respective floor by actuating the motors.

Once a cabin stops at the floor, it opens its *door* and the passenger enters the elevator. To select their destination floor, they use the *cabin call panel*. This *car call* (or *destination call*) [Sii97, pp. 6–10] is passed to the group controller, which schedules the elevator to move to the destination floor at some time. The group controller needs therefore to consider hall calls, as well as car calls in its scheduling algorithm Before the elevator starts to move, the door closes again, if no obstacle is detected by the *door sensor*. Until the elevator serves the destination floor, the car call stays active. *Additional sensors* inside the cabin provide more information about the passengers inside. This can include among others a weight scale to measure the total weight utilization, or a kind of camera system [Wan+16]. Furthermore, the position of each elevator is known to the group controller.

2.2.2. System Classifications

Elevator systems come in different configurations with different purposes. Depending on their use cases, different technical implementations can be chosen. Therefore they can be classified in various dimensions.

The first observation to make is the **number of shafts and cabins** in the system. The main difference comes in the scheduling complexity regarding single-cabin versus multi-cabin systems. Systems with multiple elevators need a group controller to decide which cabin serves which hall call. This decision can be non-trivial as laid out in section

2.2.4. When only one cabin is used, the scheduling becomes more straightforward. In small residential buildings, a single cabin might be sufficient. However, in buildings with larger passenger amounts, such as office buildings, a multi-elevator setup can become necessary increase to the handling capacity.

Next is the classification by the **type of objects or people** that are transported in the system. Especially important is the distinction regarding the purpose of the elevator system and whether people and/or large objects can be conveyed. Unger [Ung15, pp. 141–158] describes the properties of the most common types:

- Cabins that carry **only people** typically have a premium interior decoration, such as glass, mirrors or stainless steel. Usually, also cargo can be transported, but large goods might damage the cabin. Examples for a building that uses such elevators are office buildings, residential buildings or hotels.
- Elevators for **large cargo and people** have a more simple interior that is more resistant to damage by lifting carts. However, security measures regarding the doors are taken, such that the elevator is also safe to use by people. Typically a high payload weight and dimension can be moved. Examples for the usage of this type are factories, storage buildings or hospitals.
- Lifts that carry **only small cargo** can be found for example in restaurant kitchens to transport dishes. The cabin is not used by people, and the system is controlled purely from outside call panels. Due to the size restrictions, only a smaller weight can be transported.
- Cabins that convey **only large cargo** but no people can only be controlled from the outside call panel and have fewer safety restrictions, even though the cabin can be entered for loading and unloading.

This excerpt of the list is far from complete and Unger [Ung15, pp. 141–158] lists even more types, such as lifts for wheelchairs, the never-stopping paternoster or elevators on construction sites. However, these are not in the scope of this thesis and are not further considered.

Another interesting distinction to consider is the **amount of floors** that is served by an elevator system. The amount of floors influences the heeded handling capacity

of the system. The more floors there are, the more people need to travel potentially longer distances. While a building with only a few floors can be handled by a single elevator, buildings with up to 40 stories require the use of elevator groups. When more than 40 floors need to be served, introducing a *sky lobby* can be beneficial, where a fast shuttle transports passengers between the main lobby and the sky lobby. Passengers can then switch the elevator to reach their final destination [Hak03, p. 9].

2.2.3. Passenger Traffic Patterns

Passenger traffic patterns are typical streams of passengers going from one floor to another. Those patterns are reoccurring regularly based on the time of the day. The movement of passengers can be described with three directions: *incoming*, *outgoing*, and *inter-floor* traffic. With those directions, the traffic patterns can be described, as they focus on one of the directions, but also incorporates components of passengers heading in other directions [Sii93, p. 259]. According to multiple sources, three general patterns are present in office buildings [Bee15, pp. 1–2] [AB13, pp. 6–7] [Ung15, p. 194] [Sii97, p. 14]:

- **Up-peak traffic** In the morning the majority of passengers in an office building constitutes to incoming traffic. They arrive at the entrance lobby and travel to different upper floors to fill the building. The lift cars potentially need to stop at every level and return to the lobby without passengers to pick up new ones. This reduces the utilized capacity and puts a high load on the elevator system to convoy all passengers in time.
- **Down-peak traffic** In the evening the majority of the passengers are leaving the office building and travel from arbitrary upper floors to the main lobby. Most of the passengers constitute to outgoing traffic. The elevator system needs to pick up passengers at every level. Reverse to the up-peak traffic, the cars are empty when returning from the lobby. Similar to the up-peak traffic this situation induces high stress on the system.
- **Inter-floor traffic** All other traffic that is neither incoming nor outgoing can be considered inter-floor traffic. Passengers that travel from one floor to another but are not entering or leaving the building are the majority in this situation.

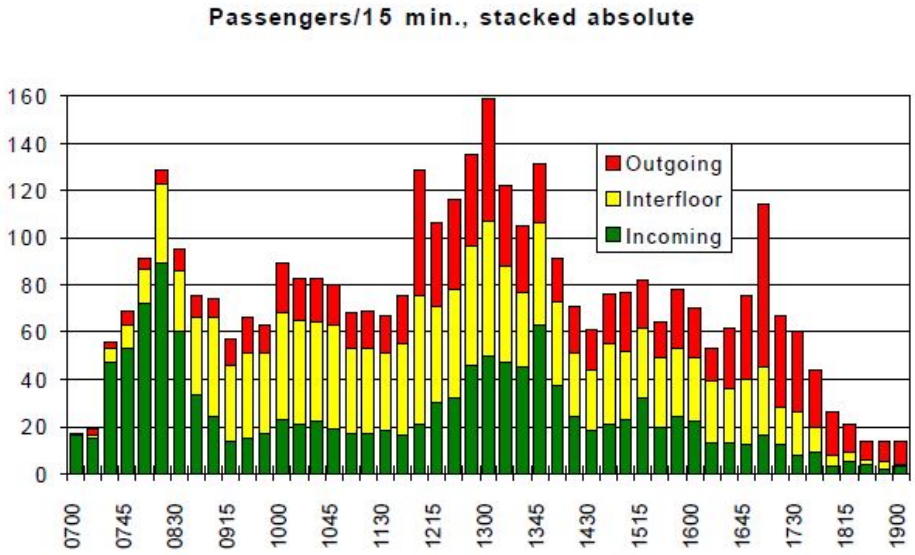

Figure 2.11.: Exemplary traffic component profile for an office building. Reprinted from Siikonen [Sii97, p. 14]

Figure 2.11 shows an example of how the traffic components contribute to overall traffic in an office building. A clear peak of incoming traffic in the morning and outgoing traffic in the evening are visible. This information can be used to model simulations and predict traffic in various circumstances. It has a significant impact on the scheduling principles employed for each situation in order to improve the efficiency of an elevator system.

2.2.4. Control Strategies

Elevator control strategies deal with the assignment of hall calls and car calls to elevator rides in order to derive a stopping schedule for the cabins in the system. The way this schedule is set up effects the performances regarding the criteria laid out in section 2.2.5. An appropriate control algorithm must be chosen in order to optimize for performance criteria depending on the type of elevator system, it's purpose and the current passenger traffic. Depending on the type of elevator system different control strategies are used, some of which are listed below.

Sequential Control

In contrast to the strategies below, this strategy is only applicable to a single-lift system. The system answers hall calls in the order they arrive and executes the car call with priority before moving on to serve the next hall call. This way only one ride is performed a time and the calls are executed in sequence. This method is also called *single call automatic control* or *non-collective control*. Drawbacks of this strategy are an increased passenger waiting time and a decreased handling capacity. However, for cargo lifts, this can be beneficial if only one cargo item can fit the cabin at a time [BA16, p. 238].

Collective Control

This is the „most common form of automatic control“ [BA16, p. 237]. Hall calls are answered in the order of the floors they originate from rather than their temporal order. On the way up and down the cabin collects all the hall calls and deliver car calls also following the floor order. Different variations of this approach are available, where the direction of the hall call is ignored, only downward calls are collected only upwards calls are collected, or the call direction is considered for collection in both directions. This approach can be expended to an elevator system with group control, such that a hall call is answered by the first cabin to pass the floor in the desired direction [BA16, p. 238]. When all cabins are idle, the nearest available is assigned to a call. Hence this strategy is also called *nearest car* [BA16, p. 244].

Zone Approach

In this approach the floors of the building are divided into static, continuous zones also called *sectors* [BA16, p. 247]. Typically there is one zone per elevator in the group. Each lift in the group serves hall calls from only one sector while also delivering passengers to floors outside of this zone [AB13, pp. 3–6]. The order of serving of the hall calls is based on the collective control. A lift is currently outside of its zone, the zone can be served by lifts from sectors above and below it.

The distribution of zones can be chosen based on the amount of passengers on each floor. This approach is useful when the traffic is evenly distributed across the building, such as in inter-floor traffic. It can be complemented with controls for special condition to react to peak-traffic [BA16, p. 247]. The static zone approach can be extended to a dynamic sector approach where the sectors are not limited by static floor numbers but by the position of the elevators [BA16, p. 250].

Cost Function Based Approaches

Modern elevator systems come with a control system that includes a computer of some sort. This allows for more sophisticated scheduling which involves planning ahead of the time and optimizing the movement of every lift in the group. [BA16, pp.261ff] The optimization goal is described as a *cost function*. This cost function can consider many of the performance criteria listed in section 2.2.5. A basic cost function is only looking at the average total journey time of the passengers, which includes waiting time and ride time [BA16, pp.278ff]. The computer can evaluate this cost function and search for a scheduling configuration, which minimizes it [AB13, pp. 3–6]. The accuracy of the cost function can be enhanced if the destination of each passenger is known beforehand. This is the case when the passenger selects their destination on the floor call panel instead of only giving an up/down direction. The system can then allocate an explicit lift to this passenger based on the evaluation of the cost function.

Parking Policies

Another consideration in the design of a control strategy are the rules where to park a cabin when it is not in use and has no current hall calls. The place where a cabin is parked can greatly influence the response time to hall calls. Having a cabin on or near the floor, where the arrival of the next passenger is expected, makes it possible to instantly answer their hall call. In a single-lift scenario, this position might be easy to determine, however in an multi-lift group, the possibilities of parking become complex [Bee15, pp. 3]. The prediction of the next arrival can be based on a model of the current traffic pattern. Brand and Nikovski [BN04] showed that for down-peak traffic, a distribution of the lifts is sufficient, which matches the arrival rates for each floor.

For up-peak traffic, however, the problem is stated to be harder, so that dynamic programming is proposed to solve it.

2.2.5. Performance Criteria

In order to evaluate the performance of an elevator system, some metrics about it are commonly considered. They are used to check if a planned system fulfills the requirements that the expected traffic poses. The metrics are mostly perceived as part of the service quality by the passengers. Many of them are linked together and influence each other.

- The **passenger waiting time** describes the amount of time between the arrival of a passenger at the landing floor and their entry into the elevator cabin [Hak03, p. 7][Sii97, pp.8-9].
- The **passenger ride time** measures the time between the entering of a passenger into the cabin and them exiting the cabin on the destination floor [Sii97, pp.8-9].
- The **total journey time**, also called *total service time* [Bee15, p. 10] or *time to destination*, describes the total time a passenger spends in the system and is calculated by the sum of waiting and ride time [Sii97, pp.8-9].
- The **round trip time** describes the time it takes for a single elevator to collect passengers at the lobby, deliver them to the upper floors and return to the lobby. This measure is critical in the up-peak traffic [Sii97, pp.8-9].
- The **handling capacity**, also called *5-minute interval* [Ung15, p. 194], describes the maximum number of passengers that can be transported (into the upper most floor) during up-peak traffic [Sii97, pp.8-9]. It is dependent on the round trip time. In theoretical considerations a maximum utilization of 60-80% of the capacity is used for this scenario [Ung15, p. 194][Hak03, p. 7].
- The **interval time** describes the time between departures of any cabin from the lobby and influences the time a passenger hat to wait at the lobby [Sii97, pp.8-9].

- The **hall call time** describes the time between the issuing of a hall call by a passenger and the cancellation of the call. The call is canceled when an elevator is scheduled to serve the call floor and hence decelerates to stop at the floor [Sii97, pp.8-9].
- The number of **total stops** necessary to serve a specified amount of passengers (or within a given time frame) is an indicator for the efficiency of the scheduling algorithm [Ung15, p. 194].
- The **energy consumption** per ride or per time interval is a measure of economic interest. However, it does not influence the perceived service quality.
- The **capacity utilization** of the cabins in spacial and weight dimensions is an interesting metric to observe, since usually only 60-80% of the capacity is used at a time [Ung15, p. 194] [Hak03, p. 7]. This utilization can be even lower when large objects are present. The average utilization is further influenced by empty rides.

