

Camera Positioning and Vision-based Fall Detection

Jerom van der Sar, Arjan Seijs,
Ivo Nelissen, and Robin Cromjongh

Bachelor End Project
For the Bachelor Computer Science and Engineering



Camera Positioning and Vision-based Fall Detection

by

Jerom van der Sar, Arjan Seijs,
Ivo Nelissen, and Robin Cromjongh

Project duration: April 23, 2019 – July 2, 2019
Thesis committee: Ir. O. Visser, TU Delft, Course Coordinator
Dr. C. C. S. Liem, TU Delft, Coach
K. Rassels MSc, Eya Solutions & TU Delft, Client

An electronic version of this thesis is available at <http://repository.tudelft.nl/>.

Executive Summary

Elderly care residences often have falling incidents, sometimes with dire consequences. This project aims to implement a method for detecting when people fall, an issue that has seen much research and practical implementations already. In contrast to other work, this project aims to create a non-intrusive, non-wearable solution through the use of cameras. It extends an existing patient monitoring system developed by Eya Solutions, a start-up in the medical field. Their system monitors patients and is composed of embedded systems (*clients*), a back-end with a database, and a web-based dashboard (the *front-end*) for administrators, caretakers and users. Our client wants to extend their system by introducing: (i) a video processing component which performs face recognition, (ii) functionality for indoor positioning of people, (iii) functionality to automatically determine near-optimal placement of cameras, and (iv) automatic detection of fall incidents.

To understand the relevant research, we survey the field investigating five key research topics: (i) object detection, (ii) fall detection, (iii) facial recognition, (iv) floor plan modelling, and (v) camera placement optimisation. Research shows that in object detection and fall detection, first using background subtraction and then fitting ellipses around objects is a common method for tracking pedestrians. Because this method is also intuitive, we employ this method in our approach. Facial recognition proved to be very challenging in our setting, and the relevant research confirms this. Because of this reason and time constraints, our project does not implement this functionality. Automatic floor plan modelling is possible, but difficult, and was thus also decided to keep as future work. Research into camera placement optimisation revealed that using a discretised representation is best suited for our purpose and that a genetic algorithm is a common and effective method of optimising. Our project implements both.

The first part of the project is detecting fall incidents in video footage. By means of image processing (in particular, subsequent frame subtraction), our system detects objects in the foreground and tracks these between frames. Based on the properties of such objects, namely size and shape, we detect fall incidents.

The second part of the project is the floor plan editor. This editor is integrated into the existing dashboard, where administrators can model the building. Eya Solutions wishes to provide the product to customers as a complete package, including (i) showing where fall incidents have occurred, and (ii) how and where to place cameras. Our system allows administrators to edit the floor while allowing caretakers to see the modelled floor and see where falls are detected. Considering (ii), it is desirable to generate a configuration where the number of cameras is low while the view coverage is high. Our contribution includes a genetic algorithm which can generate automatically a suitable configuration. Alternatively, cameras can be placed manually by the administrator.

This system can be considered privacy sensitive because elderly are filmed throughout the day and in the deployment building. This is, however, acceptable, considering caretakers have to be able to see the video feeds in order to provide proper care, and because only caretakers can see the feed.

All main aspects of the system have been tested. The fall detection was tested on accuracy, showing an accuracy of 33% and a false positive rate of 0.5 false notifications per minute. The floor plan editor is tested by means of user tests and unit tests. Our optimisation algorithm was tested on speed and bottlenecks.

In order to complete this project, we employed a loose variant of the SCRUM development methodology, with sprint lengths of one week. The team was split among the two main projects, but communication was still fluent within the entire team. Challenges in the process of this project were the high amount of requested functionality and a lack of available data. To resolve this, we cut in what functionality we were going to implement and worked with the limited data sets that was publicly available. Splitting the team in two was a limitation to the overall achievements, but a reasonable solution to the diverse challenges from the client.

Our system functions as expected and serves as a minimal viable product for the given functionality. As such, it is a good starting point for further development to more complete software. For the fall detection algorithms, finding or creating more representative data could be used to train the fall detection so that the detection accuracy can be improved. The floor plan editor could use improved user friendliness. The system could also be extended to use facial recognition on tracked objects.

Preface

This project was the final stepping stone to achieving our Bachelor of Science degrees. For us, this project was challenging for a multitude of reasons, not firstly because the stakes were high: we worked with a client outside the domain of computer science, we continued work on a pre-existing code base, which was both large in scale and complex in design, and we were tasked to implement an open-ended product with limited assistance. Essentially, we were allowed to tackle the entire process of investigating, research, design, implementing, and testing on our own. Now that our Bachelor is coming to a close, we must note that we did not succeed without help, and a thank-you is most certainly in place.

Firstly, we would like to thank our client, Kianoush Rassels, for the opportunities he has granted us in working on this project. You have made this project a true learning experience.

Secondly, we would like to thank our supervisor, Cynthia Liem, her guidance and support. As an expert on computer vision, her insight into pedestrian detection algorithms have truly been useful. Thank you for your guidance! We have enjoyed our journey through this project.

Thirdly, we would like to thank all professors, teachers, teaching assistants, coordinators, and supporting staff who have taught us much in the past years. Now we are much more knowledgeable on Computer Science and you were certainly essential.

Lastly, we would like to thank a multitude of other individuals who have helped in the process. Thank you, family and friends, of which there are too many to name. We have had a blast going through this process with you! Thank you, participants of our user tests, for your time and feedback which we have incorporated into this report. Finally, thank you, public transport, for being on time every now and then.

*Jerom van der Sar, Arjan Seijs,
Ivo Nelissen, and Robin Cromjongh
Delft, June 2019*

Contents

List of Figures	ix
List of Tables	ix
List of Algorithms	ix
1 Introduction	1
2 Problem Definition	3
2.1 System Overview	3
2.2 Project Goal	3
3 Related Work	5
3.1 Object Detection	5
3.2 Fall detection	6
3.3 Facial Recognition	8
3.4 Floor Plan Modelling	8
3.5 Camera Placement Optimisation	9
3.6 Research Conclusion	11
4 System Design	13
4.1 Object Detection, Tracking, and Fall Detection	13
4.2 Floor Plan Management	13
5 System Implementation	15
5.1 Implementation Overview	15
5.2 Object Detection	15
5.3 Object Tracking and Fall Detection	17
5.4 Floor Plan Management	17
5.5 Camera Coverage Optimisation	19
5.6 Feedback from Software Improvement Group	21
6 Ethical Implications	23
7 Evaluation	25
7.1 Fall Detection	25
7.2 Dashboard User Test	26
7.3 Floor Coverage Optimisation	27
8 Process	31
8.1 Overview	31
8.2 Challenges	31
8.3 Reflection	32
9 Conclusion	33
10 Future Work	35
A Project Description	41
B Requirements Analysis	43
B.1 Functional Requirements	43
B.2 Non-functional Requirements	44

C	Feedback from SIG	45
C.1	Feedback week six	45
C.2	Feedback week nine.	46
D	Info sheet	47
D.1	Description	47
D.2	Contributors	48
D.3	Contact	48
E	User Manual	49

List of Figures

2.1	Conceptual overview of the system provided by the project client.	4
4.1	Conceptual design of our system.	14
4.2	State diagram indicating the states of our tracker object.	14
5.1	The class diagram for the video processing module	16
5.2	The UML class diagram of the floor plan module.	18
5.3	Four stages in grid generation: from a grid to a finer grid.	20
7.1	The user cheat sheet. Each icon represents a tool in the tool bar.	26
7.2	The floor plan used for view coverage optimisation experiments.	28
7.3	Two stages of performance optimisation: from 27 cameras to 16.	29
8.1	The project road map.	32

List of Tables

3.1	Falling activities identified by Noury et al. [38].	7
3.2	Results of relevant platforms presented by other researchers.	9
5.1	Overview of hyperparameters of our genetic algorithm.	21
7.1	Performance results for view coverage optimisation of our genetic algorithm	29
7.2	Performance profiling data of our genetic algorithm	30

List of Algorithms

5.1	Computes the fitness of a given individual.	20
5.2	Approximates the view coverage of a given individual with < 1% accuracy.	20
5.3	Calculates the points for the grid for a given outer polygon, depth, and zero-depth grid size.	21



Introduction

In elderly care, there is a continuous and serious concern for fall incidents. Elderly are more sensitive to falling and have increased risk to serious injury [1]. Often, they cannot get up or call for help, though it is often essential for them to receive timely aid. Current solutions, such as a help button worn around the neck, do offer some help, are inadequate if the person is incapacitated. For example if the person is in shock, unconscious, cannot find the button, forgot how or why to use the button, or took it off in a state of confusion.

In this project, we aim to create a non wearable-based, intuitive, and non-intrusive solution. Instead, we employ cameras to monitor the residents (i.e., the elderly in care). A workable product can increase resident comfort and safety while reducing confusion. Our contribution is split into two parts: (i) detecting when people fall in camera footage, thereby helping staff locate fallen people, and (ii) placing cameras such that the configuration has a suitable trade-off between low cameras (meaning low cost), and high viewing coverage.

In this report, we describe the goals, process and results of the design and implementation of this system. We present an overview of the original system in Chapter 2. In Chapter 3, we examine relevant research on a per-topic basis. We present our system design and subsequent implementation in Chapters 4 and 5. In Chapter 6, we consider and justify some ethical concerns with our solution. Our performance and user evaluations are presented in Chapter 7. In Chapter 8, we describe and reflect on the process of the project. Finally, Chapters 9 and 10 form our conclusion and discussion.

2

Problem Definition

In this section, we present an overview of the current system and describe the project goal. The project description as presented on BEPSys is listed in Appendix A and our derived requirements analysis is presented in MoSCoW form in Appendix B.

2.1. System Overview

The system provided by the project client is designed for use in resident care facilities for monitoring purposes. A resident is someone who receives care in a care facility, for example elderly people in a elderly home. The system consists of three main modules, a *front-end* web application, a *back-end* server and database, and a [REDACTED] embedded device (the *client*). We make a distinction here between the *project client* (i.e., Eya Solutions), and the *client* (i.e., the embedded device). An overview of the entire system is provided in Figure 2.1.

The front-end is a multi-platform web application [REDACTED]

[REDACTED]

1. [REDACTED]
2. [REDACTED]
3. [REDACTED]

The client is a [REDACTED] embedded system. The client is connected over the on-site locations through ethernet or Wi-Fi, but may be behind a firewall. The software on the client [REDACTED] and the project client has provided our group with configurations to cross-compile the software for use on the embedded platform. [REDACTED]

2.2. Project Goal

The goal of the BEP is to extend on a distributed information dashboard used in hospitals and elderly homes. The project requirements are described in more detail in Appendix B in MoSCoW-form. Our client requests that we extend this product by introducing: (i) a video processing component which performs face recognition, (ii) functionality for indoor positioning of people, (iii) functionality to automatically determine near-optimal placement of cameras, and (iv) automatic detection of fall incidents.

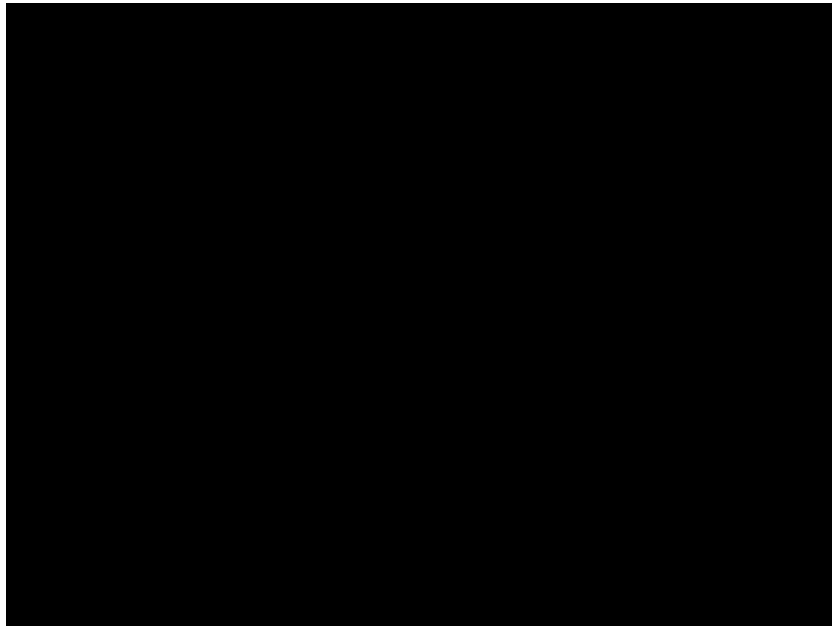


Figure 2.1: Conceptual overview of the system provided by the project client. Boxes indicate components and arrows indicate communication streams.

The end-goal of the project as a whole is to protect vulnerable residents by quickly alerting caregivers when residents have fallen. The finished product should identify users through facial recognition, perform pose estimation, and subsequently use pose estimation to determine when a fall incident has occurred. This three-phase process should occur automatically and in real-time. Moreover, the project client requests the system to be accurate and performant. Finally, the project client indicates the project should be easy to extend and should introduce a minimal number of new technologies.

3

Related Work

In this chapter, we address the related research pertinent to our Bachelor End Project. We investigate five distinct categories: (i) object detection through video streams, (ii) human fall detection based on video processing, (iii) facial recognition and authentication, (iv) modelling 2D building floor plans, (v) near-optimal placement of the camera objects. We discuss each category individually and conclude in Section 3.6.

3.1. Object Detection

In general, object detection concerns processing images and video material such that pixel-locations of displayed objects can be extracted [39, 52]. An additional challenge is the subsequent *classification* of the objects. In this analysis, we will focus on the first challenge, as our interest lies with detection of humans and human poses. Broadly speaking, there are two approaches to object detection: learning-based methods and rule-based methods. We discuss each category individually. Subsequently, we also discuss multi-camera object tracking in Section 3.1.3.

