CHALMERS
UNIVERSITY OF TECHNOLOGY

Human activity analysis and classification using RGB-D videos

Analys och klassificering av mänskliga aktiviteter via RGB-D videos

Bachelor's thesis in Automation and Mechatronics

CHRISTOPHER INNOCENTI

GUSTAV NERO

HENRIK LINDÉN

Department of Signals and Systems
Chalmers University of Technology
Gothenburg, Sweden 2015

Bachelor's thesis SSYX02-15-01

Human activity analysis and classification using RGB-D videos

Analys och klassificering av mänskliga aktiviteter via RGB-D videos

CHRISTOPHER INNOCENTI
GUSTAV NERO
HENRIK LINDÉN

Department of Signals and Systems
Division of Signal Processing
Chalmers University of Technology
Gothenburg, Sweden 2015

Human activity analysis and classification using RGB-D videos
Analys och klassificering av mänskliga aktiviteter via RGB-D videos

Supervisor: Yixiao Yun, Department of Signals and Systems
Examiner: Irene Yu-Hua Gu, Department of Signals and Systems

Bachelor's Thesis SSYX02-15-01
Department of Signals and Systems
Division of Signal Processing
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2015

Preface

At Chalmers University of Technology, after the first two and a half years of study, it is time to implement some of the gathered knowledge. This is done in the form of a project which span over the final semester of the bachelor (undergraduate) level. The project comprises 15 hp which is the equivalent to 8 weeks of full time work. Due to the versatility of the program Automation and Mechatronics, different types of project subjects were available. Our group chose to investigate the project regarding “Human activity analysis and classification using RGB-D videos”, which closely relates to the Automation part of the program. The field of creating machines that can take sophisticated action in different kinds of situations is just in its infancy but is rapidly growing. It was with great interest that we embarked upon this journey to get a glimpse of what it takes for a computer to ”see as humans do”. Our hope is that the reader will have as much joy reading this thesis as we had writing it.

The Authors

Abstract

This thesis encompasses parts of the field of computer vision. The main problem dealt with throughout this project is how to automate classification of specific human activities via video streams. The essence of this project is therefore that the developed algorithm shall be able to distinguish these activities from one another. A number of restrictions were imposed on the data-set in order to keep the problem size manageable. A video snippet was assumed to contain a single human that either falls to the ground or, in a controlled way, sits and then lies down on the ground.

Our approach relies heavily on background subtraction as the method for detecting foreground objects and comparing the performance of several such approaches for video pre-processing. Features are designed with a heuristic approach utilizing velocities, limb distance relationships and acknowledged techniques based on oriented gradients and optical flow. The classification model is a pre-implemented support vector machine from the libSVM package that is tuned to fit the generated features.

The accuracy of our algorithm reached 93.73% which was well above the goal of 85%. On average the classification process roughly took 0.6 second per frame which concludes that the run-time of our algorithm was not fast enough to process a video stream in real time. The result of this work confirms, however, that the methods used works well in a restricted setting and produces a high classification rate. The gist is thus that the proposed method is a good classifier when used in a restricted, offline environment.

Sammanfattning

Den här tesen behandlar en gren av datorseende. Huvudproblemet som hanteras i detta projekt är hur klassificering av mänskliga aktiviteter från videoströmmar kan automatiseras. Projektet handlar väsentligen om att den framtagna algoritmen ska kunna skilja på två aktiviteter. Ett antal restriktioner tillämpades för att kunna hålla problemet hanterbart. Ett videoklipp antogs innehålla en människa som antingen faller eller kontrollerat sätter sig och sedan lägger sig ned. Vårt tillvägagångsätt grundar sig i bakgrundssubtraktion som metod för att detektera objekt i förgrunden och vi jämför ett antal sådana videoprocessering-metoders prestanda. Attribut (säregenskaper) är designade utifrån hastighet, distans mellan kroppsdelar och erkända tekniker såsom optiskt flöde och orienterade gradienter. Klassificeringsmodellen är en förimplementerad stödvektormaskin från libSVM-biblioteket som justerats för att passa projektets behov. Pålitligheten för algortimen nådde 93.73% vilket var väl över det initiella målet på 85%. Algoritmen var dock inte tillräckligt snabb för att bearbeta videoströmmar i realtid då den behövde cirka en halv sekund per bild. Resultatet av detta arbete kan bekräfta att den metod som presenteras är en välfungerande lösning på problemet i en begränsad, offline miljö.