For each of the metrics, it is common to calculate statistical figures, such as average, mean, minimum, maximum and standard deviation, which also can differ depending on the traffic situation. These calculations can be based on theoretical consideration about the physical parameters of the system [Ung15, p. 194], simulations or heuristical observations in real buildings.

When the group controller schedules the elevators in a system, the order of dispatches influences the metrics listed above. The elevator allocations are calculated by optimization of a cost function which takes the metrics into consideration. Usually, the waiting time is optimized, but also the ride time can be optimized in order to increase the handling capacity [Sii97, p. 10].

2.2.6. Optimization Approaches by Simulation

Active research goes into simulating elevator systems in order to test out new control strategies. There are approaches to model existing or planned elevator systems based on their physical properties and analyze them using either with closed mathematical formulations or to simulate their behavior using computers [Bee15, p. 6].

When implementing a simulation it can be done as a deterministic or stochastic model, which operate on continuous or discrete data. For a computer simulation, the discrete stochastic model is favorable, since it can deal with discrete events, which deal with the interaction of entities in the simulation.

An industrial grade elevator simulator is developed by *Peters Research Ltd* [Pet18]. KONE offers a complete toolbox for system planning, which includes traffic calculations [KON16]. Unger [Ung15, p. 193] lists a simplified mathematical approach to finding out the performance values of an elevator system.

3. Design and Methodology

This chapter describes which research method is used to design a solution that answers the initial questions posed in the first chapter. The considerations that contribute to the design of this solution are outlined and form the foundation for its implementation in the next chapter.

3.1. Employed Methodology

Preceding to the practical part of this thesis, the method to follow along shall be laid out here. One suitable method is the general concept of *design science research*, also known as *constructive research*, since it involves the development and evaluation of artifacts to solve domain specific problems. It, therefore, conveys an improved practical relevance in comparison to purely descriptive research methods, while the outcomes still create scientific knowledge [DLA15, p. v]. The conducted research must design an artifact, such as a construct, model or method, that solves a relevant problem in a specific field, which is evaluated regarding its utility, quality, and efficacy. The performed research should be based on rigorous scientific methods and contribute to the current theoretical body of knowledge. The conducted design process must take into account the practical environment it is executed in and use the available resources. A conclusion of the project for both technology-oriented as well as management-oriented audiences should be presented in the end [DLA15, p. 70].

Figure 3.1 displays the steps that are found in the process of design science research. Those are the steps that guide the execution of this thesis. The awareness and relevance of the given problem have already been given in section 1.1.

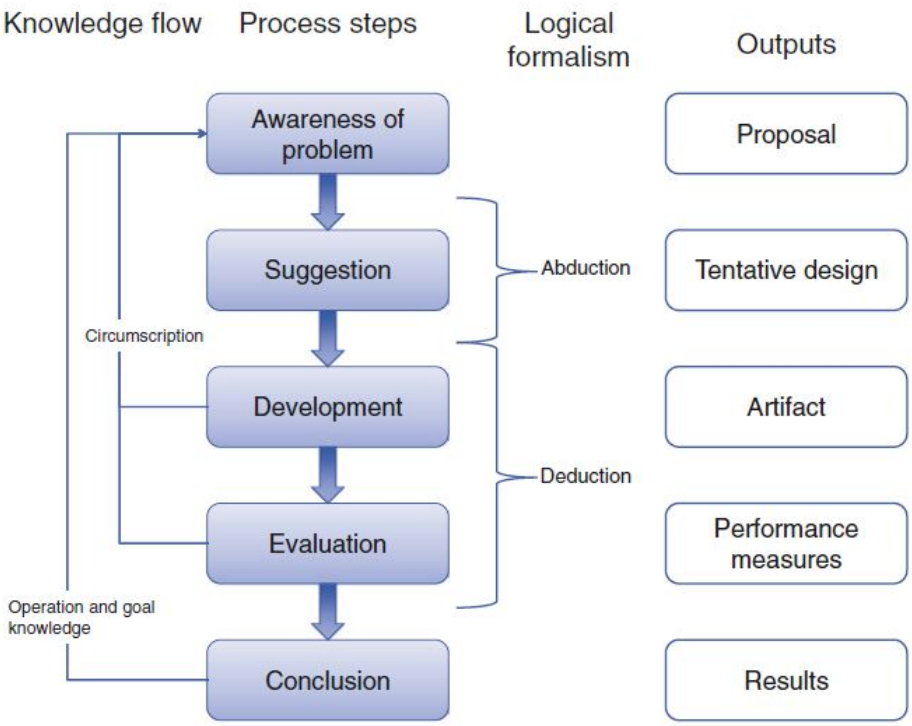

Figure 3.1.: Process steps and outputs of design science research. Reprinted from Dresch, Lacerda, and Antunes Jr [DLA15, p. 83]

Looking back to the original research questions (see section 1.2) two focuses have been identified for this thesis:

- Which computer vision techniques can be used to detect persons and objects within an elevator car and in front of the lift and estimate their spacial volume?
- How can the information acquired by a system that uses such methods constitute to the optimization of algorithms that control the movement plan for elevators?

To answer the two questions, the conducted research is therefore twofold. The first part involves the design and experimental implementation of a system to gather passengers volume data in elevators. The second part deals with the integration of this information into a scheduling algorithm. Outlined below are the steps that are taken to suggest a tentative design for each, develop the artifacts and evaluate them.

For the design and implementation of a vision system to gather volume data of passengers, the following steps are performed:

1. Find a typical or actual elevator control system architecture and define the positions to integrate the necessary components for a vision system into it.
2. Define the exact type and structure of the information that the vision system should be able to detect.
3. Find an algorithmic approach to passenger detection and volume estimation that is suitable to generate the required data.
4. Match the approach with a possible hardware setup to perform tests with it.
5. Test the proposed detection system in a real-world experiment that includes exemplary footage from within a cabin.
6. Evaluate the results of the test for the functionality and effectiveness of the employed system.

For the integration of the passenger data into an existing scheduling algorithm the following steps are performed:

1. Define a suitable elevator configuration that, could possibly benefit from the information that the vision system can provide. Find a typical or actual control algorithm that is used for such a configuration.
2. Adapt the scheduling algorithm to use the additional information.
3. Compare the two versions of the algorithm in order to determine their effectiveness regarding a suitable metric. A simulation is used to perform this comparison.

3.2. Visual System

The section ahead describes the design of a camera-based visual system that is able to capture the volume of passengers that are using an elevator system. in order to design this system, the approach described in the previous section is followed. First, the integration into a typical elevator system is described, then the detection capabilities of the system are specified. Building on that, an approach to the volume detection

is constructed. In the next chapter, an exemplary implementation of this approach is created.

The *Otis Elevator Company* hold patents, which describe a similar but more extensive elevator control system [Lin+11] [Wan+16]. The two patents describe systems with a much broader scope. They focus on tracking individual passengers over their whole journey and use this data for scheduling, door control, and security [Lin+11]. By generating a „traffic list“ from objects recognized in a depth map of they extended this concept also to volume tracking [Wan+16].

3.2.1. Proposal for Integration into Elevator Control System Architecture

There are two places in an elevator system, where the addition of a camera system can create information about passengers: inside the cabin and in front of the lift in the main or floor halls. Cameras inside the cabin could determine, if passengers are currently occupying the space in it and if additional large cargo objects are inside. Cameras in the hall on each floor can observe passengers, that are about to perform a hall call. However since this paper explores the possibilities of volume detection, the information gathered inside the cabin is focused.

In the context of an elevator system, the cameras inside the cabin, combined with a processing system, are just an additional cabin sensor. Therefore the system can be integrated into the electronics of the elevator cabin and is attached to the elevator controller, which can interact with it to make use of the gathered data. If the visual system should be integrated into a multi-cabin system, the elevator controller can forward the information from the visual system to the group controller. Figure 3.2 visualizes the proposed integration.

3.2.2. Output Data Definition

It is the purpose of the visual system to provide information about the volume of passengers and cargo objects. Therefore it should generate the following pieces of data:

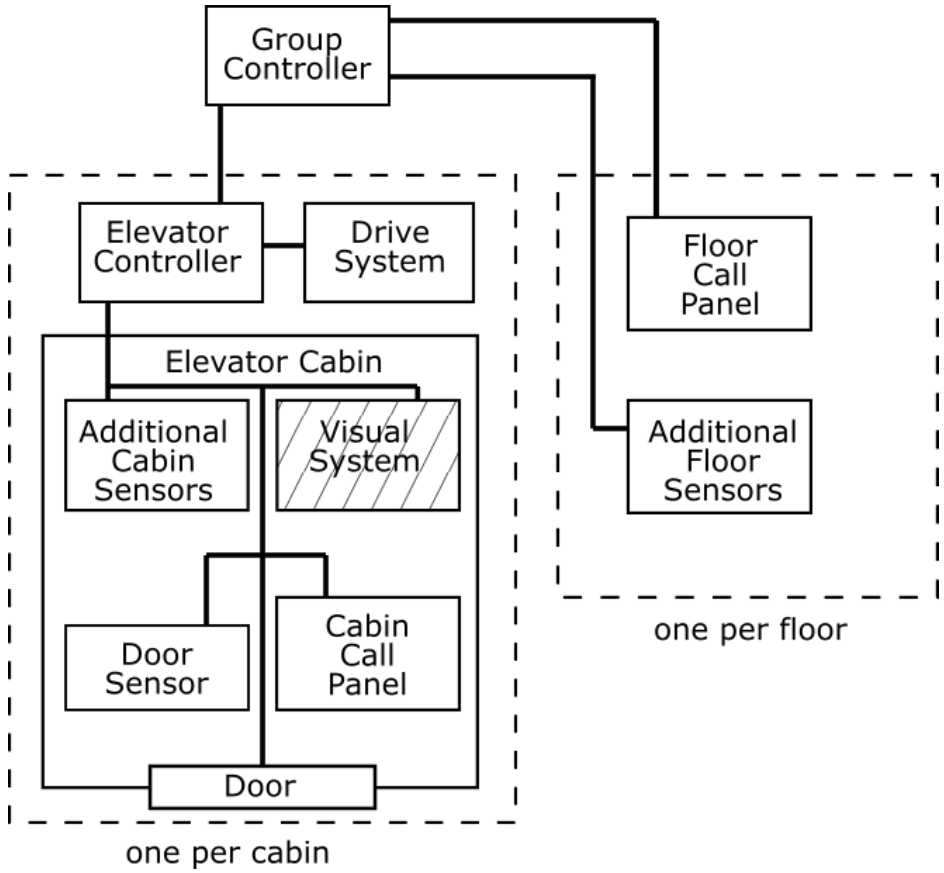

Figure 3.2.: Elevator system extended with visual system. Addition to figure 2.10

- The total number of passengers and cargo objects inside the cabin.
- The approximate volume of each passenger or cargo object.
- The approximate dimensions of each passenger or cargo object.

The system should operate in near real time, meaning the output data should match the current situation inside the cabin. Only a short time may be taken up for processing the image data. Otherwise, the situation inside the cabin could change significantly during the processing time, which would make the information gathered by the system less meaningful. In an elevator, this change can happen in seconds, for example when passengers are entering and leaving the cabin.

3.2.3. Detection Technique

The technique to generate the data that which has been defined above, is centered around the reconstruction of the volume that is occupied by the passengers and objects inside the cabin. Figure 3.3 displays the general concept of this technique. The left side of the figure exhibits the data that is input for the system, the right side shows, which steps are taken to process this data inside the system. First the video capture from the cameras inside the cabin is read, which gets processed and combined with the position and calibration data of the cameras in order to create a reconstruction of the volume inside the cabin. This volume representation is then used to take measurements of the passengers and objects inside the cabin. The next paragraphs describe each of these steps in more detail.