3.1.1. Rule-based Approaches

Rule-based approaches use data-set-independent decisioning to perform detection. Such approaches include image fragment *feature matching* and, more holistically, *template matching* [54]. In practice, feature matching approaches algorithmically pair object textural features of to the image fragment such that a probabilistic prediction can be made [30, 13]. In contrast, template matching perform matching on the whole image, suffering reduced accuracy in favour of robustness to foreground occlusion [47, 22, 23].

Manen, Guillaumin, and Van Gool propose a different method, which uses a randomised version of Prim's algorithm [36]. Prim's algorithm, in turn finds minimum spanning trees provided a a graph as input. Their approach has increased accuracy and speed in contrast to methods based on super-pixel groupings.

There are many other rule-based approaches, mostly focused on subtraction of two subsequent frames in video streams. These are further discussed in Section 3.2.

3.1.2. Learning-based Approaches

Learning-based approaches mostly focus on detection on image fragments which together form a coherent model of the world [52]. In contrast to rule-based approaches, the accuracy of the detection algorithm is grounded almost entirely in the quality of the training data. Moreover, training data is not always generalisable such that an accurate detection can be made in cross domain applications [16, 41].

Sliding-window approaches employ classifiers to scan over scaled image snippets and detect objects problematically [13, 56]. This type of approach is aimed mostly at the classification task but not at the localisation task [9]. Dollar, Belongie, and Perona propose a hybrid approach of sparsely sampled image pyramids computing object features [15, 16]. These features are computed through gradient histograms and together compose the detector at the full image scale. They gain an order of magnitude speedup compared to the baseline.

Gall, Razavi, and Van Gool present one mathematical approach to object detection on still images through random forests [19]. This method employs decision trees that are optimised with training data. Random trees are effective in regression, classification, or a combination. Each random forests consists of multiple

decision trees that map an *image patch* (i.e., an area of the image) to probability distributions for the target variables (for instance, class probability). The authors have tested their frameworks experimentally and conclude that their approach is comparably effective in contrast with the state of the art, while striking a balance between accuracy and training efficiency [20]. Tejani et al. have expanded on this work [52], introducing *latent* class Hough trees for RGB-D images (i.e, those with a depth dimension). By introducing latent variables derived from the input data, such as the depth gradient, the authors find their work can generate effective clutter occlusion masks. Subsequently, their approach is more resistant to cluttered environments.

Tang, Liu, and Kim propose a pedestrian detection framework which also employs Hough forests, but employ a novel node split function which reduces loss of discriminative information [50]. Their approach attains similar accuracy while gaining a performance increase. Similarly, Xiang et al. propose a pedestrian detection method which employs weak classifiers specified by image fragment templates. Hence, the learning element of each node is restricted solely to local features. Each weak classifier minimises the global loss individually by tweaking the sample weights.

Okada propose a discriminative model for part-based object detection [39]. Similar to Gall, Razavi, and Van Gool and Tejani et al., their approach uses randomised trees aiming to perform probabilistic discrimination of local features. However, Okada employs a novel training method, by splitting each node based on scores determined by discrimination power of the regression. Their approach yields improved accuracy.

One-class classification, originally proposed by Tax, is a method of machine-learning based classification focusing solely on positive instances [51]. In principle, the goal is to perform conservative classification such that potential negative instances are omitted. This technique has been subsequently expanded on [26].

Felzenszwalb et al. propose a model which incorporates deformable parts (e.g., hands and arms for humans) [17]. Similar to Tejani et al., their approach uses latent variables, but their method now also matches deformable models to images, which increases accuracy for non-static objects.

Bourdev and Brandt propose a method for addressing cascading problems (i.e., the loss of information at each processing stage) [10]. At each stage of the classifier optimisation, the classifier is calibrated such that a specific detection rate is reached. Their approach more quickly generates accurate detectors which are easier to train and more compact. Moreover, their architecture has tunable a speed-accuracy trade-off.

3.1.3. Multi-camera Tracking

Khan and Shah have proposed one of the more popular tracking algorithms [25]. In essence, they resolve occlusions and localise moving objects on scene planes (i.e., the floor) with introduced occupancy constraints. For example, the direction and location of people in an area.

Sato et al. load 3D-CAD floor plan models of buildings and subsequently perform multi-camera tracking of moving objects [45]. Using position estimation, they can estimate object locations for objects moving through rooms. They claim their model can realise “an accurate and wide area of object tracking for building security purposes.”. Their approach is independent of the object detection algorithm.

Sternig et al. use generalised Hough trees as developed by Gall, Razavi, and Van Gool and expand the method to multi-camera setups [48]. Their input data thus gains an extra dimension, and each input tensor requires more voting. As a result, there are fewer projection errors and the system gains increased tracking performance.

3.2. Fall detection

In this section, we address human falling behaviour, and how it can be captured through video streams. To detect falling, we need to know which types of falls there are, and which activities could look like falls, but should not be detected as such. Noury et al. [38] identified the actions presented in Table 3.1, where a positive outcome means that fall detector should detect it as a fall, and a negative outcome means that it should not.

There has been a lot of research in the field of fall detection. Mubashir, Shao, and Seed [37] surveyed fall detection techniques that used either wearable sensors, vision, or ambient methods and further classified them. We will focus on techniques that use vision, as that is what will be used in our system. These techniques were then categorised into the following groups:

1. **Body shape change and inactivity** are often used together, and generally follow the assumption that a fall is a quick change of body shape as seen by the camera, followed by a period of inactivity. One method used quite often is fitting an ellipse around the body. A fall might have occurred when this ellipse changes from a tall into a wide ellipse in a short period of time. Sitting or lying down would

Table 3.1: Falling activities identified by Noury et al. [38].

Category	Name	Outcome
Backward fall (both legs straight or with knee flexion)	Ending sitting	Positive
	Ending lying	Positive
	Ending in lateral position	Positive
	With recovery	Negative
Forward fall	On the knees	Positive
	With forward arm protection	Positive
	Ending lying flat	Positive
	With rotation, ending in the lateral right position	Positive
	With rotation, ending in the lateral left position	Positive
With recovery	Negative	
Lateral fall to the right	Ending lying flat	Positive
	With recovery	Negative
Lateral fall to the left	Ending lying flat	Positive
	With recovery	Negative
Syncope	Vertical slipping against a wall finishing in sitting position	Negative
Neutral	To sit on a chair then to stand up (consider the height of the chair)	Negative
	To lie on the bed then to rise up	Negative
	Walk a few meters	Negative
	To bend down, catch something on the floor, then to rise up	Negative
	To cough or sneeze	Negative

be slower changes of this ellipse, whereas tying shoes or picking something up would not be followed by inactivity. The basis of this technique is quite simple but can be improved using machine learning [5, 6, 8, 29, 33, 43, 63, 64].

2. **Posture** techniques try to detect which posture someone is in. For example, a sudden change from standing to a lying posture would imply a fall. These techniques often use neural networks to classify the posture and, as such, will require training data. This approach is used in many different works [14, 24, 32, 34] [53, 55, 62, 63].
3. **3D head change** technique is similar to the previously mentioned ones, but tracks the head rather than the entire body. A 3D ellipse is fitted around a head, which is then used to determine motion and velocity. This technique often requires multiple (calibrated) cameras. Used in [7] [44] [46] [61].
4. **Spatiotemporal** techniques use changes in lighting to detect motion, which then can be used to detect falls. This does not explicitly track the body or head, which makes it different from all other techniques. This technique is infrequently [18, 40].

Ideally, the fall detection system would meet the following requirements:

1. Work in any lighting condition;
2. Work on any skin colour;
3. Work on people of varying age;
4. Work on people that do not stand up straight (e.g. walking with a stick or walker);
5. Work on all types of falling
6. Does not detect non-falling events as fallin;
7. Does not require special cameras (i.e. depth, or very wide-angle);
8. Does not require multiple cameras per room (to reduce costs);
9. Does not require camera calibration;
10. Can be run on a Pine A64, potentially with multiple cameras;
11. Implementable within the project period.

Of all these techniques, we think either body shape change or posture techniques are best suited to meet these requirements, as these can work with regular cameras and should work reasonably well regardless of

lighting or skin colour. The other methods often require calibration or specific camera setups, which would increase costs and set-up time.

We decided to use OpenCV to process video and perform fall detection. OpenCV is a free, open source computer vision library used in many different projects, including some of our own. There are other computer vision libraries available with similar (or less) capabilities, but we have no experience with those, and we do not think that OpenCV lacks any functionality that other libraries might have.

3.3. Facial Recognition

In this section we focus on aspects of facial recognition that can be difficult in our situation. There are many aspects influencing the accuracy of facial recognition, some possibilities we can think of are the angle and distance of the camera, camera quality, whether the video is in colour or grey-scale, skin colour and age of individuals. The last two are dependent on the facial recognition algorithm used and the data used to train that algorithm. The most important factor we have is distance, since a few cameras mounted on walls will have to oversee entire rooms.

The further away the person is from the camera, the smaller his/her face on the video. This means the software has less pixels to work with to determine the person's identity, resulting in a lower recognition accuracy [31]. Wheeler, Weiss, and Tu solve this problem by finding individuals in view of a stationary camera and then use a pan-tilt-zoom (PTZ) camera to zoom in on their faces to get a high resolution image to work with [57].

Lin et al. try to solve the problem without special camera setups, by working purely on the obtained video material [31]. By processing several consecutive frames, they attempt to create a higher resolution image to work on. They report better results than interpolation methods and other algorithms, but the accuracy on 180×144px images is still only 50 to 70%, while participants look straight into the camera. This goes to show how important an initial resolution is to recognition accuracy.

Since cameras are generally mounted high on the wall to have a better overall coverage and elderly are often looking down, we should also consider what this viewing angle does to the ability to recognise faces. Zheng and Yao discuss several papers that attempt facial recognition at different angles than the usual frontal one [65]. They conclude that these approaches still give many false results. They themselves suggest a new approach to recognise faces in different kinds of angles. They report an improved true positive rate of 71.87%, but this is over different kinds of rotations and it is unclear what the recognition rate for the downward looking angle is.

This all combines to show that in a simple, general coverage setup of cameras, accurate facial recognition is a hard thing to achieve. The distance of the camera to faces to recognise decreases the accuracy. We did not find any conclusive articles on the viewing angle, but since rotation of the head in general is considered a negative impact, we are convinced that this also holds for our situation [65]. One approach to solve possibly this problem is to train an algorithm specifically for this situation, but for that you need a lot of training data, which we do not have.

3.4. Floor Plan Modelling

In order to determine to the optimal camera placement (Section 3.5) in a building we first need to have a model of the floors of this building. We will use a floor plan to create this model. For the modelling of the floor it is essential that we can determine where walls, doors and windows are. Ideally, we want to divide the floor plan in to different rooms using this information so that we can use this information later in the camera placement. This process can be done manually, by loading in a floor plan and indicating where walls, doors, and windows are through the input from a user, or by an automated system.

3.4.1. Manual

Our proposal for manual modelling the floor plan is as follows: The users can load an existing floor plan into a web-interface. Then the user can indicate manually on the plan corners of a room to create a model of where the walls are. The interface will then show these walls. Then the user has a possibility to add doors and windows to the walls. The user also needs to be able to indicate the scale of the image. This model could

Table 3.2: Results of relevant platforms presented by other researchers.

Work	Metric	Detection Rate [%]	Rec. [%]
Ahmed et al. [2].	Without semantic division	89	79
	With semantic division	85	82
Macé et al. [35]	Average	85	69
	Standard deviation	1.6	1.8

be extended by also modelling the different regions that make up a room. There are various tools for creating floor plans.¹²³⁴⁵ The tools that support importing images use a similar approach.

3.4.2. Automatic

Macé et al. describe a system to detect rooms in a variety of architectural floor plan images [35]. It consists of two steps: extracting the primitives, (i.e. walls) and recursively decomposing the room to get almost convex regions. Wall detection is done through a combination of classical Hough transforms (line detection) and image vectorization. Image vectorization is used to divide the images in segments to generate hints about the lines of the image. These hints are used for the Hough transforms to detect lines. To deal with the architectural plans, which often contain wide lines for walls, the line detection is applied on the contours of the walls. Determining rooms consists of determining the walls that make up its outer boundary. Macé et al. uses a top-down approach that consists of recursively separating rooms by linking gaps between walls.

The accuracy of the system was measured in detection rate and recognition accuracy. “The detection rate is roughly, the percentage of ground truth entities that are detected by the recognition system.” [42] and the “Recognition Accuracy indicates, roughly, the percentage of detected entities within the result file that have their match in the ground-truth entities.” [42] The results are shown in Table 3.2.

Ahmed et al. propose an automated analysis of floor plans that extracts both structural and semantic data from the floor plan [2]. The flow of the program is as follows:

1. The input is an image of the floor plan.
2. The text is extracted from the image
3. Thick lines are extracted from the image these form the external walls.
4. Medium lines are extracted from the image, forming the inner walls.
5. Thin lines are extracted from the image, forming the symbols
6. The thick and medium lines are combined to form a wall image
7. Wall detection is performed on the wall image using the method described in [49]
8. Gaps between walls are closed.
9. The outer boundary of the floor plan is detected
10. The boundary image is combined with the walls
11. Symbol detection is applied on the thin lines
12. Using the symbols gaps are closed at door locations
13. Rooms are detected by checking closed areas.
14. The extracted text is used to label the rooms
15. The rooms are divided based on the labels.

The results of the system are shown in Table 3.2.