Acknowledgements

We would like to express our gratitude and thank our supervisor Yixiao Yun for putting up with our questions during this project and for helping us getting a better understanding on the subject of computer vision.

The Authors

Contents

Nomenclature

Throughout this thesis we will make use of some acronyms. For simplicity these will be stated here for future reference.

SVM Support Vector Machine

HOG Histogram of Oriented Gradients

HOF Histogram of Optical Flow

VGA Video Graphics Array

EB EigenBackground

GMM Gaussian Mixture Model

RGB-D Red Green Blue – Depth

HSV Hue Saturation Value

BS Background Subtraction

ROI Region Of Interest

SE Structuring Element

PCA Principal Component Analysis

SVD Singular Value Decomposition

EMT Extended Maxima Transform

BB Bounding Box

CM Confusion Matrix

TPR True Positive Rate

TNR True Negative Rate

FPR False Positive Rate

FNR False Negative Rate

CCR Correct Classification Rate

MCR Miss-Classification Rate

CV Cross-Validation

RBF Radial Basis Function

Chapter 1

Introduction

The human mind is extremely capable of recognizing visual input, especially if the input consists of what other humans are doing. To distinguish whether a person sits down or falls down is a quite trivial task for the human mind. If a computer is given the same input however, things become considerably more complex. For example, the depth of an object has to be taken into account, as well as the time for a particular action to occur. These are things that our minds does on their own, without us needing to think about it. A computer, however, will not be able to make use of the depth, or anything at all for that matter, without proper teaching (i.e. programming). To create an effective program or algorithm that enables the computer to see as humans do has proven to be a difficult task. Perhaps this difficulty stems from the fact that we do not yet fully know how our own brains interpret the visual stimuli from the eye. Therefore, trying to create a program that does this becomes quite the challenge, because after all, how do you teach something of which you know nothing yourself?

1.1 Background

Despite the above mentioned obstacles much progress has been made and the field of computer vision is rapidly expanding with new applications found frequently. For example, the entertainment industry makes use of the technology by incorporating it into video games such as the Kinect products, where the player is able to "step into the game" so to speak [1]. Another example of computer vision in entertainment are cameras with the ability to autofocus on a person's face as well software that can perform a so called "faceswap". However, computer vision is not restricted to the entertainment business only. When used in conjunction with surveillance some new interesting areas emerge. In traffic safety, there is a prominent example of computer vision, namely the Volvo city safety system which assists the driver by detecting potential threats [2]. Computer vision could also be of use in the care for the elderly by allowing them to stay in the comfort and familiarity of their own homes but with an automated system that watches over them. If, for example, a person falls over, the system could be able to decide whether to send someone over for help (perhaps if the person has stopped moving after the fall or something similar). A system of this type also needs to be able to accurately distinguish between a person sitting down on purpose and a person falling down to make sure that no false alarms are triggered.

1.2 Purpose and Goal

The purposes of this project are to obtain knowledge in the field of computer vision and maybe broaden the general knowledge base in this relatively young area. The goal is to generate a software program in Matlab. The piece of software should manage to detect and classify specific human activities from a Red-Green-Blue-Depth (RGB-D) video. The activities that this software program should be able to support are falling and sitting/lying down. Furthermore, it should be re-trainable so that other activities from new videos can be classified.