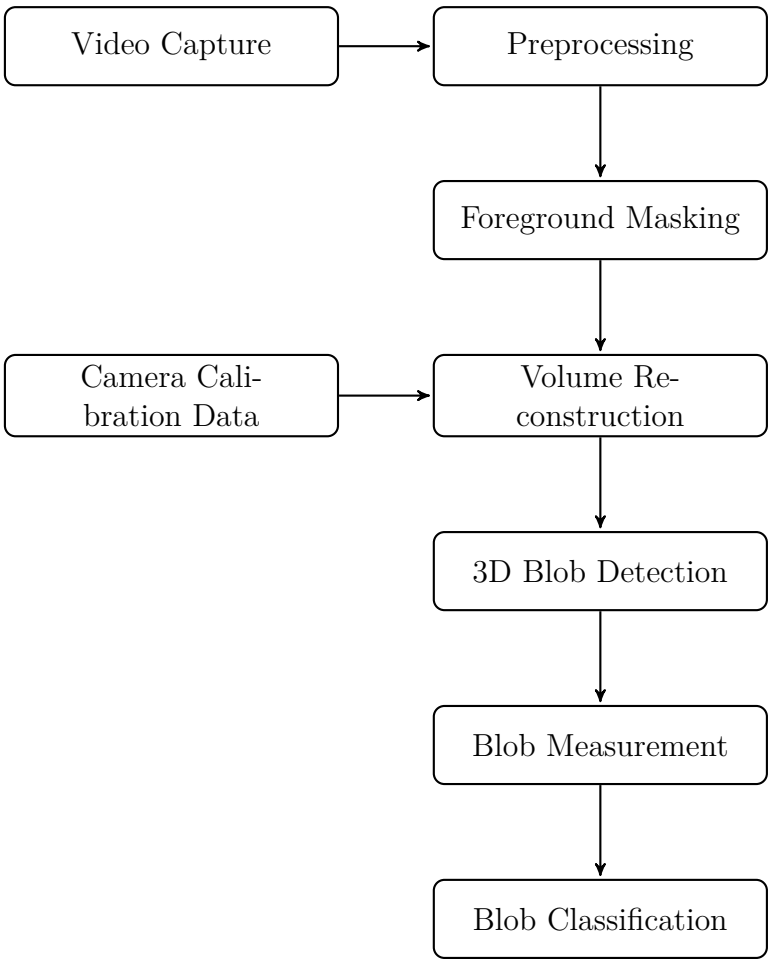

Figure 3.3.: Volume detection technique

Video Capture

Inside the cabin, multiple cameras are mounted. The cameras are positioned on the walls of the cabin, such that they observe the center of it. On each wall, one can be mounted. Where it is not possible to place the camera on the wall, for example on the side of the doors, it is also possible to position the camera in the ceiling corner. This way the space in the cabin is viewed from multiple angles, which enables ever position in the cabin to be observed by cameras. Using cameras of the same model and optic configuration makes the implementation easier. The cameras are connected to the processing unit of the visual system and provide a lifestream of image frames to it.

Preprocessing

As the first active step, the images of all the cameras are read into the memory of the system as 2D color images with the native resolution of the cameras. In order to decrease the computation power needed for the next steps, it is possible to *scale them down* to a smaller size. Furthermore, the camera images typically contain noise, which can be countered by applying a *Gaussian blur* filter so it. The convolution kernel G shown in figure 2.3 is a candidate, but also Gaussian kernels of dimension 5×5 can be used.

Foreground Masking

To find out where passengers and objects are visible in each frame from each camera the next step is to find a foreground mask for each of the images. Since the cameras are static and do not change their position or angle in the cabin, each camera always films the same section of the elevator. The perceived background therefore is static except for changes in lighting. Those can occur when the cabin doors are opened or passengers block the lamps in the cabin. In contrast to the background, the passengers in the cabin move over time. The static background with moving passengers allows the method of *background subtraction* to be used. Either static or dynamic techniques

can be used, but the dynamic approach of the MOG is favorable, since it can react to changes in lighting.

The resulting foreground masks might be noisy and can have pixel-sized artifacts. These originate from noise present in the processed image and small changes between two frames in locations, where actually no passenger is present. In order to remove this noise in the binary foreground mask, the morphological *OPEN* operation is used. It is a sequential application of the *ERRODE* and *DILATE*, which first removes all edge pixels and then extends the remaining edges again. This way, single active pixels are deleted, while other structures are preserved. After this, the *DILATE* operation is applied again to close potential holes in the masks.

Camera Calibration

In this step, the internal and external parameters for all cameras are gathered. Since the camera settings are not altered during the operation of the system, these parameters only need to be determined once and can then be reused. The following parameters must be obtained for each camera, which is also explained in figure 2.5:

- The internal camera matrix A, which includes the focal length and the point of the image center. They can be obtained from calibration footage that features an identifiable object of known geometry, such as a chess board.
- The external rotation matrix R and the translation vector t, which can be obtained by measuring the position and viewing direction of the camera.
- The internal distortion coefficients of the camera, which also can be obtained from calibration footage.

Additionally to the camera parameters, the dimensions of the elevator cabin need to be known to be later able to convert pixel units into world units of meter. The parameters obtained in this step are used in the next step to guide the volume reconstruction.

Volume Reconstruction

From the camera calibration and position data and the foreground mask of each perspective, it is possible to reconstruct the volumetric data of the interior of the cabin. The technique of *volume intersection* is used for this. Since the purpose of this system does not require a photo-realistic, the simple voxel-based method can be used. It would be possible to reconstruct the full geometry and color of the room using a point matching algorithm or the more advanced space carving. However, it is not necessary to reach such a level of detail for the purpose of this system.

In the volume intersection technique, first, the whole volume of the elevator is described as a 3D image of $X \times Y \times Z$ voxels. The number of voxels is chosen rather low, in total between 10,000 and 100,000. If a larger amount is chosen, results with higher accuracy could be obtained at the expense of processing power. The voxels describe an evenly distributed cube grid, that fills the whole elevator cabin. In order to perform the volume reconstruction, the position of every voxel is projected into the viewport of each camera by applying the formula given in figure 2.5 to it with the parameters obtained from the camera calibration in the step before. This projection also needs to take the transformation from world coordinates in meter into pixel coordinates in the camera into account. Now a 3D binary image $F_{vol} \in \{0,1\}^{X \times Y \times Z}$ is created, by checking for every voxel if it lies within the viewport of each camera and is projected to a foreground pixel in it. If this is the case for all cameras, the voxel is considered to be part of an occupied volume and hence marked as 1, otherwise as 0.

3D Blob Detection

The volume intersection in the previous step yielded a 3D binary image $F_{vol} \in \{0,1\}^{X \times Y \times Z}$, where voxels marked with 1 indicate the presence of an object or passenger at the position of the volume element. Where ever a passenger or object takes up space in the 3D image, a 3D *binary blob* is present, which is a collection of spacial coherent voxels that are positively labeled. In order to find distinct entities in that image, the *connected component labeling* method can be used. Initially introduced for 2D images, it is possible to extend the common label-propagation and label-equivalence algorithms to work on a binary image of higher dimensions [He+17,

p. 39]. Since X, Y, Z have been chosen to be of low resolution, it appears appropriate to use a label-propagation approach.

In a simplified version of this approach, the 3D image is scanned in a raster, and each encountered voxel, which is labeled as 1 in F_{vol} and has not yet been labeled by the connected component algorithm, it assigned a new unique label. Furthermore, all neighboring voxels, which are labeled as 1 in F_{vol}, are also labeled with this label. When a voxel is encountered, which is already labeled, this label is also propagated to its neighbors that are labeled as 1 in F_{vol}. The basics of this approach work exactly like in two dimensions, however, the definition of neighboring voxels is altered to match three dimensions. In general, all 26 voxels in the cube around a voxel are considered neighbors. It is also possible to only consider the 6 voxels, that share a face with the center voxel.

The described algorithm performs a mapping $\{0,1\}^{X\times Y\times Z} \rightarrow \mathbb{N}^{X\times Y\times Z}$, where the voxel values of the output represent the unique label of the blob they belong to, or 0 in case that the voxel does not belong to any blob. Has a complexity of $\mathcal{O}(XYZ)$ and operates without additional memory. The algorithm can be improved by performing the label-propagation along the surface of the 3D blobs.

Blob Measurement

The previous step yielded a 3D image F_{blobs} consisting of labeled binary blobs. For each blob, the volume and its length in the X, Y and Z axis are calculated. The volume is calculated by summing up the volume of the voxels, which make up the blob. Even when no concrete volume for each voxel is known, this volume is still proportional to the total volume of the containing image F_{vol}. The length in a particular axis can be found by calculating the distance of the two most extreme voxels in that axis that belong to the blob. Figure 3.4 describes the calculations for this measurements for a blob with label l.

$$P_l = \{(x, y, z)|F_{blobs}(x, y, z) = l\}$$

$$\Delta X_l = max(\{x|(x, y, z) \in P_l\}) - min(\{x|(x, y, z) \in P_l\})$$
$$\Delta Y_l = max(\{y|(x, y, z) \in P_l\}) - min(\{y|(x, y, z) \in P_l\})$$
$$\Delta Z_l = max(\{z|(x, y, z) \in P_l\}) - min(\{z|(x, y, z) \in P_l\})$$

$$V_l = ||P_l|| \cdot V_{voxel} = \sum_{P_l} V_{voxel}$$

Figure 3.4.: Calculation of 3D blob measurements

Blob classification

With the aid of the previously generated measurements, each blob can be classified to find out whether it reassembles a passenger, a group of passengers, or a large cargo object. Blobs with a volume smaller than a specific threshold T_{min} are probably erroneous artifacts and can easily be ignored. If the volume is human like and the proportions of the blob are higher then wide, it can be classified as a human. If the volume is larger than a threshold T_{max} or if the blob has a greater width than height, it can be either classified as a large object, or as a group of passengers.

An alternative to this static approach would be to apply machine learning algorithms to find out the classification of a blob. For this, a pre-classified learning data set needs to be available, which might not be feasible.

3.2.4. Test Arrangement and Validation Strategy

To show the general feasibility of the approach described above, a real live test is performed. For this test, an exemplary office elevator is used. Inside it, multiple cameras are mounted at the walls, all viewing the center of the cabin. The cameras do not capture a live stream that is processed but record a video for later processing. Four cameras are used in the test configuration. One mounted to the center of each wall of the elevator, in a height of 150 cm. These cameras are therefore positioned at a right angle towards each other. An additional camera is mounted over the entry door within the elevator, also facing the center of the elevator.

During the performance of the test, there are multiple scenarios of passengers residing in the cabin:

- Empty cabin
- Single passenger in multiple positions in the cabin
- Multiple passengers in multiple positions in the cabin
- Single passenger with a large cargo object in the cabin
- Multiple passengers with a large cargo object in the cabin

In the test, each of those scenarios is conducted for about a minute or less.

A typical approach to evaluate the performance of a quantitative algorithm in computer vision is to compare it against the *ground truth* present in the observed scene. The ground truth is a description of the scene in the same format which the algorithm produces, however it is generated by inspection of the original scene by a human. If the description consists of data for which an order exists, this order can be used for evaluation of the capability of the system. In general, however, the data that is compared might be arbitrarily complex and therefore it is not always possible to perform the comparison quantitatively. Rather a human has to decide, how „good“ the produced results are.

3.3. Scheduling Algorithm

This section describes the design of a scheduling strategy, which makes use of the information provided by the visual system designed above. The strategy is tailored to meet the needs of a building, in which passengers and cargo objects are sharing a lift. The strategy is then compared against conventional scheduling algorithms by designing a simulation of an elevator system that runs those algorithms. This section follows the line of action defined in section 3.1. In the next chapter, the results of this simulation are explored.

3.3.1. Suitable Elevator Configuration

Since the camera system is suitable to detect passengers as well as cargo objects, it seems reasonable to apply the passenger information to a scheduling algorithm of an elevator system which conveys both of them. This holds the opportunity to have a bigger impact on the performance of the system, in contrast to applying it to a passenger-only lift.

When looking back at the categorization of elevators, cargo lifts meet this specification. Typically an office building has one or multiple cargo lifts depending on its size, but grouping them is uncommon [BA16, p. 167]. A weight capacity of at least 1,600 kg is typical [BA16, p. 167], but also capacities of 5,000 kg are possible [KON17]. Typical application areas for cargo lifts are hospitals, where patients and beds are moved, auxiliary lifts in office buildings and multi-storey industrial buildings. The typical control algorithm is collective control or sequential control [BA16, pp. 238, 244]. For this comparison, the sequential control and collective control will be taken as a baseline.

3.3.2. Adaption of Scheduling Algorithm

The proposed visual system is able to detect, whether a passenger or a large object is present in the cabin. This information holds the possibility to be used in an appropriate elevator scheduling algorithm. Therefore a new control strategy is proposed here, which will be called *adaptive control* in the following. The strategy is a combination of collective control and sequential control. It follows these simple rules:

- When no cargo item is present in the cabin, the elevator is operated in collective mode and picks up and delivers passengers according to the collective control strategy.
- When there is a cargo item present in the cabin, deliver it directly to its destination car call, regardless of other car calls and hall calls.