3.5. Camera Placement Optimisation

Camera placement can be optimised for different goals. This results in some very different approaches to solving the problem and because of that, there is no good benchmark to compare different algorithms for

¹<https://planner.roomsketcher.com/>

²<https://www.floorplantooll.com/>

³<https://www.floorplanner.com/>

⁴<https://www.gliffy.com/examples/floor-plans>

⁵<https://www.smartdraw.com/floor-plan/floor-plan-designer.htm>
<https://ezblueprint.com/>

optimal camera placement [28]. The best way to choose then, is to consider what you need and which solutions fits those requirements the best.

When discretising the search space to reduce the problem size and thus the computation time, one arrives at the underlying NP-hard problem Set Cover, since you need to pick camera locations that cover all (or most) of the grid points [28]. This shows the infeasibility of achieving the global optimal solution.

3.5.1. Optimisation Approaches

Albahri and Hammad [4] give a pipeline for finding a very educated optimum placement for security cameras. This placement keeps in mind obstacles in the building and places where you should not place cameras, as well as priorities for covering certain areas. The drawback of this method is that you need very specific building information, in this case the Building Information Model (BIM), which is only available for some modern buildings. Additionally, the entire process is quite computationally heavy and will thus take long.

[12] uses a bee colony algorithm to find an optimal solution. Here, scouting ‘bees’ are sent out to random locations. Each location is evaluated and other locations near good locations are explored more, to find the optimal locations. The algorithm can be applied the cases where you have a set amount of cameras and want the optimal coverage, or finding the lowest cost solution given a minimum coverage percentage. The paper describes a 2D map, but it could easily be adapted to a 3D situation.

Given the underlying NP-hard problem Set Cover (SC), Kritter et al. consider a myriad of proposed solutions to solving SC to solve the Optimal Camera Placement problem [28]. Considering the benchmark performance of these algorithms, they conclude that the solution proposed by Gao et al. [21] is the most efficient algorithm for solving the SC problem with equal weights for subsets (cameras), which they consider most analogous to Optimal Camera Placement. In the case where you want to use cameras that all cost the same, or if the costs of the cameras do not matter, the unicast version of SC is indeed a direct representation of the problem. Gao et al. show that the unicast versions of SC problems are harder to solve than their differing cost versions [21]. Their algorithm, which was marked as the best by [28], finds 100% coverage with as few cameras as possible. The reported results outperform several other good algorithms in the optimality of the solution, if not always in computation time. This algorithm does not take into account regions with higher priority nor does it consider solutions with high but not complete coverage, but could possibly be adapted to do either.

Aissaoui et al. describe an algorithm to optimise camera placement in a room to capture certain movements [3]. These movements need to be modelled to be inputted in the algorithm, so this algorithm needs a lot of preparation. Because of this, this algorithm is less suited for our goal.

Many approaches to target tracking use mobile cameras to find and keep the targets in view [28]. This might not be the optimal solution in the scenarios posed by our problem, where you are interested in tracking (almost) everyone in the building and people (we assume) are generally spread about the building.

3.5.2. General Observations

The goal of the camera placement influences what requirements the placement has [28]. That is, in terms of resolution or distance from the target. The requirements should address at least the following issues:

1. Obstacles occluding the view [4];
2. Vibrations and heat locations where you should not put cameras, since they lower the camera’s life expectancy [4];
3. Reflecting surfaces like windows which affect visibility [4];
4. Areas with higher priority [4, 28];
5. Areas where you should not place cameras, like bathrooms [4];
6. The properties of different camera types [4, 12, 28], such as (i) cost, (ii) viewing angle, (iii) pan, tilt and zoom abilities (PTZ cameras), and (iv) resolution;
7. Runtime constraints of the algorithm.

Kritter et al. describe a general series of actions to take in the process of designing a camera placement optimisation algorithm [28], which we can follow for our own design:

1. Decide how to model the area/building. 2D or 3D? Do you discretise or keep it continuous? Does it include objects that could cause occlusion?
2. Choose a sensor representation. How do you model the location and orientation of the camera? How do you model its field of view? For example, a triangle is easy to work with, but hides a blind spot directly below the camera, while a circle or cone is less representative of the real situation.

- 3. How do you check what can be seen by which cameras? Simple point-in-triangle and similar techniques do not take into account occlusion.
- 4. How do you generate and evaluate different solutions?

3.6. Research Conclusion

In this section, we conclude each of our research topics.

[REDACTED]

4

System Design

This chapter details the design of our system, which is comprised of two new modules extending the existing system.

1. The floor plan editor, which runs on the front-end. This allows administrators to design a floor plan, place cameras, and optimise camera positions. Caretakers can then see the floor plan and track resident locations.
2. The client program, which inputs video feeds and performs object detection, person tracking, and fall detection based on these videos.

The complete system with our contributions can be seen in Figure 4.1. Our contributions are highlighted in green.

4.1. Object Detection, Tracking, and Fall Detection

Object detection is done in three phases. The first phase consists of separating the foreground from the background, this process is called background subtraction. This background subtraction returns an image where black pixels represent background and white pixels represent foreground. The next phase uses morphology operations to remove small sections and connect larger contiguous sections. The last step of object detection detects the contours of the remaining foreground sections.

This object detection is performed per frame, resulting in a list of contours every frame. Objects need to be coupled to one of these contours to track them and get properties such as speed or the location history. This is done by comparing bounding boxes of known objects to bounding boxes around the newly found contours. When a bounding box is close in size and location to the last seen bounding box of an object, it is considered the same object, and it is added to the object's history.

The fall detection is based on the properties of the tracked object. Each tracked object has a state, either normal, falling or fallen. Based on the properties of the tracked object, it is determined whether the object is falling, lying or acting normal. Based on these activities, the state of the object changes. The transition between states is shown in Figure 4.2.

Once a tracked object has been in the fallen state for a certain time, we can be sure the person has actually fallen and is not getting up on their own. At this point, a notification is sent to the caretakers, specifying what room they need to check.

4.2. Floor Plan Management

The floor plan management will be integrated in the existing system provided by the client. [REDACTED]

[REDACTED] We will add a component in the front-end and a component in the [REDACTED] in the back-end as can be seen in figure 4.1. In the front-end we extend the existing web-application with a graphical interface for modelling a floor plan.

The following features will be implemented in order to have the basic ability to model the floorplan:

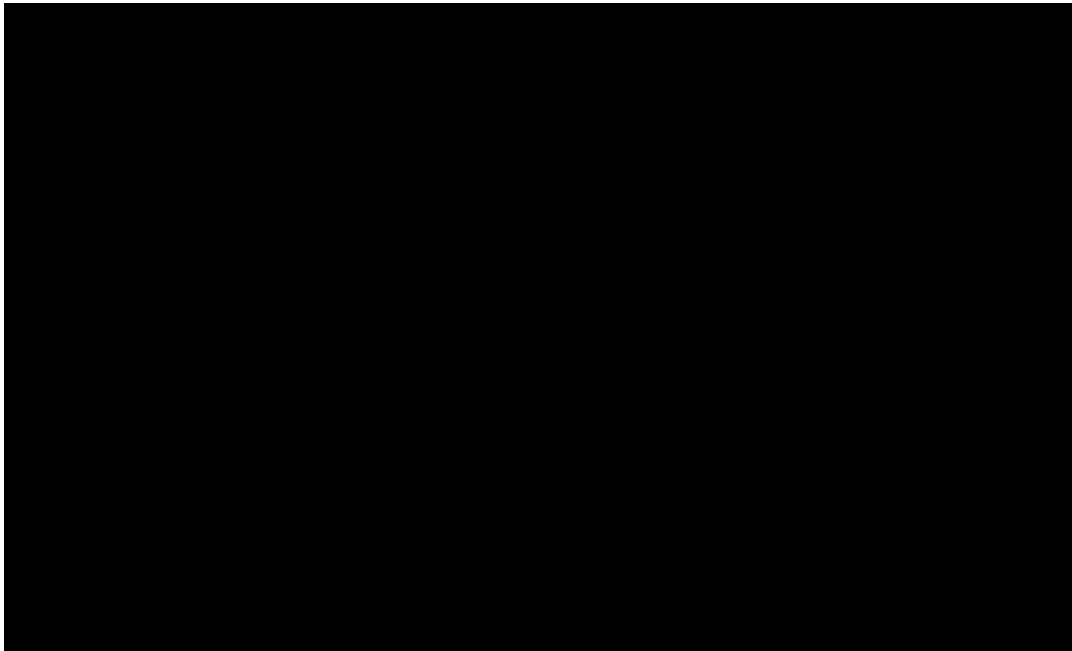


Figure 4.1: Conceptual design of our system, with our contributions highlighted in green. Arrows indicate communication between components.

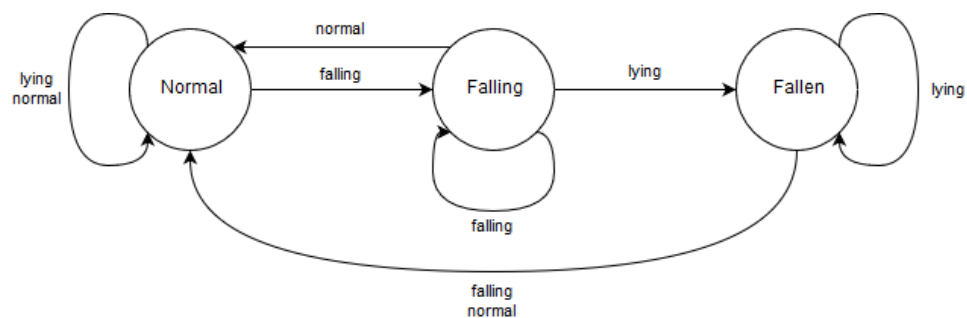


Figure 4.2: State diagram indicating the states of our tracker object. The states and transitions model the physical states of how objects fall.

1. adding and removing walls,
2. adding and removing doors,
3. adding and removing cameras,
4. adding and removing wall obstructions,
5. labelling rooms,
6. tagging rooms as 'forbidden' so that camera's will not be placed here, and
7. saving and Loading floor plans from the database.

The graphical interface will be added to a new page in the web-application. [REDACTED]
 [REDACTED] The editor will contain various tools that each will handle a specific part of the floor plan functionality. For example we will have separate tools for adding walls and adding doors. [REDACTED]
 [REDACTED] We will follow the Model-View-Control principle in that we split the view from the model. All the tools will be implemented using different implementation of a module interface that contains methods for handling user input. These modules will then be able to be switched out by selecting the corresponding tool to this module on the toolbar.
 [REDACTED]
 [REDACTED]

5

System Implementation

5.1. Implementation Overview

In this chapter, we will elaborate on the implementation of the system.

Object tracking and fall detection are implemented in a standalone module for the [REDACTED]. For many of the image processing actions, the open-source library OpenCV¹ is used. The class diagram for this module can be found in Figure 5.1.

The floor plan editor and the optimisation of camera placement are implemented [REDACTED] as a separate tab in the existing dashboard.

5.2. Object Detection

Object detection consists of three separate modules, which act upon the result of the previous module. These three modules are background subtraction, morphology, and object tracking. Before these modules are applied, there is a pre-processing step. The implementation of all of these models are explained in the following subsections.

5.2.1. Pre-processing

The raw footage from the camera is often quite noisy. There is noise from the camera sensor itself, but also from temporal changes such as shaking of the camera. First, the temporal noise is reduced by taking the average of multiple frames. To this end, the stream has a buffer holding a set amount of frames. Next, other noise is removed by applying a Gaussian blur filter over the whole frame.

5.2.2. Background subtraction

Background subtraction is used to separate the foreground from the background. [REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

1. [REDACTED]
2. [REDACTED]
3. [REDACTED]
4. [REDACTED]

¹<https://opencv.org/>

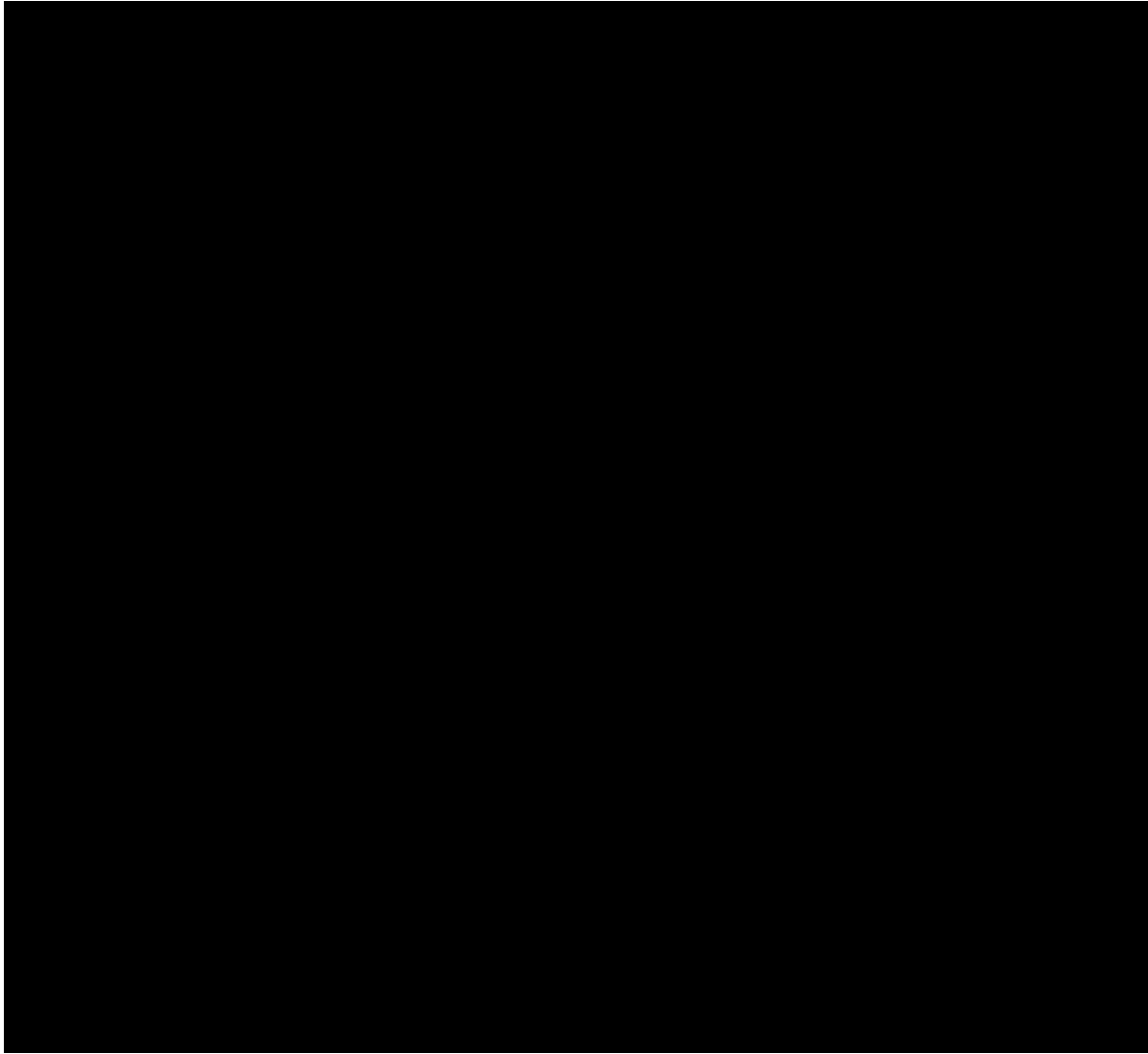


Figure 5.1: The class diagram for the video processing module

- 5. [Redacted]
- 6. [Redacted]

[Redacted text block containing multiple lines of obscured content]

[REDACTED]

5.2.3. Morphology

The image produced by the background subtraction still contains much noise. Small changes in light create small spots of pixels falsely detected as foreground and moving objects are not completely detected if they locally look too much like the background. To get rid of this noise, some morphology is applied. The process of closing attaches small objects close to each other, to properly detect whole objects that are partly seen as background. In the next step, opening removes small objects in the image. At this point, we are left with mostly just the moving object, but it is often still in several parts. To connect those, another round of closing is applied.

5.3. Object Tracking and Fall Detection

The process of tracking and detecting has a lot of tunable parameters that together determine how successful the system is. At the moment, these are tweaked manually. We provide a complete evaluation of our system in Chapter 7.

After the object detection, the result is often one or multiple large blobs of objects on the foreground, assuming there is an object to track. However, this is just for a single frame, and does not tell us anything about the movement or history of an object. This is where object tracking comes in.

[REDACTED]

5.4. Floor Plan Management

The floor plan editor supporting the camera optimisation was implemented in JavaScript and communicates [REDACTED] with the [REDACTED] database. It was integrated in the existing system provided by the product client. The UML for the front-end is provided in Figure 5.2.

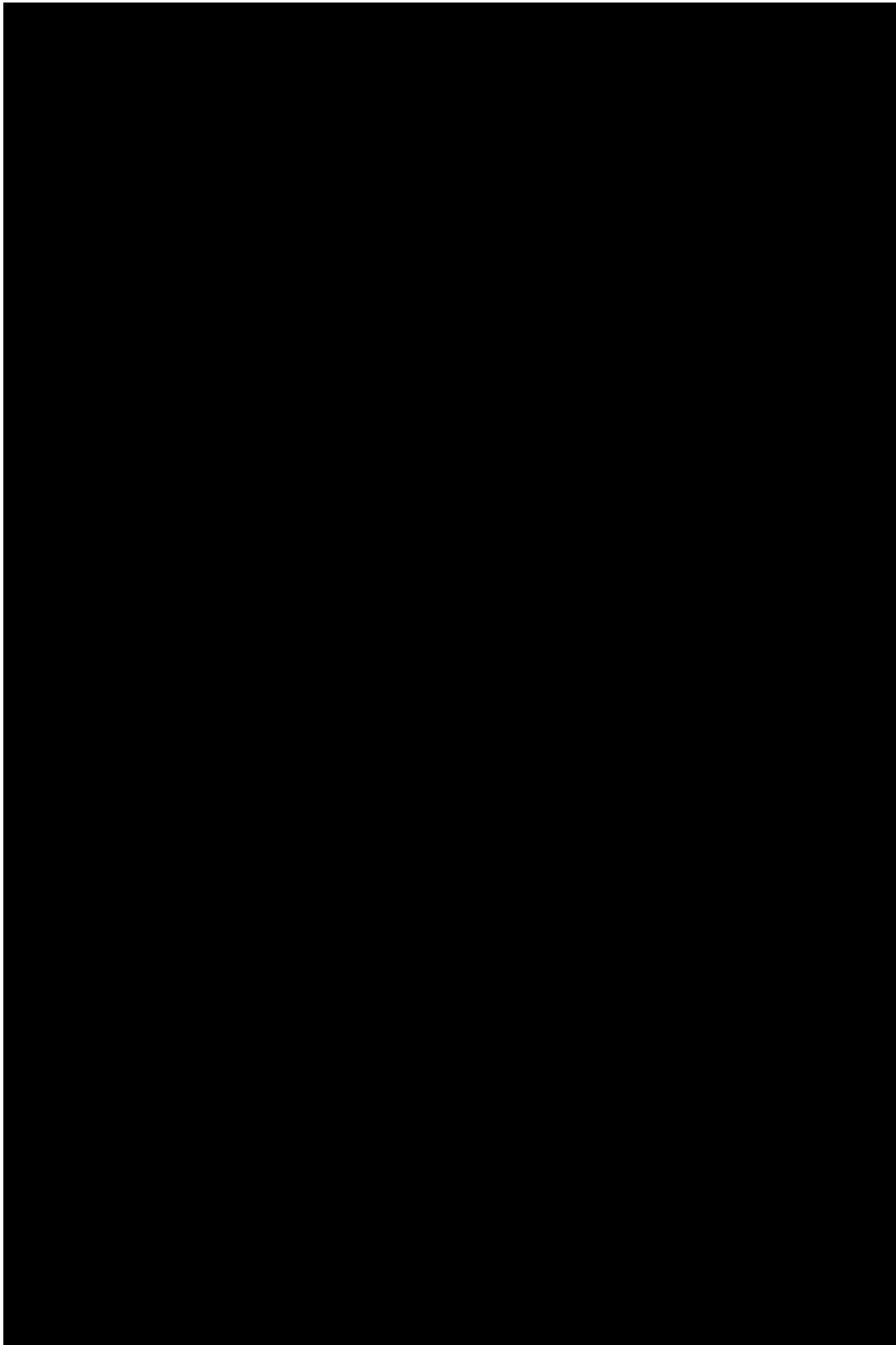


Figure 5.2: The UML class diagram of the floor plan module.

The floor plan contains one Editor instance. The editor contains a list of modules, and a list of floors. We use a decorator pattern to switch the functionality of the current editor by changing the current selected module. The editor subscribes to events such as keyboard press and mouse press. These events are then

propagated to the current selected module, the toolbar and the floor selector. The modules each have their own functionality that interacts with the floors of the editor.

The toolbar handles the interaction of changing out and displaying the selected module and the Floor Selector handles the traversal of different floors.

The model contains information about location of walls, doors, and cameras. The locations are converted to screen coordinates from an offset and scale stored in the editor, this way the canvas can be moved and scaled. The editor automatically saves once the floor plan is edited.

Once the page is loaded or the user goes to the next floor then another request is made to load the latest version of the saved floor plan.

The floor plan is modelled using Multiple Floors. A floor consists of walls, and rooms. A room consists of a polygon, an optional room label, and a Boolean that indicates if this area is 'forbidden' (for indicating that no camera's should be placed in this area). The polygon is a set of walls that make the outer boundary of the polygon. A wall consists of two points that indicate the end points of the wall and a list of 'features'. A feature is an object on a wall like a door, camera or an obstruction. The obstruction is to indicate area's where camera's cant be placed. In the model walls are never intersecting. When a new wall is placed it will automatically split the walls into smaller walls that are non-intersecting. The splitting of the walls is done by finding all the intersection points and then sorting the points in order. Then between all the consecutive points a new wall is created. The existing walls that are intersected are split into two walls at the point of existing sections. The features of this wall like doors and cameras will be copied to the one of the new walls.

When a wall is removed it will be checked if two walls can be merged into one wall. This will happen when two walls: (i) are parallel, (ii) have a common end point, and (iii) no other wall is connected at the common end point. Rooms are automatically detected when an user clicks on a room that was not detected yet. When a room is clicked, the closest wall to the click will be found. Then to detect the room it will traverse from the closest wall connected walls until it reaches the starting wall again. The next wall in the traversal will be determined by the smallest angle between the current wall and the connected walls in opposite direction of the traversal. For example, when the walls are traversed in a clockwise order, the next wall be the wall with the smallest anti-clockwise angle between this wall and the current wall.

5.5. Camera Coverage Optimisation

In this section, we present our approach to optimising the localisation and direction of the virtual cameras such that the viewing coverage is high and the total number of cameras is low. We employ a domain space genetic algorithm which iteratively discovers new placement configurations with better fitness across generations. Our domain space is dynamically generated by the layout of a floor (i.e., the position and lengths of the walls). The genetic algorithm tweaks only camera objects, including removing and adding cameras where necessary to improve fitness. Furthermore, our implementation features tuned parameters for quick convergence to an acceptable setting, as we have discovered that global maxima, while differing greatly in parameter value, differ only little in fitness. We further describe our implementation and trade-offs below. The performance of our genetic algorithm is analysed in Section 7.3.

5.5.1. Genetic Algorithm Fitness

In conversation, we have discovered our client desires a suitable trade-off between two optimisation goals: (i) a minimal number of cameras, and (ii) a high coverage of the viewing area of the cameras. In turn, our input domain is solely the placement and orientation of the cameras. Upon further investigation, the client notes that the second goal may have a constrained lower bound for acceptable configurations. For example, acceptable solutions must all have 75% viewing coverage. To this end, our fitness function has discontinuous range, resulting in a two-phase optimisation process: (i) attaining a view coverage of 75%, and (ii) reducing the number of cameras, while incrementally increasing view coverage.

Our fitness function is presented in Algorithm 5.1. Our fitness function solely optimises for camera coverage so long as the coverage is less than 75% of the total floor area. Above 75%, the fitness function gains a bonus for *efficiency*. In this case, the coverage over the camera utilisation in terms of the maximal number of cameras. Hence, there is a point of discontinuity at a fitness value of 0.75. In practice, the algorithm quickly attains a coverage of 75% within only a few generations, and then does not

Algorithm 5.1. Computes the fitness of a given individual.

procedure FITNESS(i)



Figure 5.3: Four stages in grid generation: from a grid to a finer grid. Red dots indicate newly generated points in the grid stage. The origin in this Cartesian system is the top-left corner and the y-axis is inverted.



trade to lower coverage in favour of fewer cameras. The weighing of the efficiency bonus ensures this does not happen unless a significant advantage is found, which is unlikely given our optimisation strategy of quick convergence in favour of attaining global maxima.

5.5.2. Determining the View Coverage

The fitness function depends on the determination of the view coverage of a given floor. In general, we have identified two approaches, one exact algorithm and one approximation algorithm. Given our development constraints (in particular, time), and our use case, our implementation employs the approximation algorithm, which is presented in Algorithm 5.2 and 5.3. Our algorithm calculates the coverage by systematically selecting points on a grid overlaying the floor plan. For each point, we subsequently perform a ray trace to each of the cameras to determine its visibility. In particular, whether or not it is occluded by other walls. Then, we approximate the coverage by the number of visible points divided by the total number of points.

One challenge of this approach is determining the grid scale. Should we compute the coverage with 100 units between each of the points, or 1? We tackle this issue by introducing arbitrary grid precision. Our algorithm automatically increases grid density such that the approximation error is less than one percent. We implement this mechanism by generating a more dense grid from any arbitrary grid.

We can repeat this process until we reach the desired precision. An example of three iterations of this process can be found in Figure 5.3.

Given this process of generating an arbitrarily dense grid, we can achieve an arbitrary precise approximation by continuously increasing the grid density and refining the ratio between visible and occluded points. This is the essence of Algorithm 5.2. We abort the process when the most recent phase has not altered the approximate view coverage more than one percent.

5.5.3. Problem-space Genetic Algorithm

With the fitness function presented in Algorithm 5.1, we now describe our approach for view coverage optimisation. Given that a floor plan may have any layout in number and lengths of walls, as well as fixed constraints such as private rooms and glass walls where cameras may not be placed, calculating the optimal camera positioning is nontrivial. Hence, our approach employs a genetic algorithm which iteratively improves a subset of possible configurations (the *population* consisting of *individuals*). Genetic algorithms typically represent individuals as a fixed-length encoded bit strings in order to perform mutations and crossovers. One drawback of this approach is that it can not cleanly represent individuals with unbounded encoding length. Our

Algorithm 5.2. Approximates the view coverage of a given individual with $< 1\%$ accuracy.

procedure CALCULATECOVERAGE(i)

Algorithm 5.3. Calculates the points for the grid for a given outer polygon, depth, and zero-depth grid size.

procedure GENERATEGRID(o, d, δ)



Table 5.1: Overview of hyperparameters of our genetic algorithm.

problem does indicate such individuals can exist, as there may theoretically be infinitely many cameras for any given floor. Variable-length representations do exist, but implementation is more strenuous, employing for example tree-like data structures (cf. [58]). More issues arise as we consider the encoding. To keep closely related individuals grouped, genetic algorithms can use Gray encoding, but this approach does not work with variable-length chromosomes. Regardless, typical genetic algorithms require encoding and decoding of individuals from the problem space to gene space.