Since those rules make use of the information about the destination floor of the cargo object, the necessity to associate the cargo item to its car call within the system is

implied. This association can be determined from the combination of the video data and car call data. When the cargo object is first detected, one of the next car calls made must be from the object or the passenger escorting it.

3.3.3. Preparation for Simulation and Validation Strategy

Scheduling Algorithms

As discussed before the introduced *adaptive control strategy* is evaluated using a simulation of an elevator system. It is compared against the *sequential control strategy* and the *collective control strategy*. In order to guide the implementation of the simulation, the concrete actions that these three strategies take to move an elevator are laid out in the following.

The **sequential control strategy** delivers one passenger at a time and serves hall calls sequentially. Its behavior can be described with the following actions:

1. Update the queue of hall calls by appending all active hall calls that are not yet in the queue.
2. If there is an active car call:
   2.1. Move to is respective floor.
   2.2. Open the doors.
   2.3. Unload the passenger.
   2.4. Close the doors.
   2.5. Clear the car call.
3. Otherwise, if there is an hall call in the queue:
   3.1. Move to the floor of the first hall call in the queue.
   3.2. Open the doors.
   3.3. Load one passenger into the lift.
   3.4. Delete the hall call from the queue.
   3.5. Update the car call to be the destination floor of the passenger.

The **non-directional collective control strategy** collects and drops off passengers along the way while serving the hall and car calls. Its behavior can be described with the following actions:

1. Let the car calls be all the destinations of all passengers present in the lift.
2. Let the hall calls be all the unique floors where passengers wait for the elevator
3. If the car calls contain the current floor:
    3.1. Open the doors.
    3.2. Unload the passengers for this floor.
4. Otherwise, if the hall calls contain the current floor:
    4.1. Open the doors.
    4.2. Load all passengers on the floor into the lift, which fit in it.
5. Otherwise, if there is another car call or hall call in the current direction of travel:
    5.1. Close the doors.
    5.2. Move to the closest car call or hall call in the direction of travel.
6. Otherwise reverse the current direction of travel.

The **adaptive control strategy** introduced before makes a decision based on whether a cargo object is currently located inside the cabin. Its behavior can be described with the following actions:

1. If there is a cargo object in the elevator:
    1.1. Close the doors.
    1.2. Move to its destination floor.
    1.3. Open the doors.
    1.4. Unload the cargo object and other passengers with this destination.
    1.5. Reset the current travel direction.
2. Otherwise, follow the collective control strategy.

Description	Value	Unit
Building data		
Number of floors	8	
Interfloor distances	3.5	m
Lift data		
Number of lifts	1	
Rated weight load	5000	kg
Rated passenger capacity	65	
Rated Speed	1.6	ms^{-1}
Door opening time	1.5	s
Door closing time	1.5	s
Flight time single floor	8	s
Passenger Data		
Arrival rates for floors	$(f,t) \mapsto \frac{7}{300}$	$\mathbb{N} \times s \to s^{-1}$
Weight and volume distribution	see table 3.2	$kg \times m \times m \times m \to \mathbb{R}$
Transfer time into / out of car	2	s
Floor bias	unbiased	$\mathbb{N} \times \mathbb{N} \to \mathbb{R}$
Cargo Data		
Arrival rates for floors	$(f,t) \mapsto \frac{2}{300}$	$\mathbb{N} \times s \to s^{-1}$
Weight and volume distribution	see table 3.2	$kg \times m \times m \times m \to \mathbb{R}$
Transfer time into / out of car	5	s
Floor bias	unbiased	$\mathbb{N} \times \mathbb{N} \to \mathbb{R}$
Simulation Parameters		
Simulation Period	1	h
Time slice	0.1	s
Number of simulations	1000	

Table 3.1.: General configuration for the comparative simulation

Type	Width / cm	Length / cm	Height / cm	Weight / kg
Passenger	50	25	160	60
	50	25	174	65
	50	25	180	80
	70	30	190	110
Cargo Item	250	150	150	1000
	250	150	150	2000
	250	150	150	3000
	250	150	150	4000
	250	150	150	4900

Table 3.2.: Prototypes for traffic items in the simulation, chosen at equal likelihood

Simulation Parameters

In the section beforehand, the building type that acts as a base for the simulation has been described: a single elevator configuration, shared by passengers and cargo items. Barney and Al-Sharif [BA16, p. 347] describe, which parameters need to be determined in order to run a simulation. Table 3.1 gives an overview of the parameters that were chosen for this simulation. Some of the values are taken from elevators manufactured by KONE [KON17], other values are taken from Barney and Al-Sharif [BA16, p. 349]. Especially the lift parameters are based on the sources. In paragraphs ahead, the other parameters are discussed more thoroughly.

Table 3.1 features two sections that describe passenger data and cargo data. The elevator can convey both, passengers and cargo objects. Therefore, in the simulation they are both considered to be *traffic items*. Even though, not all cargo objects can move on their own, and need to be carried by a passenger, for the purpose of this simulation, they are considered to behave identical as passengers. In general, traffic items arrive at a certain time on a certain floor and have a defined destination floor. The only aspect in which they differ is their arrival rate, in that more passengers than cargo objects arrive, and their physical properties.

The simulation is performed using discrete events. Therefore a discrete, strictly increasing integer clock is used to mark the timing of events. Each of these discrete steps is called a *tick*. The duration of such a tick is determined by the *time slice* parameter.

The *arrival rates* of passengers and cargo objects is chosen constant over the who time, which means that at any point in the simulation the chance that a given amount (0, 1, 2, ...) of traffic items arrives is fixed. Furthermore, the arrival rate on each floor is assumed to be the same. This behavior can be modeled with a Poisson distribution

$$A(k) = e^{-\lambda}\frac{\lambda^k}{k!}$$

where $A(k)$ describes the likelihood of a specific number of traffic items arriving within a time slice (duration of a tick) in the whole building. The parameter λ is chosen to be the multiple of the time slice and the arrival rate. This constant arrival model simplifies the simulation and effectively simulates an *inter-floor traffic pattern*. It would also be possible to model an up-peak or down-peak traffic pattern, however, this would make the modeling more complex.

The *weight and volume distribution* for both types of traffic items is given by choosing from predefined *prototypes* (or stereotypes). Every passenger or cargo object in the simulation has the same weight and volume properties as one of the prototypes. Table 3.2 lists the prototypes that are defined for this simulation. A passenger in the simulation is created from any of the passenger prototypes with the likelihood of $\frac{1}{4}$, a cargo item from any of the cargo item prototypes with a likelihood of $\frac{1}{5}$. Choosing from prototypes rather than a continuous distribution helps to simplify the simulation implementation.

The *floor bias* for both types of traffic items is *unbiased*. There is no precedence in arrival and destination floor, every journey appears with the same likelihood. However, a journey where arrival and destination floor are equal, can not occur. The probability of any given trip to occur, is described as a probability function

$$b(f_{arrival}, f_{destination}) = \begin{cases} \frac{1}{N^2-N} & f_{arrival} \neq f_{destination} \\ 0 & else \end{cases}$$

where N is the number of floors present in a building, $f_{arrival}$ and $f_{destination}$ are an integer number between 0 and $N-1$, which describe the arrival and destination floor number.

Simulation Measurements

To evaluate the simulations, several measurements are taken to compare the scheduling algorithms. For the each of the measures the minimum, maximum, average and standard deviation σ values are calculated over all the simulation runs of each scheduling algorithm. The following measurements shall be taken:

- Total passenger in the simulation
- Total cargo objects in the simulation
- Stops of the lift
- Delivered to their destination
- Cargo items delivered to their destination
- Waiting time
- Ride time

4. Implementation and Evaluation

In this chapter the implementation of the two artifacts identified and designed in the previous chapter is executed. First, an attempt is made to implement the visual volume detection system and test it out in a real-world test environment. Then simulation of an elevator system is implemented to compare tree scheduling strategies and the results of this simulation are evaluated.

4.1. Visual System

4.1.1. Implementation with OpenCV and Python

The design proposal given in section 3.2.3 serves as a base for the actual implementation of the visual system. The implementation is written in Python 3 in version 3.6 [Pyt18b] with bindings to OpenCV 3.4.2. Python was chosen as the implementation language since it provides a higher abstraction and better ergonomics than C++. Furthermore, the author is proficient in it. The code of the `main` procedure that has been implemented can be found in listing B.1 in appendix B. The whole program, including the processed camera footage, can be found on the attached CD or DVD.

There exists a C++ implementation of a perspective correct volume intersection, which however is used for a more detailed reconstruction of a small statue, which is photographed 36 times and rotated around its upward axis [XW13]. This C++ implementation gave a first reference for the implementation in Python.

The interior volume of the cabin is modeled with a voxel image of dimension $20 \times 25 \times 30$ with a total of 15000 voxels. The chosen numbers are a compromise between accuracy and detail.

The camera capture is implemented by reading from a video file, which is created during the test execution. The preprocessing of the camera images and the foreground masking is implemented as designed. For blurring the image, the OpenCV function `cv2.GaussianBlur(img, kernel_size, std_deviation)` is used. In order to perform the background subtraction, the function `cv2.createBackgroundSubtractorMOG2()` is used to create an instance of a MOG based background subtractor for each input camera. The resulting foreground mask is then post-processed by applying `cv2.threshold(mask, 254, 255, cv2.THRESH_BINARY)` and `cv2.morphologyEx(mask, cv2.MORPH_OPEN, open_kernel` and `cv2.dilate(mask, dilate_kernel, iterations= 10)` in series to the mask. Interactive sliders are used to adjust the strength of blurring the image and the level of morphological opening and dilating by altering the respective kernel used. This creates the opportunity to find the optimal value for each of the settings by hand. The described foreground masking operates therefore as planned.

In contrast to the planned design, however, the implementation here uses an even more simplified version of the volume intersection. Instead of using a perspective projection of the cabin to the camera images, an orthogonal projection is used. This means the image is assumed to be taken without perspective distortion and with an image sensor that spans over the whole wall. This is obviously physically incorrect, as usual cameras are closer to pinhole cameras than to orthographic cameras. But this simplification enables a more straight forward implementation of the volume intersection, as only images with right angles to the room need to be considered. No external and internal camera matrix is therefore needed.

Listing 4.1 shows the code for the orthogonal volume intersection. First, the three foreground masks of the three perspectives are scaled. Then, every voxel position in the room is checked in a list comprehension. To do so, the coordinate `[u, v, w]` is iterated over the Cartesian product of all possible voxel coordinate values of the width, depth and height axis of the room. The voxel position is then „projected“ orthogonally into the scaled foreground masks, which is essentially only an array index into the masks, where `w` is the primary axis of the image, and `v` or `u` is the secondary axis of the image (depending on the side). The voxel position is then scaled from voxel coordinates to room coordinates. The list comprehension yields an array of

voxel positions, where each of the voxels is part of the volume description.

```
# draw masks A B C to the side, orthogonal
scaled_masks = draw_3d_side(side_3d, bgs_pp_masks_b)

# create points of orthogonal intersection of A B C
vol_poss = numpy.array([ \
    [ \
        u * ROOM_SCALE_MPVX['width'],\
        v * ROOM_SCALE_MPVX['depth'],\
        w * ROOM_SCALE_MPVX['height']\
    ] \
    for u in range(ROOM_DIM_VX['width']) \
    for v in range(ROOM_DIM_VX['depth']) \
    for w in range(ROOM_DIM_VX['height']) \
    if  scaled_masks[0][w][v] == 255 \
    and scaled_masks[1][w][u] == 255 \
    and scaled_masks[2][w][v] == 255 ])
```

Listing 4.1: Orthogonal volume intersection implementation

As mentioned, this implementation uses orthogonal projection. In order to turn it into a perspective projection, instead of using a direct array look-up
`... scaled_masks[0][w][v] == 255 ...` a projected look-up can be made:
`... masks[0][project_and_scale(CAM_CALIBRATION[0], [u, v, w])] == 255`
where the function `project_and_scale(c, p)` takes the internal and external calibration data `c` of a camera and a voxel position `p` and returns a projected and scaled pixel position, that can be used to index into the foreground mask of that camera.

The program depends on several libraries. The Python Software Foundation [Pyt18a] provides a repository, in which all of these dependencies can be found.