For these reasons, and to save valuable programming time, we instead use a *problem-space* genetic algorithm. Our algorithm does not encode individuals in any particular form for mutation, but instead mutates individuals in-place in the problem space. In particular, our implementation supports four mutations:

1. [REDACTED]
2. [REDACTED]
3. [REDACTED]
4. [REDACTED]

[REDACTED] Hence, our mutations covers the entire problem space, a necessity for a well-functioning genetic algorithm.

Details on the hyperparameters for our implementation can be found in Table 5.1. Overall, the computation of each generation follows a four-step procedure:

1. Perform selection copy, effectively multiplying individuals with high fitness.
2. Perform mutations on each individual.
3. Calculate the fitness for each individual.
4. Sort the population from highest fitness to lowest.

5.6. Feedback from Software Improvement Group

We submitted our code to Software Improvement Group (SIG) for code review and evaluation during the sixth week of our project. The verbatim feedback from SIG can be found in Appendix C. SIG evaluated our code based on their quality model, which results in a score of 4.0/5.0. This indicates that the maintainability of the project is comparable to the industrial average. The main points of improvements were unit interfacing and unit complexity.

Considering unit interfacing, we received the feedback that some methods have a high number of parameters. This may indicate a lack of abstraction and could cause unclarity when referring to these methods. Parameters that are related to each other can be grouped together into one parameter object. For example when you have a location on a plane that can be passed to a method, you could introduce an object which contains the x and y coordinates instead of two separate parameters for both the x and y coordinates. We identified several cases in our code base where it was appropriate to replace function parameters with compound objects. For example, we found instances where we already had objects that contained both parameter types in conjunction. Instead of passing a compound object, we passed both parameters individually.

Vector objects are a prime example. The x-coordinate and y-coordinate of the vector were often passed to a function instead of the vector object itself.

For unit complexity we received the feedback that some methods had too much embedded functionality; these methods should be split into smaller functions. We identified that most of these functions were functions that handled user input, such as key presses and mouse clicks. Depending on the event and the state of the editor, a lot of computation needs to occur. We split those method into smaller methods that each contain a smaller portion of the functionality of the original method.

The last remark was that no (unit-)tests were found. Although some tests were present, there was no automated script that executed the test code, which could explain why their quality model did not detect it. The primary reason why we did not have many automatic tests was that our code was made for in-browser execution. We improved this through use of a library that mocks the browser and performs automated testing. We also included a script that automatically runs all unit tests.

Our improved code was submitted in the ninth week of our project. The feedback we received on the code submitted in week nine was that the feedback from week six was implemented as suggested and that the maintainability of our code base improved as a result of this.

6

Ethical Implications

A major ethical concern about this application is the privacy of residents. Residents are filmed throughout the day and often also in personal spaces. These concerns are addressed at multiple points in the application. Firstly, the placement of cameras. The administrators that build the digital representation of the floor plan can specify rooms where no cameras should be placed, such as bathrooms. Secondly, only caretakers and administrators can see the video feed from the cameras. It is appropriate for them to see the video feeds, since they need it to be able to provide proper care for the residents. Consider the event that the system detects that someone has fallen. It can be argued that it is shameful to show elderly in such a compromising situation, yet it is essential for the ability of the caretakers to help them. A caretaker can open the feed on which the fall was detected to see how serious the injury is likely to be. Additionally, only caretakers can see this, who will already have to see the situation in real life in order to help.

Care facilities using our system should make their own privacy agreements with residents or their legal representatives, staff and visitors.

Another ethical concern is the effects of failures of the system. Our system is not perfect. If the system reports a fall while there is none, not much harm is done. A caretaker can open the video feed of the camera that reported the fall and conclude that there is nothing wrong. This costs a few minutes at most. If the system would throw many false positives, the danger exists that caretakers will not take the incident alarms as seriously as they should. The only real way to prevent this is by keeping the true positive / false positive ratio as good as possible.

If a fall is not detected, implications could be a lot worse. This is why care facilities should not trust solely on our system, but also take other measures. Other forms of detection should be in place, such as worn help buttons, sensors on floors and caretakers checking up on people. This is the responsibility of the care facility.

7

Evaluation

7.1. Fall Detection

In order to be able to fairly evaluate the performance of the fall detection functionality, a section of 20% of the video database was set aside for verification. These videos have not been used when tweaking parameters for fall detection performance, and so simulate new data for the system.

We considered two options for evaluating the accuracy of the system. Firstly, we could look at the states the system reported (see section 4.1) and compare those to the states of the ground truth. This would provide overlap percentages. Secondly, we could look at when the system reports a fall (in the form of a notification) and compare that to the ground truth. This would provide a count of true positives, false negatives and false positives. We decided to opt for the second version, since this metric more closely resembles the goal of the system. The goal is to accurately tell when someone falls, not the state each person is in at any point.

The product client requested that the system looks if a person is getting up by themselves before notifying caretakers. To this end, we determined a specific period in which we want the notification to happen. This period starts a set amount of seconds from the end of the fall and lasts for about a second. Any notifications send before or after this period are regarded as false positives. For evaluating purposes, we set the start of the period to 20 frames after the end of the fall. This is rather short, but our testing videos are too short to allow for a longer period.

7.1.1. Results

The results for the training set, the set that the system was trained on, are as follows: 71.1% of falls recognised, being 54 out of 76 falls. 28.9% of falls go undetected. On average, the system reports 0.6 false positives per minute on this data set.

The results for the test set are: 33.3% of falls recognised, being 6 out of 18 falls. 66.7% of falls go undetected. On average, the system reports 0.5 false positives per minute on this data set.

7.1.2. Limitations of the Data Set

Unfortunately, we had to work with a rather small data set, published by Charfi et al. [11], because the client did not have any data for us to work with. This data set is somewhat biased and is not the best fitting for our use case. The most obvious limitation is that the videos only contain adults that imitate elderly, not elderly themselves. Next to that, they are mostly white and male and do not use tools such as walking sticks or walkers. Also absent are other moving objects, such as pets or television images. There is also an over-representation of falls versus non falls in the data set, making the result metric a bit skewed.

Each video also only has one person in view. This might be realistic for private rooms most of the time, but it is uncertain how the system will react to two people in the video. Additionally, almost all videos start with the person in the video. Because of this, the first move can give off false positives because to the system, the person seems to suddenly appear from the background. This is very unlikely to happen in a real situation that this system is meant for. Lastly, almost all videos are fairly short and stop shortly after the fall. This is unrealistic for our use case.

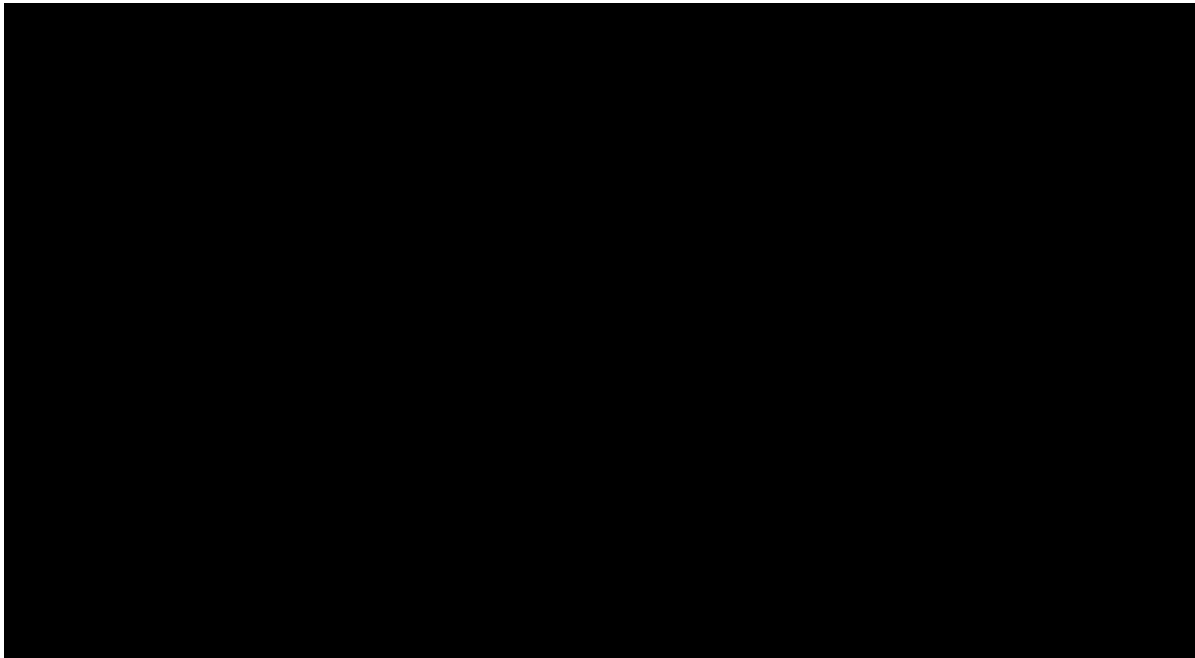


Figure 7.1: The user cheat sheet. Each icon represents a tool in the tool bar.

All in all, this data set is reasonable for giving an idea of the functionality of the system, but not very suited for definitive results. Since this set was used for optimising variables, if a better data set was to be created, this optimising has to be performed again.

7.1.3. Implications of results

The results our system achieves are not very reassuring. Nevertheless, it is still a good addition to existing detection methods, since it might catch incidents that existing methods do not. However, our system is clearly not perfect and should thus always be used in combination with other detection methods and caretakers checking up on residents.

Some things to note about these results is that false positives in empty rooms are very unlikely. A caretaker will thus almost never be send to check up on empty rooms, so any trips undertaken due to this system will almost always be useful in some way, since they will still check up on residents.

7.2. Dashboard User Test

To evaluate the usability of the floor plan editor we asked someone who was not familiar with the system to perform some tasks. We provided the users with a cheat sheet with descriptions of the functionality of all the available tools as seen in figure 7.1.

The user was then asked to perform the following tasks:

1. [REDACTED]
2. [REDACTED]
3. [REDACTED]
4. [REDACTED]
5. [REDACTED]
6. [REDACTED]
7. [REDACTED]
8. [REDACTED]
9. [REDACTED]
10. [REDACTED]
11. [REDACTED]
12. [REDACTED]
13. [REDACTED]

14. ████████████████████

Once the task were completed the following questions about the system were asked:

1. Which task was the most difficult one and why?
2. Which task was the least difficult one and why?
3. Was the interface intuitive to use?
4. What would you want to see explained more explicitly?
5. What could be improved?
6. Additional remarks.

The results from the user test showed that it can be difficult at times to use the editor. Especially drawing of the walls was difficult. It took multiple tries to draw the wall on the correct position, since it was not clear to the user that you need to click on the two endpoints of the wall to create a wall instead of dragging the mouse. Additionally, the user had trouble properly connecting walls, resulting in gaps left in the rooms. Leaving those gaps resulted in that no room could be found, at which point the user didn't notice that this was because of the gaps in the walls. The task of uploading an floor plan image and then copying this floor plan was the most difficult task because the floor plan contained a lot of small walls that needed to be added. Additionally, it was not clear that obstruction should have been placed on walls, which resulted in the user trying to draw obstructions in invalid locations. We used this feedback to update our system and improve the user friendliness of the system.

7.3. Floor Coverage Optimisation

In this section, we present our performance testing experiments for the coverage optimisation algorithm discussed in Section 5.5. We run two experiment sets: (i) overall performance experiments, composed of ten optimisation runs while measuring total run-time, mutations, and generations, and (ii) in-depth performance profiling experiments, which grants in-depth performance analysis and bottleneck identification. All experiments are performed on Chrome version 75.0.3770.100 64-bit running on a HP ZBook Studio 15 G3. This machine has 8GB RAM, an Intel Core i7-6700HQ clocking at 2.60GHz, and runs Microsoft Windows 10 Home. We now discuss each experiment set individually.

7.3.1. Performance Benchmarking

To test the overall performance of our system, we run ten optimisation runs on a predefined room (See Figure 7.2) until the genetic algorithm yields a coverage of 75 percent with sixteen cameras. Because our fitness is dependent solely on the view coverage and the number of cameras (see Algorithm 5.1), the duration of such an optimisation is indicative of how quickly our algorithm can achieve a certain fitness (i.e., the *time-to-fitness*).

Figure 7.3 shows two key moments of an experiment: achieving 75 percent coverage, and achieving the experiment goal. Our algorithm features a fitness function which causes two-phase optimisation (see Section 5.5.1). Hence, our algorithm first rapidly mutates to a configuration with high coverage, high camera count, but low camera efficiency. Subsequently, it iteratively mutates to remove cameras by optimising for camera efficiency, while maintaining high coverage.

Our experiment results are presented in Table 7.1. On average, optimisation takes just under a minute and a half, although the room may be optimised in a little as 33 seconds, or may take just under two and half minutes. Considering that optimisation needs to occur just once for per building during the installation phase of the product, an optimisation time in the order of minutes are clearly an acceptable result. Hence, this high variance is not a limiting factor.

The algorithm considers around two generations of populations per second, which is relatively high. One reason for this is the low number of individuals in our population, just 50. Indeed, when increasing the population size to 150, the generation frequency drops below 1Hz. Our low population size may also be a hint to the cause of the high run-time variance. Our algorithm hyperparameters are tuned for quick-convergence in favour of avoiding local maxima (see Section 5.5). Hence, our algorithm may get stuck in local maxima for extended periods, which causes a high run-time (e.g., in experiment 2).

7.3.2. Profiling and Performance Analysis

We now turn to the internal performance of our system. To measure component performance, our genetic algorithm includes a high-resolution hierarchical profiler. We add profiler hooks to key processing steps of

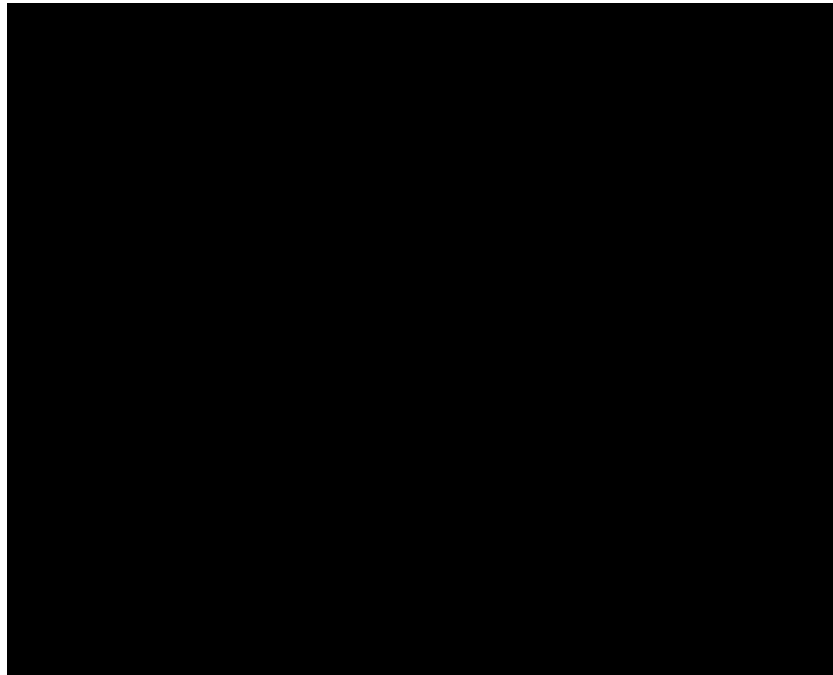


Figure 7.2: The floor plan used for view coverage optimisation experiments. The camera configuration has 75 percent coverage and 16 cameras in use. The editor pane also shows key information: the total optimisation run-time, the generation, the amount of cameras, the view coverage, and the total amount of mutations.

our algorithm (e.g., computing the outermost bounding box). Subsequently, the profiler captures the execution of these processing steps and composes a performance tree similar to a call tree. In particular, the profiler captures total execution time and number of runs, which we use to compute the average execution time of each processing step.

We run our algorithm for 600 generations on an empty instance of the floor presented in Figure 7.2. The results are presented in Table 7.2. Note that the number of executions is indicative of the variance; given that the true average run-time is distributed normally, a higher number of runs decreases the variance of the measured samples. Note also that the `sort` fragment executes only once and constitutes the initialisation phase of our algorithm.

Our results indicate that ray tracing is the primary computational bottleneck for our genetic algorithm. Throughout 600 generations the *rayTrace* fragment has been executed over twelve million times, totalling a run-time of 242 seconds of our 284-second-long optimisation run. This constitutes 85.1% of the computation of a single generation. Indeed, ray tracing is not implemented efficiently in code. For each generation and individual, the algorithm ray traces from each point to each camera. Moreover, ray tracing to a given camera consists of iterating through each wall, although the intersection operation is constant time. Formally, given c cameras, p points in the outer bounding box, i individuals, w walls, and g generations, and $n = |\text{input}| = c + p + i + w + g$, the algorithm has $O(n^5)$ run-time at minimum, solely considering ray tracing. We address how to address this issue in Section 10.

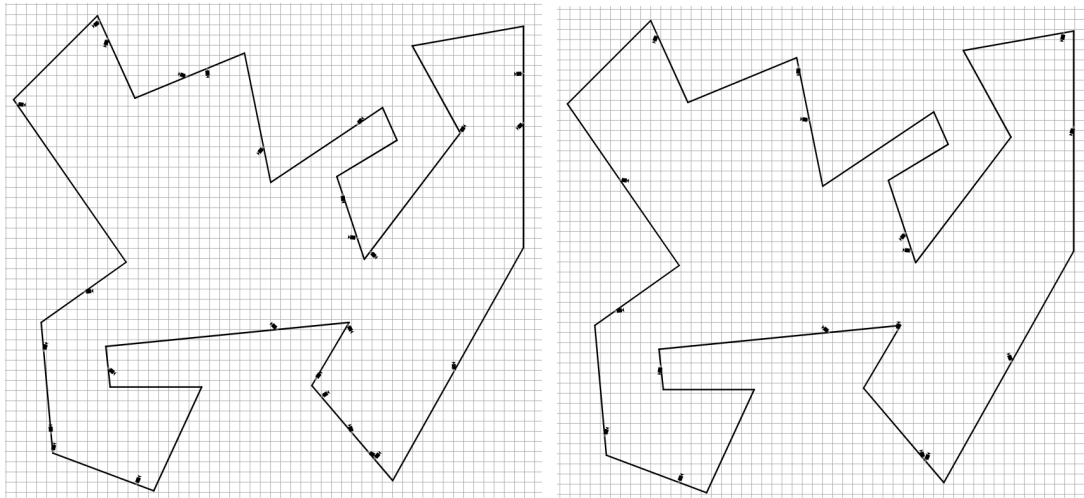


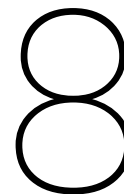
Figure 7.3: Two stages of performance optimisation: from 27 cameras to 16.

Table 7.1: Performance results for optimising coverage for a one-room floor until the genetic algorithm yielded a coverage of 75 percent with sixteen cameras. For each experiment, we capture total run-time, number of mutations, and number of generations. We compute the relevant frequencies from the latter two in combination with the run-time.

Experiment	Time [s]	Generations		Mutations	
		Total	Freq. [Hz]	Total	Freq. [Hz]
1	64.7	118	1.82	11527	178.16
2	154.6	328	2.12	32179	208.14
3	65.3	162	2.48	16035	245.56
4	49.7	117	2.35	11493	231.24
5	33.5	82	2.44	8060	240.60
6	55.7	132	2.37	12964	232.75
7	74.2	167	2.25	16375	220.68
8	112.5	254	2.26	24983	222.08
9	144.9	327	2.26	32192	222.16
10	100.7	171	1.70	17791	176.67
Average	86.7	187	2.21	18359	217.80

Table 7.2: Performance profiling data of our genetic algorithm. Each row indicates a named core processing step (e.g., a call to a function) where the name is the italicised suffix. The computation is described in a tree structure similar to a call hierarchy, although processing steps need not represent an individual function calls. Note that the sum of the processing time of each each child may not constitute the total processing time of the parent (i.e., only key processing steps are captured). Each row has zero or more percentages as a prefix, indicating the relative runtime in comparison to the column parent's run-time. For example, `rayTrace` takes up 85.2% of the run-time of `doGeneration`. The last two columns indicate the number of executions of the row's code fragment, and the average runtime of one execution.

Run-time vis-a-vis parent step [%]							Executions	Average [ms]	
T1	T2	T3	T4	T5	T6	T7			
							1	328.3	
99.6	<i>fitness</i>						50	327.0	
99.5	99.9	<i>coverage</i>					50	326.8	
0.7	0.7	0.7	<i>boundingBox</i>				50	2.1	
3.7	3.7	3.7	<i>outerPolygon</i>				50	12.2	
93.8	94.2	94.2	<i>doGrid</i>				50	308.0	
1.8	1.8	1.8	1.9	<i>generateGrid</i>			100	5.8	
13.6	13.7	13.7	14.5	<i>filter</i>			100	44.7	
78.0	78.3	78.4	83.1	<i>addToGrid</i>			100	256.1	
63.3	63.6	63.6	67.5	81.2	<i>rayTrace</i>		20500	207.9	
	<i>mainLoop</i>						600	284770.8	
99.9	<i>doGeneration</i>						600	284488.0	
0.2	0.2	<i>copy</i>					600	525.7	
0.2	0.2	<i>mutate</i>					600	683.7	
97.5	97.6	<i>sort</i>					600	277553.9	
97.4	97.5	99.9	<i>fitness</i>				29400	277372.6	
97.3	97.4	99.9	99.9	<i>coverage</i>			29400	277185.2	
0.0	0.0	0.0	0.0	0.0	<i>boundingBox</i>		29400	74.2	
0.3	0.3	0.3	0.3	0.3	<i>outerPolygon</i>		29400	965.1	
96.9	97.0	99.4	99.5	99.6	<i>doGrid</i>		29400	275945.8	
0.2	0.2	0.2	0.2	0.2	0.2	<i>generateGrid</i>	58800	662.2	
4.4	4.4	4.5	4.5	4.5	4.5	<i>filter</i>	58800	12480.4	
92.1	92.2	94.5	94.6	94.6	95.1	<i>addToGrid</i>	58800	262338.0	
85.1	85.2	87.3	87.3	87.4	87.8	92.4	<i>rayTrace</i>	12054000	242281.7
0.0	0.0	<i>boundingBox</i>					600	1.7	
0.0	0.0	<i>outerPolygon</i>					600	24.2	
2.0	2.0	<i>doGrid</i>					600	5684.6	
0.0	0.0	0.2	<i>generateGrid</i>				1200	13.5	
0.1	0.1	4.3	<i>filter</i>				1200	246.7	
1.9	1.9	95.3	<i>addToGrid</i>				1200	5415.5	
1.8	1.8	88.1	92.5	<i>rayTrace</i>			246000	5007.0	
0.1	<i>consoleMessage</i>						600	152.4	
0.0	<i>postMessage</i>						600	114.7	



Process

In this chapter, we describe various elements of our process, acknowledge challenges we came across and reflect on the whole process.

8.1. Overview

We divided the design and implementation of the system into three distinct phases. First, we dedicated two weeks of research into relevant research, the results of which can be read in Chapter 3. In the second phase, we designed and implemented our system over a period of five weeks. During these weeks, we used SCRUM methodology to guide our process, with sprint length of one week. We chose this methodology because we were already familiar with it and it allows to dynamically prioritise functionality over the duration of the development, in collaboration with the client. This was especially useful in this project, since there was a lot of functionality desired for, which we would not be able to all do. At the start of this period, a planned road map was created, which can be seen in figure 8.1. Finally, there were two weeks of integration, evaluation and finishing this report.

The team had a very clear division of labour. Since the project has two distinct parts, the group was split in two, each part of the team being responsible for a different part of the product. This does not mean the parts of the team worked separately. Problems encountered were often discussed with members from the other part as well and everyone did keep an eye at the others' progress and merge requests.

The team has several ways of communicating, for different purposes. The team organised multiple meetings each week to discuss problems and work together. Small issues encountered outside of these meetings were discussed over WhatsApp¹. There were weekly meetings with the client, at the end of the sprint. This is where progress and new priorities were discussed and questions for the client were answered. Meeting planning and some small questions were discussed over WhatsApp or phone calls. Communication with the coach mostly happened over Mattermost². Additionally, there were a few in person meetings to discuss the progress and process and to get questions answered.

8.2. Challenges

There were a few challenges we encountered during the course of the project. Firstly, at the start of the project, it became clear that the project client had a large amount of desired functionality and high, sometimes unrealistic expectations for the duration and complexity of this project. In addition to this, the client was unhappy with certain features missing. This was hard for us to do anything about, since the client was present at all sprint plannings and other functionality always had more priority. Working on those prioritised features, we did not find the time to also implement those less prioritised features.

Another challenge was the lack of available data to work with. Some of the implications of this are discussed in section 7.1. This also meant that certain learning based solutions had to be ruled out for the single fact that there is not enough data to train on.

¹<https://www.whatsapp.com/>

²<https://mattermost.com/>

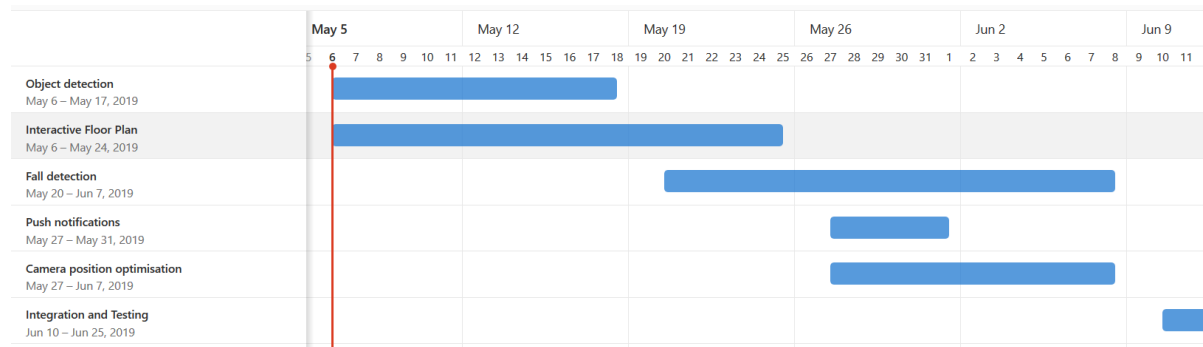


Figure 8.1: The project road map. Time runs from left to right. Each horizontal section indicates a key component in our project. Blue bars indicate when the component is under active development.

Finally, the fact that we are unfamiliar with certain factors of this development, such as GUI design or working with ■■■, resulted in that these parts needed some additional attention. Our choice of ■■■ in combination with OpenCV and ■■■ and our little experience with ■■■ resulted in us not being able to setup an automated testing framework. However this is not a large drawback since there is a manual performance test written that can be used to evaluate the system and image processing already not ideal is for unit testing.