- `numpy`: Provides typed n-dimensional arrays to store image data
- `scipy`: Dependency of scikit-image
- `matplotlib`: Used for plotting of 2D graphs
- `opencv-python`: Open Computer Vision library
- `opencv-contrib-python`: Provides additional functions for OpenCV
- `PyYAML`: Used for parsing and writing of YAML Ain't Markup Language (YAML) files in the camera calibration
- `scikit-image`: Image processing functions
- `PyQt5`: Dependency of `pyqtgraph`, used to create application windows
- `PyOpenGL`: Dependency of `pyqtgraph`, used for for 3D rendering
- `pyqtgraph`: Used to display real-time 3D graphs

4.1.2. Test Execution

The test was performed as described in section 3.2.4 with the help of two fellow students. The elevator cabin has a dimension of 130 × 165 × 230 cm (width, depth, height). The used cameras are three smartphones, attached to the wall with tape, and a wide-angle camera attached above the door at an angle of 54.5° down. The image resolution of each of the cameras was set to 1280 times 720 pixels. The cameras on the wall were mounted in the center of the wall at the height of 150 cm each, facing the center of the cabin. Multiple configurations of passengers inside the cabin were tested, as described in the test plan. A large carton was used as a cargo object, which was held first by two, then by one passenger.

A camera calibration using a chessboard has been performed before the actual test. The chessboard was filmed by every camera individually from multiple angles. A calibration program by Šmíd [Šmí15] has been used to generate the internal camera matrix and distortion coefficients for the cameras based on the footage taken.

Figure 4.1 shows images of three of those configurations and the preview of the processed images. The top left grey-scale images show the input images captured by the cameras from the right, rear, left and over-the-door position. Under each input image the calculated foreground mask is displayed, where white areas highlight the foreground. On the right of each figure, the detected 3D volume is displayed. The white outline marks the dimensions of the cabin. The red dots mark the position of each camera. The blue areas consist of the foreground masks projected and stretched to the walls. The green cloud represents the points that are part of the orthogonal volume intersection of the the three wall-mounted cameras. This composition image is also saved as a video by generating it for every frame of the input images.

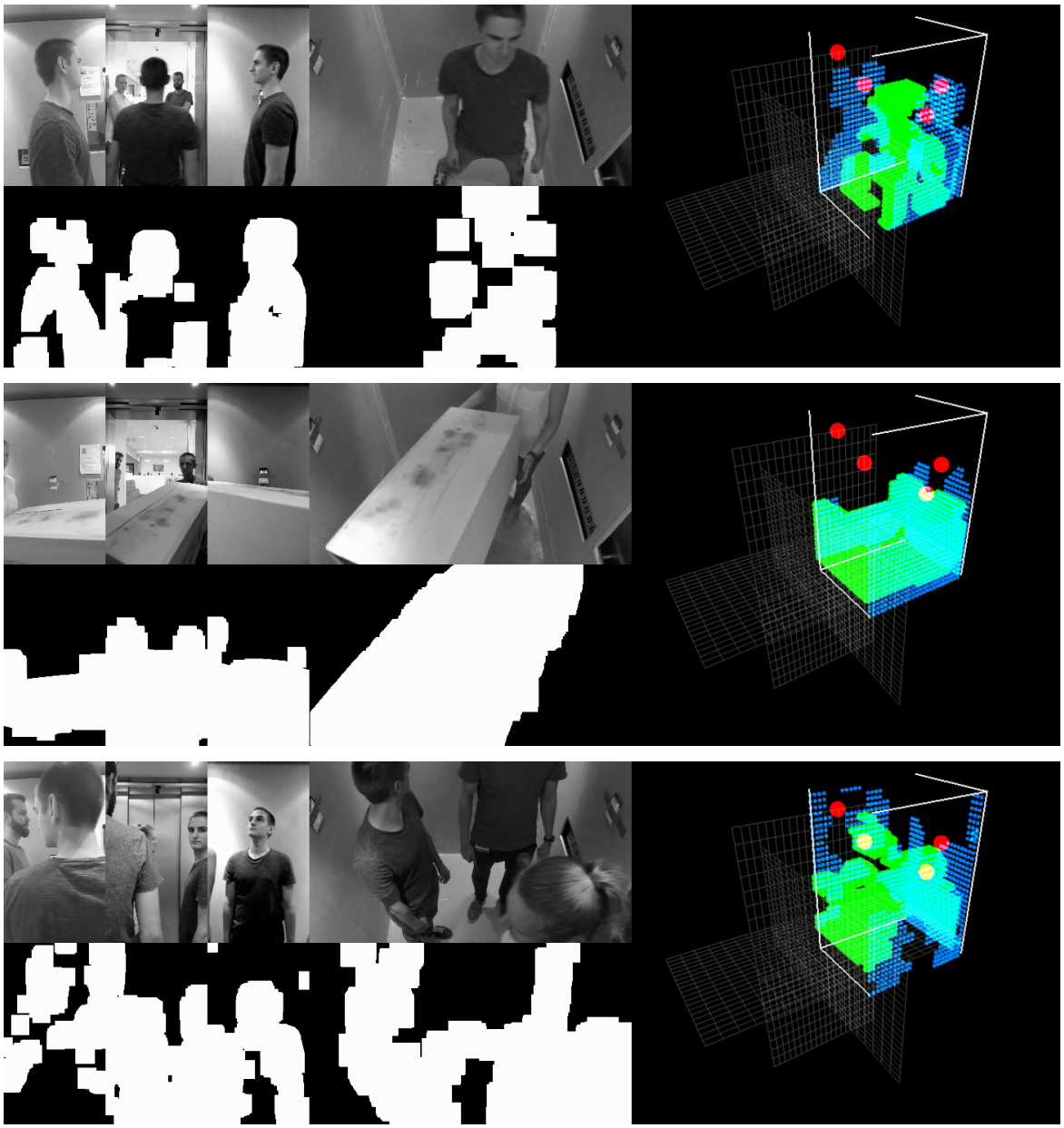

Figure 4.1.: Images of volume intersection test with orthogonal projection. Each preview showing the camera images, foreground masks, and the resulting volume intersection. (a) single passenger (b) cargo object (c) multiple passengers.

4.1.3. Result Evaluation

No ground truth for the volumetric data inside the cabin was recorded during the execution of the test. This means the volume of the test subjects is unknown, as well as their position at every moment of time. Therefore, here a qualitative evaluation is performed.

The video capture works as intended. However, the choice of cameras resulted in a viewing angle, which did not cover the whole cabin. This has the consequence that the corners of the cabin were not recorded a passenger standing there was not registered by the system. A wide-angle camera on all sides might yield better results.

The foreground masking does not produce results of the desired quality. It detects the passengers and cargo objects while they are moving. However, often the center of the subject is not completely detected. The background subtractor recognizes only the outline and moving edges. This leads to „holes“ in the reconstructed volume. On the positive side of the adjustable parameters for the blur of the input image before the background subtraction and the adjustable amount of morphological opening and dilating of the foreground mask served good while experimenting with the footage. In order to address the shortcomings of the foreground masking via background subtraction, combining the technique with other image segmentation methods could be possible.

The volume reconstruction using a simple voxel-based method of volume intersection shows to be feasible in general for the purpose of live volume estimation since no high accuracy is needed. However, the implemented orthogonal version is not entirely correct, as a perspective projection would be needed. Still, it demonstrates the general feasibility of this approach, as the orthogonal projection yields a volume description that is proportional but skewed. Additionally, due to the holes in the foreground masks, these holes are also present in the volume image. This leads to inconsistencies.

The 3D blob detection and measurement has not been implemented as proposed. The blob classification is not feasible if the quality of the volume data is low because of the missing observed corners and the holes in the volumes. The holes lead to a high degree of segmentation. No connected components would be found and the ones found

would be lower than the detection threshold. Figure 4.1 shows the short times, where actually connected blobs are detected. But most of the time the results are worse. The video data can also be found on the attached CD or DVD.

In conclusion: The overall approach is feasible and implementable using OpenCV, however many improvements need to be made for a production-ready system.

4.2. Scheduling Algorithm

4.2.1. Implementation of Simulation

In section 3.3 the *adaptive control strategy* has been introduced and an evaluation via a simulation has been planned. This section describes the details of this implementation.

The simulation is implemented in the *Rust* programming language. Rust is a programming language initially developed by Mozilla. According to its website it is a „systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety“ [Rus18b]. It features an expressive static type system and can be used to build applications that take advantage of modern hardware [MK14]. The rust compiler targets common platforms and operating and compiles source code to native machine code. The language is chosen since the author is proficient in it.

The whole program spans over 880 lines of code. It is not printed here but can be found on the attached CD or DVD. Instead of printing it, its behavior and structure is explained in the following.

The `main` function is the entry point into the program. In it, the general behavior of the simulation is defined. It initializes the simulation parameters and traffic patterns and runs the actual simulation runs. The actions it performs can be described as follows:

1. Initialize the `BuildingParameters b`, `LiftParameters l` and `SimulationParameters s`.

2. Generate `s.simulations` many traffic `TrafficGenerator` instances, which each generates a traffic pattern. A traffic pattern describes the arrivals of traffic items for each tick of the simulation period. A Poisson distribution can be used to determine the number of traffic items that arrive within this tick [Bee15].
3. For each traffic patterns, run the simulation with the three different scheduling strategies. Each simulation is given the same parameters (`b, l, s`) and the same traffic pattern. The procedure to run a single simulation is described below.
4. Collect the results of all simulations and filter out the results of any run, where any of the three strategies produced an invalid result.
5. Calculate the specified metrics by aggregating all the results for each strategy.

The implementation in Rust can be found in listing A.1 in appendix A.

The actions taken in order to run a single simulation are then as follows:

1. Create a new `ElevatorSystem e`, which holds the system state to be altered during the simulation.
2. Create an empty `SimulationResults` metric container, which will aggregate the metrics for this run.
3. While `e.ticks < s.total_ticks()`, repeat the following simulation cycle:
   - 3.1. Ask the control strategy for an action to be performed on the elevator system based on the current state of said system and the internal state of the control strategy. The action can be one of the actions listed as `ControlAction`.
   - 3.2. Perform the given action on the elevator system, thereby updating all fields effected by this action.
   - 3.3. Calculate the time it took to perform this action on the system and advance the ticks accordingly.
   - 3.4. Update the metrics about waiting and ride time for each traffic item that is currently in the floor queues or in the elevator. Also update the metrics about the number of delivered traffic items and elevator stops.

   3.5. Copy the traffic item form the traffic pattern into the floor queues, which arrived in the time between the last tick and the updated tick.

4. Correct the gathered metrics by removing the ride times of traffic items, who did not arrive at their destination yet and waiting times of traffic items, who did not yet enter the elevator.

The implementation in Rust can be found in listing A.2 in appendix A.

The three control strategies are passed to the execution of a single simulation as generic parameters. Therefore, they all implement an interface `ControlStrategy`. A control strategy decides in each tick of the simulation, which action should be taken on the elevator system. The behavior is the same as defined in section 3.3.3, however, the actions such as „move elevator to floor X“ are not performed in one action, but require instructions for the elevator system spanning over multiple ticks. Hence, at each point in time and instruction from the `ControlAction` enumeration is given.

Figure 4.2 shows a class diagram of the simulation program. Rust is not a pure object-oriented language, but rather an imperative systems programming language, which solves generic programming via interfaces. Therefore, the Unified Modeling Language (UML) class diagram is sub-optimal for the accurate representation of a Rust program. It can still be used and gives a good overview of the types and instance variables present in the program. Note that the `Main` class is not a class but a collection of functions. It uses almost all of the types shown, even if no association is indicated in the diagram.