8.3. Reflection

The focus of the project was split between the management of the floor plan and fall detection. The most logical way to deal with this was to split the team into two. As a result of this, both projects received less attention than we wished and the team felt a bit disjunct at times. Should we have had only one of the projects to focus on, we would have been able to deliver a more complete version of that project and reached more of a feeling of finishing something at the end.

In addition to this, having less time for the projects also meant we needed to focus more on the functionality and had less time for user testing. This resulted in that user testing was done in a later stage of the project and therefore we did not have enough time to implement the user feedback to improve the system.

In addition to these functionality based problems, we neglected to work on the report at appropriate times during the project. This resulted in that some sections might be less complete, since we most likely forgot some small elements that we came across during the project.

On a positive note, the team spirit was high throughout the project and collaboration came easy. Many issues were solved by discussing them with the entire team.

9

Conclusion

In this project, we were tasked with creating a program to detect when people have fallen, specifically the elderly. Elderly people are more prone to fall, and have a greater chance of injury when they do. It is key to detect these falls on time to prevent further injury or distress, as they might need immediate help. This project uses cameras to detect falls rather than worn sensors or buttons, as elderly people might forget how they work or take them off in a state of confusion. Since we are using cameras, we also need a floor plan to know where to place the cameras and in which room someone has fallen. Ideally, camera placement into the floor plan would be automatic to make it easier and faster to use, while also guaranteeing a certain coverage.

We decided to detect falls using the most common and, to us, most intuitive method that we found: [REDACTED]

[REDACTED]

[REDACTED] Using this algorithm, we can detect 33% of the falls, with a false-positive rate of 0.5 notifications per minute. This can be further improved by using machine learning to detect falls, or combining this fall detection with other sensors such as pressure-detecting floor mats.

To place cameras, we created an entire floor plan editor. This editor allows an administrator to [REDACTED]

[REDACTED]

[REDACTED] Once the general floor plan has been made, the administrator can set certain rooms where cameras should not be placed, such as bathrooms. The last step is to start the automatic camera placement. This automatic camera placement is done by a genetic algorithm. This genetic algorithm starts with an initial population where each individual has completely random placement of cameras. The individuals that give the best coverage are selected and will be the basis for the next population. Every individual of the population receives mutations such as adding, moving, or removing a camera, to try and get better coverage. This process is repeated for optimal coverage until coverage is 75%, after which the algorithm tries to achieve 75% coverage with the least amount of cameras.




10

Future Work

The solution we created in this project is a good start on solving two important aspects of the larger problem. However, both are not at their optimal state yet. As shown in chapter 7, fall detection accuracy can and should be much improved in the future, and the floor plan editor can and should be made more user friendly. Additionally, there are many Should Have and Could Have requirements (see appendix B) that could not be included. Unfortunately, due to time constraints, this will not be able to be included in this project, but many issues should receive attention in the future.

Fall detection could see big improvements by first creating or finding a more representative and larger annotated data set to train on and by applying a genetic algorithm or other automated optimisation method to tweak parameters. The current optimisation is based on human trial and error and computers sometimes find optimal solutions a human would not think of.

The floor plan could see improvements in the user friendliness of the system. The snapping could be improved to a larger distance and could be modified to let the user to set their snapping preference. As described in section 7.3 the performance of the ray tracing could be improved to improve the performance of the optimal camera placement. This could be done by using memoization and by only ray tracing to points in the same room as cameras. Some additional features could be implemented such as



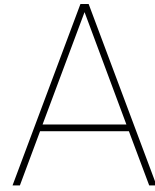
The larger system could be improved at several points. More work should be put into making facial recognition work in this setting, so fall notifications can be person specific and more false positives can be eliminated because they are from non human objects. This would also allow for tracking residents and staff in the building and show people's locations on the floor plan. The system could combine the results of this camera based fall detection with fall detection mats.

Bibliography

- [1] T. Al-Aama. "Falls in the elderly". In: *Canadian Family Physician* 57.7 (2011), pp. 771–776.
- [2] S. Ahmed et al. "Automatic room detection and room labeling from architectural floor plans". In: *Proceedings - 10th IAPR International Workshop on Document Analysis Systems, DAS 2012*. IEEE, 2012, pp. 339–343.
- [3] A. Aissaoui et al. "Designing a camera placement assistance system for human motion capture based on a guided genetic algorithm". In: *Virtual Reality* 22.1 (2018), pp. 13–23.
- [4] A. H. Albahri and A. Hammad. "Simulation-Based Optimization of Surveillance Camera Types, Number, and Placement in Buildings Using BIM". In: *Journal of Computing in Civil Engineering* 31.6 (2017), p. 04017055.
- [5] D. Anderson et al. "Linguistic summarization of video for fall detection using voxel person and fuzzy logic". In: *Computer Vision and Image Understanding* 113.1 (2009), pp. 80–89.
- [6] D. Anderson et al. "Recognizing Falls from Silhouettes". In: *2006 International Conference of the IEEE Engineering in Medicine and Biology Society*. 2006, pp. 6388–6391.
- [7] E. Auvinet et al. "Fall Detection With Multiple Cameras: An Occlusion-Resistant Method Based on 3-D Silhouette Vertical Distribution". In: *IEEE Transactions on Information Technology in Biomedicine* 15.2 (2011), pp. 290–300.
- [8] M. Belshaw et al. "Towards a Single Sensor Passive Solution for Automated Fall Detection". In: *Conference proceedings : ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference 2011* (2011), pp. 1773–6.
- [9] M. B. Blaschko and C. H. Lampert. "Learning to Localize Objects with Structured Output Regression". en. In: *Computer Vision – ECCV 2008*. Ed. by D. Forsyth, P. Torr, and A. Zisserman. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, pp. 2–15.
- [10] L. Bourdev and J. Brandt. "Robust object detection via soft cascade". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 2. 2005, 236–243 vol. 2.
- [11] I. Charfi et al. "Optimized spatio-temporal descriptors for real-time fall detection: comparison of support vector machine and Adaboost-based classification". en. In: *Journal of Electronic Imaging* 22.4 (2013), p. 17.
- [12] D. Chrysostomou and A. Gasteratos. "Optimum multi-camera arrangement using a bee colony algorithm". In: *IST 2012 - 2012 IEEE International Conference on Imaging Systems and Techniques, Proceedings*. IEEE, 2012, pp. 387–392.
- [13] N. Dalal and B. Triggs. "Histograms of Oriented Gradients for Human Detection". en. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. San Diego, CA, USA: IEEE, 2005, pp. 886–893.
- [14] G. Diraco, A. Leone, and P. Siciliano. "An active vision system for fall detection and posture recognition in elderly healthcare". In: *Proceedings of the Conference on Design, Automation and Test in Europe*. EDAA, 2013, pp. 1536–1541.
- [15] P. Dollar, S. Belongie, and P. Perona. "The Fastest Pedestrian Detector in the West". en. In: *Proceedings of the British Machine Vision Conference 2010*. Aberystwyth: British Machine Vision Association, 2010, pp. 68.1–68.11.
- [16] P. Dollar et al. "Pedestrian Detection: A Benchmark". en. In: (), p. 8.
- [17] P. F. Felzenszwalb et al. "Object Detection with Discriminatively Trained Part Based Models". en. In: (), p. 20.
- [18] H. Foroughi, B. S. Aski, and H. Pourreza. "Intelligent video surveillance for monitoring fall detection of elderly in home environments". In: *2008 11th International Conference on Computer and Information Technology*. 2008, pp. 219–224.
- [19] J. Gall, N. Razavi, and L. Van Gool. "An Introduction to Random Forests for Multi-class Object Detection". en. In: *Outdoor and Large-Scale Real-World Scene Analysis*. Ed. by F. Dellaert et al. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 243–263.
- [20] J. Gall et al. "Hough Forests for Object Detection, Tracking, and Action Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.11 (2011), pp. 2188–2202.
- [21] C. Gao et al. "An efficient local search heuristic with row weighting for the unicost set covering problem". en. In: *European Journal of Operational Research* 246.3 (2015), pp. 750–761.

- [22] S. Hinterstoisser et al. "Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes". In: *2011 International Conference on Computer Vision*. 2011, pp. 858–865.
- [23] E. Hsiao and M. Hebert. "Occlusion reasoning for object detection under arbitrary viewpoint". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 2012, pp. 3146–3153.
- [24] C. Juang and C. Chang. "Human Body Posture Classification by a Neural Fuzzy Network and Home Care System Application". In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 37.6 (2007), pp. 984–994.
- [25] S. M. Khan and M. Shah. "Tracking Multiple Occluding People by Localizing on Multiple Scene Planes". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31.3 (2009), pp. 505–519.
- [26] S. S. Khan and M. G. Madden. "One-Class Classification: Taxonomy of Study and Review of Techniques". In: *The Knowledge Engineering Review* 29.3 (2014). arXiv: 1312.0049, pp. 345–374.
- [27] K. Kim et al. "Real-time Foreground-background Segmentation Using Codebook Model". In: *Real-Time Imaging* 11.3 (2005), pp. 172–185.
- [28] J. Kritter et al. "On the optimal placement of cameras for surveillance and the underlying set cover problem". In: *Applied Soft Computing Journal* 74 (2019), pp. 133–153.
- [29] T. Lee and A. Mihailidis. "An Intelligent Emergency Response System: Preliminary Development and Testing of Automated Fall Detection". In: *Journal of telemedicine and telecare* 11 (2005), pp. 194–8.
- [30] B. Leibe, A. Leonardis, and B. Schiele. "An Implicit Shape Model for Combined Object Categorization and Segmentation". en. In: *Toward Category-Level Object Recognition*. Ed. by J. Ponce et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 508–524.
- [31] F. Lin et al. "Super-Resolved Faces for Improved Face Recognition from Surveillance Video". en. In: *Advances in Biometrics*. Ed. by S.-W. Lee and S. Z. Li. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, pp. 1–10.
- [32] C.-L. Liu, C.-H. Lee, and P.-M. Lin. "A fall detection system using k-nearest neighbor classifier". In: *Expert Systems with Applications* 37.10 (2010), pp. 7174–7181.
- [33] H. Liu and C. Zuo. "An Improved Algorithm of Automatic Fall Detection". In: *AASRI Procedia*. AASRI Conference on Computational Intelligence and Bioinformatics 1 (2012), pp. 353–358.
- [34] H. Lu et al. "Intelligent Human Fall Detection for Home Surveillance". In: *2014 IEEE 11th Intl Conf on Ubiquitous Intelligence and Computing and 2014 IEEE 11th Intl Conf on Autonomic and Trusted Computing and 2014 IEEE 14th Intl Conf on Scalable Computing and Communications and Its Associated Workshops*. 2014, pp. 672–676.
- [35] S. Macé et al. "A system to detect rooms in architectural floor plan images". In: *Proceedings of the 8th IAPR International Workshop on Document Analysis Systems - DAS '10*. New York, New York, USA: ACM Press, 2010, pp. 167–174.
- [36] S. Manen, M. Guillaumin, and L. Van Gool. "Prime Object Proposals with Randomized Prim's Algorithm". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2013, pp. 2536–2543.
- [37] M. Mubashir, L. Shao, and L. Seed. "A survey on fall detection: Principles and approaches". In: *Neurocomputing*. Special issue: Behaviours in video 100 (2013), pp. 144–152.
- [38] N. Noury et al. "Fall detection - Principles and Methods". In: *2007 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. 2007, pp. 1663–1666.
- [39] R. Okada. "Discriminative generalized hough transform for object detection". In: *Proceedings of the IEEE International Conference on Computer Vision*. Kyoto, Japan: IEEE, 2009, pp. 2000–2005.
- [40] D. N. Olivieri, I. Gómez Conde, and X. A. Vila Sobrino. "Eigenspace-based fall detection and activity recognition from motion templates and machine learning". In: *Expert Systems with Applications* 39.5 (2012), pp. 5935–5945.
- [41] F. Perronnin, J. Sánchez, and Y. L. Xerox. "Large-scale image categorization with explicit data embedding". In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2010, pp. 2297–2304.
- [42] I. T. Phillips and A. K. Chhabra. "Empirical performance evaluation of graphics recognition systems". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 21.9 (1999), pp. 849–870.
- [43] C. Rougier et al. "Fall Detection from Human Shape and Motion History Using Video Surveillance". In: *21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07)*. Vol. 2. 2007, pp. 875–880.
- [44] C. Rougier et al. "Monocular 3D Head Tracking to Detect Falls of Elderly People". In: *2006 International Conference of the IEEE Engineering in Medicine and Biology Society*. 2006, pp. 6384–6387.
- [45] K. Sato et al. "CAD-based object tracking with distributed monocular camera for security monitoring". In: *Proceedings of 1994 IEEE 2nd CAD-Based Vision Workshop*. 1994, pp. 291–297.