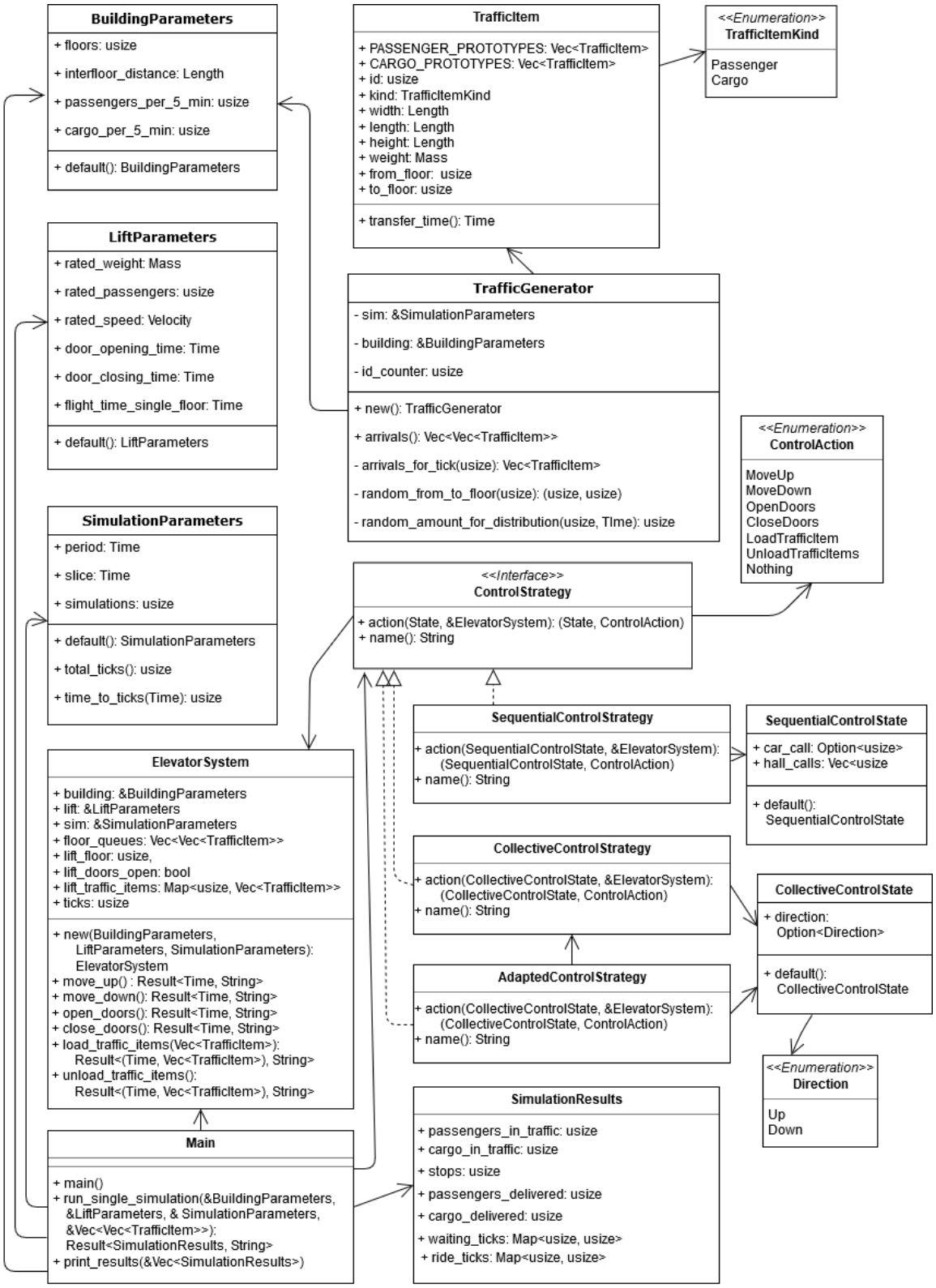

Figure 4.2.: Class diagram for the elevator simulation program

The Rust implementation as a few dependencies, which are libraries used by the program. All of them can be found in the rust package repository [Rus18a]. This list describes their purpose in the program:

- `lazy_static`: Used for initialization of prototype list
- `rand`: Provides randomness functions and common likelihood functions including the Poisson distribution
- `rayon`: Provides data-driven multi-threading capabilities and is used to run the simulations in parallel
- `streaming-stats` (`stats`): Provides functions to create metrics with online algorithms such as average, minimum, maximum and standard deviation
- `uom`: Provides types for units of measurements such as Mass and Time, as well as the respective International System of Units (SI) units such as kilograms and seconds

		Sequential Control	Collective Control	Adaptive Control
Total passengers	**min**	57.00	57.00	57.00
	max	113.00	113.00	113.00
	avg	84.09	84.09	84.09
	σ	8.97	8.97	8.97
Total cargo	**min**	9.00	9.00	9.00
	max	42.00	42.00	42.00
	avg	24.10	24.10	24.10
	σ	4.95	4.95	4.95
Stops	**min**	102.00	93.00	96.00
	max	135.00	158.00	157.00
	avg	118.47	125.75	129.60
	σ	5.22	10.41	9.30
Passengers delivered	**min**	37.00	46.00	53.00
	max	65.00	107.00	108.00
	avg	49.32	78.06	79.67
	σ	4.24	9.45	8.75
Cargo delivered	**min**	3.00	3.00	4.00
	max	23.00	29.00	33.00
	avg	14.13	14.61	19.80
	σ	3.20	3.54	4.38
Waiting time / s	**min**	1.50	0.00	0.00
	max	2873.50	202.60	662.70
	avg	688.09	41.25	53.03
	σ	499.11	32.63	48.98
Ride time / s	**min**	13.00	13.00	13.00
	max	64.00	2180.20	1809.70
	avg	29.57	125.35	102.29
	σ	13.92	160.37	130.77

Table 4.1.: Simulation results for tested scheduling algorithms

4.2.2. Evaluation of Simulation Results

By running the implemented simulation, the results displayed in table 4.1 are gathered. It is visible that the three scheduling algorithms have distinct manifestations of the metrics that are recorded. The sequential control sets focus on low ride time, the collective control sets focus on a balance between wait and ride time while maintaining a high delivery rate. The adaptive control strategy favors the delivery of cargo objects with lower ride times while trading for higher waiting times. The key takeaways, expressed in relative numbers, here are:

- Adaptive control delivers **more cargo items** on average than sequential control (+40.13 %) and collective control (+35.52 %).
- Collective control (+58.27 %) and adaptive control (+61.53 %) deliver **more passengers** on average compared to sequential control.
- Sequential control shows a drastically higher mean waiting time then the other two (+1568.09 %, +1297.55 %). However, it features a lower average ride time (−76.41 %, −71.09 %), which is is only linear dependent on the travel distance.
- Adaptive control shows a slightly higher average waiting time (+30.56 %) but has a lower (−18.40 %) average ride time, which is less spread (−18.46 %) compared to collective control.

In conclusion, these results for the adaptive control show, that giving cargo items priority and delivering them in a sequential style, while delivering passengers in a collective style, can increase the amount of cargo that gets delivered. This reduces the average ride time. However, it comes with a trade-off in waiting time. The increased performance can be explained by the reduced time, the cargo items block the space and weight restrictions present in the elevator cabin and prevent other passengers from entering. Therefore delivering them faster causes a shorter blocking of these resources, resulting in better overall utilization.

The precondition of applying the adaptive control strategy is the knowledge, whether a cargo object is currently inside the elevator cabin and is blocking it for other passengers and the information about the destination of this object. This information can be

obtained by combining the visual system presented before, which is capable of detecting cargo objects by their volume, with the information about car calls made for this object. To create the association of an object present in the lift and its destination, the time of entering the cabin and performing the car call can be correlated. Otherwise, in an environment that features hall calls with information about the destination, this association can be used. An alternative would be an *override* button in the cabin, to give priority to the current car call, regardless of the presence of a cargo object in the cabin.

However, the scenario chosen for the simulation is very specific and can benefit from the kinds of optimizations the adaptive control yields. The given traffic conditions in a multi-storey building with a single elevator, which is used by passengers and cargo alike, do favor the prioritization of cargo items by sequentially delivering them. The results for other traffic patterns, e.g. for elevators that only convey passengers, might differ and might not benefit from adapted scheduling.

4.3. Economic Considerations

As stated above, the presented scheduling algorithm can use the visual system to determine the presence of a cargo object, which can benefit the handling capacity of elevators which are regularly used by passengers and cargo items. There are several possible real-life scenarios with these properties:

- **Hospitals** with elevators where patient beds are moved. The medical beds are usually as large as the elevator allows and therefore block the elevator from being used by groups of additional passengers. Additionally, patients with medical conditions who occupy the bed often should have a prioritized transport, as their health might depend on every second spent in the elevator.
- Large office buildings with **auxiliary lifts** that are used by janitors and craftsman, who carry carts, tool cabinets or hardware material. These lifts could also be used by regular passengers and might even be integrated into an elevator group. However, the cart could block the elevator.

- Multi-storey **Warehouses** or **fabrication plants** with heavy duty cargo lifts used by factory employees and forklifts or other transportation devices for heavy goods. The lifts block each other from entering the lift and therefore need to be handled sequentially.

As already stated in the introduction, possible customers for DXC Technology and the Digital Service Innovation team are Otis, Kone, Schindler and Thyssen, which Unger [Ung15, p. 4] identifies as the biggest elevator companies. Some of those companies already have business relations with DXC Technology and therefore might be interested in setting up an innovative project to explore the ideas presented in this paper. They could consider offering an elevator system with an integrated visual system to customers with the needs described above, who construct a new building or want to upgrade an existing lift, as an optional feature at an additional charge.

The cost structure for the visual solution is threefold. A project implementing this or a similar solution generates cost during development, installation of the individual solution and later operation. Since elevator technology experts and computer vision experts need to work together for the development of an advanced visual solution in elevators, the costs in this phase are characterized by expenses for said experts, as well as laboratory equipment. Developing resilient and high-quality software requires careful planning and implementation and is therefore crucial for the costs at this stage. When installing the solution into an elevator cabin and integrating it into the control circuit, the costs are governed by the hardware installed and the labor of the technician installing it. The costs for the hardware includes multiple cameras, do neither need an extraordinary high resolution nor frame rate, and a general purpose computer to process the images, which needs to have enough computing power to process the images in near real-time. Furthermore, a connection to the elevator controller is necessary which might require some kind of adapter. During operation, the costs are reduced down to the electricity consumed by the additional components and the costs for maintaining the system.

Possible savings from implementing the proposed visual system and the scheduling algorithm manifest in the faster delivery of cargo items and therefore the reduction of the time they block the cabin for other passengers. This can result in a higher overall passenger throughput and therefore reduce the energy needs per passenger. Further-

more, the faster delivery of cargo objects can have impacts on the costs associated with their travel time, which highly depend on the type of good that is transported and the environment it is situated in. For example in a production plant, reduced transport times for the produced goods could lead to reduced production costs in general.

4.4. Privacy Considerations

Installing a camera system unavoidably raises privacy and security concerns for the filmed individuals. Especially in elevators, which are publicly accessible, there are legal requirements to fulfill. In Germany the Federal Data Protection Act, Bundesdatenschutzgesetz (BDSG) [Fed09], regulates the usage of video surveillance systems. Public areas may only be observed for specified purposes. The presence of a surveillance system must be indicated, as well as the name and contact of its controller. When the data gathered by the system is no longer needed for the specified purpose, it must be deleted [Fed09, § 4]. The European General Data Protection Regulation (GDPR) [Cou16] describes similar, more broad principles: *data transparency, purpose limitation, data minimization, storage limitation and confidentiality* [Inf18]. Oriented at this principle the following paragraphs provide privacy consideration about the visual system.

The presented visual system has a *limited purpose*, in that it is only used to determine whether passengers or cargo items are present in the elevator cabin and what volume they take up. This information is used to improve the scheduling of the elevator.

In order to keep the usage of video data *transparent*, passengers should be informed about the video system before they enter the cabin. This could be done by installing a sign, which indicates the presence of the video system, clearly states, for what purpose the video data is used for, and who is operating the system and can be contacted in case of issues. The sign should be in front and inside of the elevator, so that potential passengers can decide, if they want to use the lift and be filmed.

Since the system has a limited purpose, only data specific to that purpose should be gathered and processed. The data usage should be *minimized*. Only as few cameras as possible, but as many as necessary should be installed in the lift. Areas that are not

of interest for the lift system should not be filmed. Furthermore, the resolution of the cameras should be only as high as necessary. For purpose of detecting the presence, position, and outline of a passenger or cargo object in the cabin, it is not required to capture fine details. The DIN EN 62676-4 norm [DIN16] recommends to use a camera resolution, which captures 25 to 62.5 pixel per meter, for the purpose of detecting or monitoring people. In addition, the kind of processing of the video data should be limited to the extent necessary for the purpose of the video system. For example, it is not necessary to perform face detection or to identify individuals. However, tracking the path of an individual over the course of their journey might be beneficial for the application.

The presented system can operate in real time. Only the current video frame is needed to determine the presence and volume of passengers and objects. Therefore, the *storage* of the video data can be *limited*. There is no need to store the video data at all.

In order to keep the video data *confidential*, measures must be taken to prevent access by unauthorized parties. For the purpose of the system, all data processing can be done locally within the elevator system. The visual system only needs to interact with the elevator controller. Therefore there is no need to make the data available to the outside, e.g. the internet. Doing so would open up unnecessary possibilities for misuse of the video system.

5. Conclusions

This chapter summarizes the outcomes of the thesis at hand and reflects on the conducted methods. Key ideas of the previous chapters are pointed out, and a discussion regarding the solution quality and alternative solutions is held. Lastly, a suggestion for the next steps at DXC Technology regarding the utilization of this thesis is given.