- [46] M. Shoaib, R. Dragon, and J. Ostermann. "View-invariant Fall Detection for Elderly in Real Home Environment". In: *2010 Fourth Pacific-Rim Symposium on Image and Video Technology*. 2010, pp. 52–57.
- [47] C. Steger. "Similarity Measures for Occlusion, Clutter, and Illumination Invariant Object Recognition". en. In: *Pattern Recognition*. Ed. by B. Radig and S. Florczyk. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2001, pp. 148–154.
- [48] S. Sternig et al. "Multi-camera multi-object tracking by robust hough-based homography projections". In: *Proceedings of the IEEE International Conference on Computer Vision*. IEEE, 2011, pp. 1689–1696.
- [49] S. Suzuki and K. be. "Topological structural analysis of digitized binary images by border following". In: *Computer Vision, Graphics, and Image Processing* 30.1 (1985), pp. 32–46.
- [50] D. Tang, Y. Liu, and T.-k. Kim. "Fast Pedestrian Detection by Cascaded Random Forest with Dominant Orientation Templates". en. In: *Proceedings of the British Machine Vision Conference 2012*. Surrey: British Machine Vision Association, 2012, pp. 58.1–58.11.
- [51] D. Tax. "One-class classification". PhD thesis. Delft University of Technology, 2001.
- [52] A. Tejani et al. "Latent-class Hough forests for 3D object detection and pose estimation". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8694 LNCS.PART 6 (2014), pp. 462–477.
- [53] N. Thome, S. Miguet, and S. Ambellouis. "A Real-Time, Multiview Fall Detection System: A LHMM-Based Approach". In: *IEEE Transactions on Circuits and Systems for Video Technology* 18.11 (2008), pp. 1522–1532.
- [54] M. Trajkovic. "Optimal multi-camera setup for computer-based visual surveillance". EP1433326A1. 2004.
- [55] P. Vallabh et al. "Fall detection using machine learning algorithms". In: *2016 24th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. 2016, pp. 1–9.
- [56] P. Viola and M. Jones. "Rapid object detection using a boosted cascade of simple features". en. In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. Vol. 1. Kauai, HI, USA: IEEE Comput. Soc, 2001, pp. I–511–I–518.
- [57] F. W. Wheeler, R. L. Weiss, and P. H. Tu. "Face recognition at a distance system for surveillance applications". In: *2010 Fourth IEEE International Conference on Biometrics: Theory, Applications and Systems (BTAS)*. 2010, pp. 1–8.
- [58] D. Whitley. "A genetic algorithm tutorial". en. In: *Statistics and Computing* 4.2 (1994), pp. 65–85.
- [59] T. Xiang et al. *Random Forest with Adaptive Local Template for Pedestrian Detection*. en. Research article. 2015.
- [60] G. Yao et al. "Comparative Evaluation of Background Subtraction Algorithms in Remote Scene Videos Captured by MWIR Sensors". In: *Sensors (Basel, Switzerland)* 17.9 (2017).
- [61] M. Yu, S. M. Naqvi, and J. Chambers. "A robust fall detection system for the elderly in a smart room". In: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*. IEEE, 2010, pp. 1666–1669.
- [62] M. Yu et al. "A posture recognition-based fall detection system for monitoring an elderly person in a smart home environment". In: *IEEE Transactions on Information Technology in Biomedicine* 16.6 (2012), pp. 1274–1286.
- [63] X. Yu and Xinguo Yu. "Approaches and principles of fall detection for elderly and patient". In: *2008 10th IEEE Intl. Conf. on e-Health Networking, Applications and Service, HEALTHCOM 2008*. IEEE, 2008, pp. 42–47.
- [64] X. Yu et al. "Fall detection and alert for ageing-at-home of elderly". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 5597 LNCS. Springer, Berlin, Heidelberg, 2009, pp. 209–216.
- [65] Y.-Y. Zheng and J. Yao. "Multi-angle face detection based on DP-Adaboost". en. In: *International Journal of Automation and Computing* 12.4 (2015), pp. 421–431.



Project Description

This chapter lists the verbatim project description as presented on BEPSys.

Context

Do you want to be part of an innovation project and have a direct and tangible positive impact on the society?

In collaboration with a startup in the medical field and as a small sub-project, we would like to have a HTML 5 front-end (GUI) for our existing system to administrate the system users.

About Us

Do you want to be part of an innovation project and have a direct and tangible positive impact on the society?

In collaboration with a startup in the medical field and as a small sub-project, we would like to have a HTML 5 front-end (GUI) for our existing system to administrate the system users.

Assignment

In general, elderly people don't like or refuse to carry any wearables on their body. However, for indoor tracking or fall detection, they have to wear sensors and beacons.

In this project, you are being challenged to track and detect fall incidents of elderly in their home or a care center without using any wearables. You will research and create a system that tracks elderly and detects possible fall incident by means of an ONVIF supported camera in a dynamic light environment.

An incident alert and indoor tracking position have to be sent to the caretakers using our current frontend, middleware and backend (PHP). You may upgrade our push notification services to send the alerts to the caretakers even the frontend is running on the background.

For tracking purposes, the frontend uses the floor or a building map as an input to interact and show the position of the individuals, cameras or incidents. The users should be able to track and localize their friends in the building and send messages to each other to make meet up.

The administrator should be able to add, remove or move camera objects on the map to have a better field of view coverage. The cameras have to recognize people and trace them on the map in real-time and send an alarm in case an individual is prohibited to enter a certain room (in case a room has a NFC lock system, the door should locked or closed).

Beside the administrator, an automated camera positioning algorithm should be able to estimate the best position of the cameras in the rooms and corridors for the best coverage and create a coverage map.

You will provide analytics about the individuals' and groups' locations, activities, incidents, amount of active time extracted based on historical data.

At the end, your work must be migrated into our system, which means you may use and upgrade existing components to accomplish your assignment. Documentation of the code, deployment and testing of the system are part of this assignment!

Nice to have: Investigate the concurrent connections and the response time of the push notifications and the system with respect to server's CPU load and the number of active connection with the database.

B

Requirements Analysis

B.1. Functional Requirements

The functional requirements, prioritised with MoSCoW, are as follows:

Must Have

1. [REDACTED]
2. [REDACTED]
3. [REDACTED]
4. [REDACTED]
5. [REDACTED]
6. [REDACTED]
7. [REDACTED]

Should Have

1. The application should show a video feed of clients' locations to users with the 'caretaker' and 'user' roles.
2. 'Caretaker' users should receive a push notification through the front end when their clients have had an incident.
3. When the application detects that a client has fallen, video streaming should be turned off.

Could Have

1. The application could show users profiles of nearby people with whom they are connected.
2. The application could automatically create a floor plan model from a JPEG scan of a floor plan. This functionality would automatically detect windows (cannot place cameras) and doors.
3. The application could keep track of the movement and direction of humans and persistently attach profile data to an avatar moving between cameras.
4. The application could detect when unauthorised people are present in a room and subsequently send caretakers a push notification.

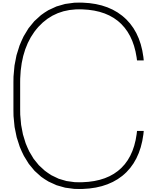
Would Have

1. The application would open and close electronic doors for users based on their credentials and their recognised faces.
2. The application would blur clients on video that have not given permission to be visible on video.

B.2. Non-functional Requirements

The non-functional requirements consist of:

1. The project must integrate well with the existing platform.
2. The application must be well-documented. Non-trivial routines must have applicable documentation.
3. The application must be well-tested. A minimum of 60% code coverage is required with exception of image processing components.
4. The project must be twice submitted to SIG for software quality review.



Feedback from SIG

This appendix lists the verbatim feedback received from SIG after submission of our code in the week six and week nine of our project. We address the feedback in Section 5.6.

C.1. Feedback week six

De code van het systeem scoort 4.0 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code marktgemiddeld onderhoudbaar is. We zien Unit Interfacing en Unit Complexity vanwege de lagere deelscores als mogelijke verbeterpunten.

Voor Unit Interfacing wordt er gekeken naar het percentage code in units met een bovengemiddeld aantal parameters. Doorgaans duidt een bovengemiddeld aantal parameters op een gebrek aan abstractie. Daarnaast leidt een groot aantal parameters nogal eens tot verwarring in het aanroepen van de methode en in de meeste gevallen ook tot langere en complexere methoden. Dit kan worden opgelost door parameter-objecten te introduceren, waarbij een aantal logischerwijs bij elkaar horende parameters in een nieuw object wordt ondergebracht. Dit geldt ook voor constructors met een groot aantal parameters, dit kan een reden zijn om de datastructuur op te splitsen in een aantal datastructuren. Als een constructor bijvoorbeeld acht parameters heeft die logischerwijs in twee groepen van vier parameters bestaan, is het logisch om twee nieuwe objecten te introduceren.

Voorbeelden in jullie project:

- `utils.js:roundRect`
- `walls.js:WallModule.getSnappedPoint`

Voor Unit Complexity wordt er gekeken naar het percentage code dat bovengemiddeld complex is. Dit betekent overigens niet noodzakelijkerwijs dat de functionaliteit zelf complex is: vaak ontstaat dit soort complexiteit per ongeluk omdat de methode te veel verantwoordelijkheden bevat, of doordat de implementatie van de logica onnodig complex is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, makkelijker te testen is, en daardoor eenvoudiger te onderhouden wordt. Door elk van de functionaliteiten onder te brengen in een aparte methode met een beschrijvende naam kan elk van de onderdelen apart getest worden, en wordt de overall flow van de methode makkelijker te begrijpen. Bij grote en complexe methodes kan dit gedaan worden door het probleem dat in de methode wordt opgelost in deelproblemen te splitsen, en elk deelprobleem in een eigen methode onder te brengen. De oorspronkelijke methode kan vervolgens deze nieuwe methodes aanroepen, en de uitkomsten combineren tot het uiteindelijke resultaat.

Voorbeelden in jullie project:

- `floors.js:FloorSelector.onClick`
- `model.js:Floor.addWall`

Als laatste nog de opmerking dat er geen (unit)test-code is gevonden in de code-upload. Het is sterk aan te raden om in ieder geval voor de belangrijkste delen van de functionaliteit automatische

tests gedefinieerd te hebben om ervoor te zorgen dat eventuele aanpassingen niet voor ongewenst gedrag zorgen. Op lange termijn maakt de aanwezigheid van unit tests je code ook flexibeler, omdat aanpassingen kunnen worden doorgevoerd zonder de stabiliteit in gevaar te brengen.

Over het algemeen is er dus nog wat verbetering mogelijk, hopelijk lukt het om dit tijdens de rest van de ontwikkelfase te realiseren.

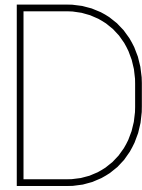
C.2. Feedback week nine

In de tweede upload zien we dat het codevolume is gegroeid, terwijl de score voor onderhoudbaarheid is gestegen.

We zien dat de verbeterpunten uit de feedback op de eerste upload zijn aangepast, en op deze gebieden is dan ook een verbetering in de deelscores te zien. Dat is wel voornamelijk zo bij Unit Interfacing, waar jullie echt een duidelijke sprong hebben gemaakt, en iets minder bij Unit Complexity.

ook is het goed om te zien dat er naast nieuwe productiecode ook nieuwe testcode is geschreven.

Uit deze observaties kunnen we concluderen dat de aanbevelingen van de vorige evaluatie zijn meegenomen in het ontwikkeltraject.



Info sheet

Camera Positioning and Vision-based Fall Detection

Commissioned by Eya Solutions.

To be presented July 3, 2019.

D.1. Description

The setting of this project is care facilities where elderly are treated. This project aims to find a solution for detecting when people fall. Detecting falls is an issue that has seen much research already and many different approaches have been employed. This project aims to create a non-intrusive, non-wearable solution through the use of cameras. It extends an existing patient monitoring system developed by Eya Solutions, a start-up in the medical field. Their system monitors patients and is composed of embedded systems (*clients*), a back-end with a database, and a web-based dashboard (the *front-end*) for administrators, caretakers and users.

One part of the project is detecting fall incidents in video footage. Relevant related work showed that first using background subtraction and then fitting ellipses around objects is a common method for tracking pedestrians. Because this method is also intuitive, we employ this method in our approach. By means of image processing, our system detects objects in the image and subsequently tracks these between frames. Based on the properties of such objects, we detect fall incidents. This detection has an accuracy of 33% and a false positive rate of 0.5 false notifications per minute.

A second part of the project is the floor plan editor. This editor is integrated into the existing dashboard, where administrators can model the building. Eya Solutions wishes to provide the product to customers as a complete package, including (i) showing where fall incidents have occurred, and (ii) how and where to place cameras. Our system allows administrators to edit the floor while allowing caretakers to see the modelled floor and see where falls are detected. Our research showed that it was unfeasible to create the floor plan automatically. Considering (ii), it is desirable to generate a configuration where the number of cameras is low while the view coverage is high. Our contribution includes a genetic algorithm which can generate automatically a suitable configuration. Alternatively, cameras can be placed manually by the administrator.

All main aspects of the system have been tested. The floor plan editor is tested by means of user tests. The camera placement optimisation was tested on speed and bottlenecks. The fall detection was tested on accuracy. In order to complete this project, we employed a loose variant of the SCRUM development methodology, with sprint lengths of one week. Challenges in the process of this project were the high amount of requested functionality. To resolve this, we cut in what functionality we were going to implement.

This project needs more development to improve performance and user friendliness before it is taken into production. The aim is to take the overall system into production in the future.

D.2. Contributors

Jerom van der Sar

Interests: Distributed systems, artificial intelligence, genetic algorithms, embedded systems.

Contributions: GUI design, floor plan editor tools, coverage calculator, genetic algorithm, hyperparameter tuning.

Arjan Seijs

Interests: Artificial intelligence, genetic algorithms, machine Learning.

Contributions: Floor plan editor, user manual, dashboard tests.

Ivo Nelissen

Interests: Computer vision, computer graphics, algorithms, embedded software.

Contributions: Background subtraction, project setup (██████, OpenCV), startup service, camera management and streaming in the front end.

Robin Cromjongh

Interests: Artificial intelligence, algorithms, human-computer interaction.

Contribution: Morphology, object tracking, fall detection, fall detection accuracy testing.

All of the team members contributed to the report.

D.3. Contact

Client: Kianoush Rassels MSc, Eya Solutions

Coach: Dr. Cynthia C. S. Liem, Intelligent Systems: Multimedia Computing Group

Contact: Kianoush Rassels — k.rassels@tudelft.nl

A digital version of this thesis can be found at <http://repository.tudelft.nl>.