5.1. Summary

This thesis explored the possibilities to support the movement control of elevator systems with the help of a camera system to estimate the volume occupied in the cabins. First, the theoretical background in 2D and 3D computer vision have been explained. After that, two major artifacts have been designed, created and evaluated by following the principles of design science research. Primarily a design for a volume recognition system inside elevator cabins has been conducted. The system uses techniques which include foreground-masking by background subtraction with the MOG method and most importantly the technique of voxel-based volume intersection. The system was partially implemented and tested on a real elevator cabin to show the general feasibility of the chosen approach.

As a secondary artifact a new elevator control strategy has been presented. It extends the common collective control strategy by giving priority to delivering cargo objects in order to free up the cabin quickly. A simulation which models an elevator system with discrete events has been used to evaluate the performance of this control strategy. It has been shows that in a single-elevator set-up with constant arrival rates of passengers and cargo objects the new strategy delivers 35.52 % more cargo objects than collective control. It has a 18.40 % lower average ride time which comes at the cost of a 30.56 %

higher waiting time. Possible use cases include buildings where the elevator commonly conveys passengers and cargo objects. Examples are hospitals, auxiliary lifts and fabrication plants.

5.2. Discussion and Critical Reflection

In the introduction of this thesis (section 1.2) several research questions have been stated, which over the over the course of this document have been implicitly answered. It has been explained, that the computer vision techniques of background subtraction and volume intersection are one possibility to detect passengers and other objects inside an elevator cabin. The movement of that elevator can then optimized by altering its control strategy based on whether cargo objects are present and prioritize their delivery. Other interesting objects that might be detected have been deduced from specific use cases and include janitor carts, forklifts or hospital beds. It is useful to integrate the proposed visual system into an elevator in buildings where passengers and cargo objects share a single lift. The extent of the usefulness of the integration has been shown by running a simulation. It turns out that a desirable optimization goal is the fast delivery of cargo objects.

In the consideration for both artifacts, only one particular solution each has been explored in greater detail. However, many more use cases and approaches would have been interesting. A broader range of consideration and a thorough decision process could enhance the scientific demands of this thesis. Alternative solutions for the visual system could be found in depth sensor technology, feature-based depth reconstruction with stereo vision. However, these solutions would come at a higher expense or complexity. In the initial problem exploration, the installation of cameras in front of the cabin on each floor has also been proposed, but the idea was not pursued further. Alternative solutions for the conducted elevator simulation could be found by investing in industrial software instead of modeling the elevator system by manually.

The conducted simulation was kept simple with specific building parameters and models the elevator system by using discrete events, which yields a presumably correct implementation. Moreover, the simulation only models one particular situation and

therefore many premises were made for the parameters of the system. Some parameters were not even considered in the implementation, even though they were specified. It would also have been possible to model the physical behavior in more detail of the system and take more parameters into account. Also, only a simple traffic pattern and basic control strategies have been implemented. A more detailed modeling in this respects could lead to more informed results. Taking more metrics out of the simulation can also create a deeper understanding of the properties of the simulation, like the direct measurement of capacity utilization or the total journey time. The results generated by the simulation do not contradict the general intuition and lie in an expected range. However, the results give only a vague indication of how the proposed scheduling algorithm would perform in reality.

The chosen methodology of design science research proved capable of conducting the research in this thesis. It provides a broad guideline to the project execution and ensures the produced artifacts are not only designed and implemented but also evaluated and presented. In particular, also the re-circulation of the gained insight into the body of knowledge of academia is demanded. However, the results of this thesis are not likely to be published in scientific conferences or journals.

5.3. Further Work

The presented visual solution is far from production-ready and only showed the general feasibility of the approach. If the project of this thesis is continued, a complete implementation of the visual system is desirable, which uses a correct perspective projection and realizes the proposed blob detection and classification. However, based on the shown results, DXC Technology can try to get possible customers from the elevator business interested in developing innovative ideas together with them. They can explore many more use cases of camera systems in elevators and team up with computer vision professionals or start-ups to put a solution into place, which meets the customers' needs.

Bibliography

[AB13] Johanna Axelsson and Sarah Bernelind. *Elevator Control Strategies*. 2013. URL: https://www.diva-portal.org/smash/get/diva2:668654/FULLTEXT01.pdf (visited on 07/04/2018).

[BA16] Gina Barney and Lutfi Al-Sharif. *Elevator Traffic Handbook, Theory and Practice*. 2nd ed. Routledge, 2016.

[Bee15] Leonie Beers. *Reducing passenger waiting time in elevator traffic by adding information about passenger arrivals*. 2015. URL: https://beta.vu.nl/nl/Images/werkstuk-beers_tcm235-431096.pdf (visited on 06/13/2018).

[BL01] Andrea Bottino and Aldo Laurentini. „A Silhouette Based Technique for the Reconstruction of Human Movement". In: *Computer Vision and Image Understanding* 83.1 (2001), pp. 79–95. ISSN: 1077-3142. DOI: https://doi.org/10.1006/cviu.2001.0918. URL: http://www.sciencedirect.com/science/article/pii/S107731420190918X.

[BN04] Matthew Brand and Daniel Nikovski. „Optimal parking in group elevator control". In: *ICRA'04. 2004 IEEE International Conference on Robotics and Automation, 2004. Proceedings.* Vol. 1. IEEE. 2004, pp. 1002–1008.

[Can86] John Canny. „A computational approach to edge detection". In: *IEEE Transactions on pattern analysis and machine intelligence* 6 (1986), pp. 679–698.

[CMV11] Antoni B. Chan, Vijay Mahadevan, and Nuno Vasconcelos. „Generalized Stauffer-Grimson background subtraction for dynamic scenes". In: *Machine Vision and Applications* 22.5 (2011), pp. 751–766. ISSN: 1432-1769. DOI: 10.1007/s00138-010-0262-3. URL: https://doi.org/10.1007/s00138-010-0262-3.

[Cou16] Council of the European Union. *General Data Protection Regulation*. 2016. URL: http://data.consilium.europa.eu/doc/document/ST-5419-2016-INIT/en/pdf (visited on 08/16/2018).

[DIN16] DIN. *DIN EN 62676-4:2016-07, Video surveillance systems for use in security applications - Part 4: Application guidelines*. 2016.

[DLA15] Aline Dresch, Daniel Pacheco Lacerda, and José Antônio Valle Antunes Jr. *Design Science Research*. Springer, 2015. ISBN: 978-3-319-07373-6.

[Fed09] Federal Ministry of Justice and Consumer Protection. *Federal Data Protection Act*. 2009. URL: https://www.gesetze-im-internet.de/englisch_bdsg/englisch_bdsg.html (visited on 08/16/2018).

[Hak03] Henri Hakonen. *Simulation of Building Traffic and Evacuation by Elevators*. 2003. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.14.9003&rep=rep1&type=pdf.

[He+17] Lifeng He et al. „The connected-component labeling problem: A review of state-of-the-art algorithms“. In: *Pattern Recognition* 70 (2017), pp. 25–43. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2017.04.018. URL: http://www.sciencedirect.com/science/article/pii/S0031320317301693.

[Inf18] Information Commissioner's Office. *Guide to the General Data Protection Regulation (GDPR) / The Priciples*. 2018. URL: https://ico.org.uk/for-organisations/guide-to-the-general-data-protection-regulation-gdpr/principles/ (visited on 08/16/2018).

[Int18] Intel Corporation. *Intel RealSense Technology*. 2018. URL: https://www.intel.com/content/www/us/en/architecture-and-technology/realsense-overview.html (visited on 08/30/2018).

[KLB14] Ohhoon Kwon, Eunji Lee, and Hyokyung Bahn. „Sensor-aware elevator scheduling for smart building environments“. In: *Building and Environment* 72 (2014), pp. 332–342.

[KON16] KONE Corporation. *KONE Elevator Toolbox*. 2016. URL: https://toolbox.kone.com/ (visited on 08/24/2018).

[KON17] KONE Deutschland. *Alle KONE Aufzüge im Überblick*. 2017. URL: https://www.kone.de/aufzug-aufzuege.aspx (visited on 07/25/2018).

[KS99] K. N. Kutulakos and S. M. Seitz. „A theory of shape by space carving“. In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Vol. 1. 1999, pp. 307–314. DOI: 10.1109/ICCV.1999.791235.

[Lau94] A. Laurentini. „The visual hull concept for silhouette-based image understanding“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 16.2 (1994), pp. 150–162. ISSN: 0162-8828. DOI: 10.1109/34.273735.

[LHS90] Leon Levine, Victor Huang, and Israel Saguy. „Use of Computer Vision for Real Time Estimation of Volume Increase During Microwave Baking“. In: *Cereal Chemistry* 67.1 (1990), pp. 104–105. URL: https://www.aaccnet.org/publications/cc/backissues/1990/Documents/67_104.pdf.

[Lin+11] Lin Lin et al. *Video Aided System For Elevator Control*. US Patent 8,020,672 B2. 2011. URL: http://www.google.com/patents/US8020672B2.

[Mal15] Satya Mallick. *Blob Detection Using OpenCV*. 2015. URL: https://www.learnopencv.com/blob-detection-using-opencv-python-c/ (visited on 08/30/2018).

[Mic18] Microsoft. *Kinect for Windows*. 2018. URL: https://developer.microsoft.com/en-us/windows/kinect (visited on 08/30/2018).

[MK14] Nicholas D. Matsakis and Felix S. Klock II. „The Rust Language“. In: *Proceedings of the 2014 ACM SIGAda Annual Conference on High Integrity Language Technology*. HILT '14. Portland, Oregon, USA: ACM, 2014, pp. 103–104. ISBN: 978-1-4503-3217-0. DOI: 10.1145/2663171.2663188. URL: http://doi.acm.org/10.1145/2663171.2663188.

[Ope18a] OpenCV. *cv::SimpleBlobDetector Class Reference*. 2018. URL: https://docs.opencv.org/3.4/d0/d7a/classcv_1_1SimpleBlobDetector.html (visited on 08/07/2018).

[Ope18b] OpenCV Dev Team. *Camera Calibration and 3D Reconstruction*. 2018. URL: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html (visited on 08/03/2018).

[Ope18c] OpenCV Dev Team. *Making your own linear filters*. 2018. URL: https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/filter_2d/filter_2d.html (visited on 08/03/2018).

[Ope18d] OpenCV Team. *OpenCV Library*. 2018. URL: https://opencv.org/ (visited on 08/03/2018).

[Pet18] Peters Research Ltd. *Elevate*. 2018. URL: https://www.peters-research.com/index.php/elevate (visited on 08/23/2018).

[Pic04] M. Piccardi. „Background subtraction techniques: a review“. In: *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583)*. Vol. 4. 2004, pp. 3099–3104. DOI: 10.1109/ICSMC.2004.1400815.

[Pyt18a] Python Software Foundation. *PyPI - the Python Package Index*. 2018. URL: https://pypi.org/ (visited on 08/27/2018).

[Pyt18b] Python Software Foundation. *Python 3.6.6*. 2018. URL: https://www.python.org/downloads/release/python-366/ (visited on 08/27/2018).

[Rus18a] Rust Team. *Cargo: Packages for Rust*. 2018. URL: https://crates.io/ (visited on 08/27/2018).

[Rus18b] Rust Team. *The Rust Programming Language*. 2018. URL: https://www.rust-lang.org/en-US/ (visited on 08/10/2018).

[Sap+17] D.D. Sapkal et al. „Volume Estimation of An Object Using 2D Images“. In: *International Journal of Pure and Applied Mathematics* 114.12 (2017), pp. 333–341.

[SG99] C. Stauffer and W. E. L. Grimson. „Adaptive background mixture models for real-time tracking“. In: *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*. Vol. 2. 1999, p. 252. DOI: 10.1109/CVPR.1999.784637.

[Sii93] Marja-Liisa Siikonen. „Elevator Traffic Simulation“. In: *Transactions of The Society for Modeling and Simulation International - SIMULATION*. Vol. 61. 1993, pp. 257–267.

[Sii97] Marja-Liisa Siikonen. *Planning and Control Models for Elevators in High-Rise Buildings*. 1997. URL: http://sal.aalto.fi/publications/pdf-files/rsii97b.pdf (visited on 06/14/2018).

[Šmí15] Matěj Šmíd. *Camera Calibration Using OpenCV*. 2015. URL: https://github.com/smidm/video2calibration (visited on 08/27/2018).

[Son11] Sonaten. *Object Volume Calculation/Estimation*. 2011. URL: https://stackoverflow.com/questions/8337920/object-volume-calculation-estimation (visited on 06/19/2018).

[SS85] Suzuki and Satoshi. „Topological structural analysis of digitized binary images by border following“. In: *Computer vision, graphics, and image processing* 30.1 (1985), pp. 32–46.

[Sze10] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[Ung15] Dieter Unger. *Aufzüge und Fahrtreppen*. Springer Vieweg, 2015. ISBN: 978-3-662-56241-3.

[Wan+16] Xiangyu Wang et al. *Traffic List Generation For Passenger Conveyance*. European Patent 3 075 693 A1. 2016. URL: http://www.google.com/patents/EP3075693A1.

[Wan+17] Dandan Wang et al. „Computer vision for bulk volume estimation of apple slices during drying". In: *Drying Technology* 35.5 (2017), pp. 616–624. DOI: 10.1080/07373937.2016.1196700. URL: https://doi.org/10.1080/07373937.2016.1196700.

[WCH92] J. Weng, P. Cohen, and M. Herniou. „Camera calibration with distortion models and accuracy evaluation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.10 (1992), pp. 965–980. ISSN: 0162-8828. DOI: 10.1109/34.159901.

[XW13] Xocoatzin and Kai Wolf. *Voxel-Carving*. 2013. URL: https://github.com/xocoatzin/Voxel-Carving (visited on 08/10/2018).

[Zha17a] Yujin Zhang. *Image Analysis*. De Gruyter, 2017. ISBN: 978-3-110-52428-4.

[Zha17b] Yujin Zhang. *Image Processing*. De Gruyter, 2017. ISBN: 978-3-110-52422-2.

[Zha17c] Yujin Zhang. *Image Understanding*. De Gruyter, 2017. ISBN: 978-3-110-52423-9.

[Zoo09] ZooFari. *File:Epipolar Geometry1.svg*. 2009. URL: https://commons.wikimedia.org/wiki/File:Epipolar_Geometry1.svg (visited on 08/30/2018).

A. Simulation Source Code (excerpts)

```
fn main() {
    let b = BuildingParameters::default();
    let l = LiftParameters::default();
    let s = SimulationParameters::default();
    println!("{:#?}\n{:#?}\n{:#?}\ntotal_ticks: {}\n", &b, &l, &s, s.total_ticks());

    let results =
        rayon::iter::repeatn((), s.simulations) // enable parallel execution
        .enumerate()
        .inspect(|(num, _)| println!("running simulation {}", num))
        .map(|_| TrafficGenerator::new(&s, &b).arrivals())
        .map(|t| (
            run_single_simulation::<SequentialControlStrategy>(&b, &l, &s, &t),
            run_single_simulation::<CollectiveControlStrategy>(&b, &l, &s, &t),
            run_single_simulation::<AdaptedControlStrategy>(&b, &l, &s, &t),
            ))
        .collect::<Vec<_>>();
    let invalid = results.iter()
        .filter(|&(a, b, c)| a.is_err() || b.is_err() || c.is_err())
        .count();
    let valid: Vec<_> = results.into_iter()
        .filter(|&(ref a, ref b, ref c)| a.is_ok() && b.is_ok() && c.is_ok())
        .map(|(a, b, c)| (a.unwrap(), b.unwrap(), c.unwrap()))
        .collect();
    let valid_a = valid.iter().map(|&(ref a, ref _b, ref _c)| a);
    let valid_b = valid.iter().map(|&(ref _a, ref b, ref _c)| b);
    let valid_c = valid.iter().map(|&(ref _a, ref _b, ref c)| c);

    println!("invalid results: {}", invalid);
    println!("{}", SequentialControlStrategy::name());
    print_stats(valid_a);
    println!("{}", CollectiveControlStrategy::name());
    print_stats(valid_b);
    println!("{}", AdaptedControlStrategy::name());
```

```
35      print_stats(valid_c);
36
37      println!("\ndone.");
38  }
```

Listing A.1: Main function of the simulation program

```
1   fn run_single_simulation<S: ControlStrategy>(
2       building: &BuildingParameters,
3       lift: &LiftParameters,
4       sim: &SimulationParameters,
5       traffic: &Vec<Vec<TrafficItem>>,
6   ) -> Result<SimulationResults, String> {
7
8       println!("running simulation: {:?}", S::name());
9       let mut system = ElevatorSystem::new(building, lift, sim);
10      let mut state = S::State::default();
11      let mut results = SimulationResults::default();
12
13      results.passengers_in_traffic = traffic.iter()
14          .flat_map(|t| t)
15          .filter(|&t| t.kind == TrafficItemKind::Passenger)
16          .count();
17      results.cargo_in_traffic = traffic.iter()
18          .flat_map(|t| t)
19          .filter(|&t| t.kind == TrafficItemKind::Cargo)
20          .count();
21
22      while system.ticks < sim.total_ticks() {
23          let last_ticks = system.ticks;
24          let (next_state, action) = S::action(state, &system);
25          state = next_state;
26          match action {
27              ControlAction::MoveUp => {
28                  let time = system.move_up()?;
29                  system.ticks += sim.time_to_ticks(time);
30              },
31              ControlAction::MoveDown => {
32                  let time = system.move_down()?;
33                  system.ticks += sim.time_to_ticks(time);
34              },
35              ControlAction::OpenDoors => {
36                  let time = system.open_doors()?;
37                  results.stops += 1;
38                  system.ticks += sim.time_to_ticks(time);
39              },
40              ControlAction::CloseDoors => {
41                  let time = system.close_doors()?;
42                  system.ticks += sim.time_to_ticks(time);
43              },
44              ControlAction::LoadTrafficItems(amount) => {
45                  let mut to_load = Vec::with_capacity(amount);
46                  for _ in 0..amount {
47                      if let Some(i) = system.floor_queues[system
                            .lift_floor]
48                          .pop_front()
```

```
49                      {
50                          to_load.push(i);
51                      }
52                  }
53                  let (time, rest) = system.load_traffic_items(
                        to_load)?;
54                  for i in rest {
55                      system.floor_queues[system.lift_floor]
56                          .push_front(i);
57                  }
58                  system.ticks += sim.time_to_ticks(time);
59              },
60              ControlAction::UnloadTrafficItems => {
61                  let current_floor = system.lift_floor;
62                  let (time, items) = system.
                        unload_traffic_items_for_floor(current_floor
                        )?;
63                  results.passengers_delivered += items.iter()
64                      .filter(|&i| i.kind == TrafficItemKind::
                            Passenger)
65                      .count();
66                  results.cargo_delivered += items.iter()
67                      .filter(|&i| i.kind == TrafficItemKind::
                            Cargo)
68                      .count();
69                  system.ticks += sim.time_to_ticks(time);
70              },
71              ControlAction::Nothing => {
72                  system.ticks += 1;
73              }
74          }
75
76          let next_tick = cmp::min(system.ticks, sim.total_ticks
                ());
77          let tick_diff = next_tick - last_ticks;
78
79          // update waiting times
80          for id in system.floor_queues.iter()
81              .flat_map(|v| v.iter())
82              .map(|i| i.id)
83          {
84              *results.waiting_ticks.entry(id).or_insert(0) +=
                    tick_diff;
85          }
86
87          // update ride times
88          for id in system.lift_traffic_items.values()
89              .flat_map(|v| v.iter())
90              .map(|i| i.id)
91          {
92              *results.ride_ticks.entry(id).or_insert(0) +=
                    tick_diff;
93          }
94
95          // advance ticks and update floor queues with traffic
```

```
            items
96          let new_traffic_items = traffic[last_ticks..next_tick]
97              .iter()
98              .flat_map(|v| v)
99              .cloned();
100         for i in new_traffic_items {
101             system.floor_queues[i.from_floor].push_back(i)
102         }
103
104     }
105
106     // remove ride times of passenegers that did not reach
        destination yet
107     // and are in the lift
108     for id in system.lift_traffic_items.values()
109         .flat_map(|v| v.iter())
110         .map(|i| i.id)
111     {
112         results.ride_ticks.remove(&id);
113     }
114
115     // remove waiting times of passenegrs still waiting
116     for id in system.floor_queues.iter()
117         .flat_map(|v| v.iter())
118         .map(|i| i.id)
119     {
120         results.waiting_ticks.remove(&id);
121     }
122
123     Ok(results)
124 }
```

Listing A.2: Implementation of a single simulation run

B. Volume Intersection Source Code (excerpts)

```
def main():
    print_versions()

    cv2.namedWindow('sliders', flags=cv2.WINDOW_NORMAL)
    cv2.createTrackbar('open_kernel','sliders',3,101,nop)
    cv2.createTrackbar('dilate_kernel','sliders',17,101,nop)

    view_3d = init_3d_plot()
    create_3d_room(view_3d)
    create_3d_cams(view_3d)
    vol_3d = create_3d_volume(view_3d)
    side_3d = create_3d_side(view_3d)

    caps = [ cv2.VideoCapture(c['file']) for c in CAPTURES ]
    bgss_pp = [ cv2.createBackgroundSubtractorMOG2() for _ in
       caps ]

    video_out_res = (1920, 480)
    if OUTPUT:
        fourcc = cv2.VideoWriter_fourcc(*'XVID')
        video_out = cv2.VideoWriter('output.avi',fourcc, 25.0,
           video_out_res)

    while True:
        # read all capture images at this moment
        raws = [ c.read()[1] for c in caps]

        # convert to grayscale
        gry = [ cv2.cvtColor(c, cv2.COLOR_BGR2GRAY) for c in
           raws ]
        gry_preview = multi_preview(gry)

        # get parameters from trackbars
        k_size = cv2.getTrackbarPos('open_kernel','sliders')
        k_size = k_size - k_size % 2 +1
        d_size = cv2.getTrackbarPos('dilate_kernel','sliders')
        d_size = d_size - d_size % 2 +1

        # pre process images and appy background subtractor
        pp = [ cv2.GaussianBlur(r, (k_size,k_size), 0) for r in
            raws ]
```

```
38          bgs_pp_masks = [ bgss_pp[i].apply(pp[i]) for i in range
               (len(raws)) ]
39
40          # post process foreground masks
41          open_kernel = np.ones((k_size,k_size),np.uint8)
42          dilate_kernel = np.ones((d_size,d_size),np.uint8)
43          bgs_pp_masks_t = [ cv2.threshold(b, 254, 255, cv2.
               THRESH_BINARY)[1] for b in bgs_pp_masks]
44          bgs_pp_masks_b = [ cv2.dilate(cv2.morphologyEx(b, cv2.
               MORPH_OPEN, open_kernel), dilate_kernel, iterations
               = 10)\
45              for b in bgs_pp_masks_t ]
46          bgs_pp_preview = multi_preview(bgs_pp_masks_b)
47
48          # draw masks A B C to the side, orthogonal (semi
               correct)
49          scaled_masks = draw_3d_side(side_3d, bgs_pp_masks_b)
50
51          # create orthogonal  intersection of A B C
52          vol_poss = np.array([ \
53              [ \
54                  u * ROOM_SCALE_MPVX['width'],\
55                  v * ROOM_SCALE_MPVX['depth'],\
56                  w * ROOM_SCALE_MPVX['height']\
57              ] \
58              for u in range(ROOM_DIM_VX['width']) \
59              for v in range(ROOM_DIM_VX['depth']) \
60              for w in range(ROOM_DIM_VX['height']) \
61              if  scaled_masks[0][w][v] == 255 \
62              and scaled_masks[1][w][u] == 255 \
63              and scaled_masks[2][w][v] == 255 ])
64          vol_3d.setData(pos=vol_poss)
65
66          # create combined preview
67          image_3d = convertQImageToMat(view_3d.readQImage())
68          scale = 480 / image_3d.shape[0]
69          image_3d_preview = cv2.resize(image_3d, (0,0), fx=scale
               , fy=scale)
70          image_3d_preview = cv2.cvtColor(image_3d_preview, cv2.
               COLOR_BGRA2BGR)
71          full_preview = np.hstack((\
72              cv2.cvtColor(np.vstack((gry_preview, bgs_pp_preview
                   )), cv2.COLOR_GRAY2BGR), \
73              image_3d_preview \
74              ))
75          cv2.imshow('full_preview', full_preview)
76          out = np.zeros((video_out_res[1], video_out_res[0], 3),
               dtype=np.uint8)
77          out[:, 0:full_preview.shape[1], :] = full_preview
78          if OUTPUT:
79              video_out.write(out)
80
81          if cv2.waitKey(int(1000/25)) != -1:
82              break
83
```

```
84      cv2.destroyAllWindows()
85      for c in caps:
86          c.release()
87      if OUTPUT:
88          video_out.release()
```

Listing B.1: Main function of the volume intersection program

C. CD or DVD with Source Code