1.3 Problem definition

The main problem dealt with throughout this project is how to automate classification of specific human activities via video streams. The data used in this project are RGB-D videos captured by a Kinect camera[1]. Intuitively, actions being very similar to each other would be the most difficult ones to distinguish and tell apart. Success in developing an algorithm that is able to accurately classify similar actions would imply that the same algorithm, after proper training, most likely would classify diverse activities even better. The aim of this project therefore focuses on classification of two similar activities. The first activity is a person that starts in an upright pose followed by a fall to the ground, this activity will be referred to as activity 1 throughout the rest of this report. The second activity is a sequence of actions where a person starts from an upright pose followed by sitting down then followed again by lying down, and will similarly be referred to as activity 2. The process of classifying one activity alone is quite complex and thus separating this process into sub-tasks will give an overview that is easier to comprehend. This process can be divided into three sub-tasks, *detection*, *feature extraction* and *classification*. Each of these gives rise to various problems;

Looking at the first task, detection, which in itself could contain all three tasks mentioned above. The first issue at hand is the extraction of manageable data from a video stream, i.e. extracting images, frame by frame, upon which further analysis can be performed. Once the raw image data can be accessed, the location of the object of interest (in this case, the person) should be found. In order to locate an object, features that makes it possible to determine whether the image contain an object or not should be extracted. Once these features has been extracted, some classification process should be applied for detection. If an object is detected, the region of interest[2] (ROI) should be masked out in order to reduce the effect of different backgrounds/environments.

[1]These videos contain a coloured (red) skeleton that's projected onto the person captured on the video. The skeleton is connected by the persons joints i.e. head, shoulder, elbow etc.

[2]A region of interest is a small region of the image that encloses the object of interest.

The second step, the task of feature extraction (features corresponding to activities), should be done by use of the ROI. These features must be characteristic of the individual activities so that it is possible to distinguish them. To get a sense of motion additional features should be extracted, features that are non-static in the sense that they are extracted from a sequence of images.

Finally the classification task, initially the data-set needs to be separated into training and testing data. The training data is the data used to develop a classifier. This data should be diverse so that it is not biased towards classifying activities performed by persons of a certain height, gender or look. Further aspects, e.g. the trajectory and distance of the fall should also follow this criteria. Since the set of activities is limited to the size of two, a binary classifier should be used to decide whether a performed activity belongs to one class or the other. Such a classifier that has shown good performance in earlier work is the support vector machine (SVM). Because of this, the (SVM) will be used as the main classifier in the project.

To ensure the robustness and accuracy of the classifier some limits by measures of performance must be reached. The classification rate i.e. the number of classifications of activity 1 and 2 that maps them to their true class should be approximately 85-90%. The false alarm or false positive rate (FPR) i.e. the number of classifications that maps anything that is not activity 1 to class 1 should be less than 10%. Other aspects such as the speed and execution time of the algorithm should be, without any formal limit, fast enough so that it is considered useful. From the descriptions above a summary of the problems can be derived:

- How to extract manageable data from a video stream?
- How to locate the object of interest in an image?
- How to extract the ROI from the image? What shape?
- What features should be used to enable accurate classification?
- How should the distribution of training vs. testing data be selected?
- How should the training of the classifier be performed?
- How to test and verify the accuracy and robustness of the classifier?
- How to test and verify the speed and execution time?

1.4 Limitations

Since the area of computer vision is quite broad, with a plethora different approaches available to solve a problem, some restrictions have to be enforced on the project in order to keep it manageable in the short time frame available for the bachelor thesis. There exists such a large amount of possible algorithms to use that covering them all is too overwhelming. Therefore, in order to keep the problem simple enough to tackle with no prior experience in the area of image processing and computer vision, as well as with respect to the time restriction, the following limitations will be made.

- There will be only one human to classify in every video-stream. (In each frame there will only be one person) In addition, the only moving object will be a human.
- There will only be two classes of activities to handle. In this case; falling and sitting down.
- The data to be used have been captured with a fixed Kinect camera and can have a skeleton overlayed on top of any humans present in the video.
- For every video there will be possible to access at least one ”empty” frame with only the background and no objects of interest visible
- The algorithm will be coded in Matlab and will make use of the available toolboxes such as the Computer Vision System and Image Processing toolboxes in order to achieve its results. (For example by using the vision.blobAnalyzer object.)
- The classifier (for activity 1 and 2) to be used will be a SVM that we will train by using some of our data.
- There will be no formal performance demands. The algorithm will not have to run within a certain time frame.

1.5 Outline

Chapter 2 – Theory Introduces general concepts and methods used throughout this thesis with regard to detection (pre-processing), feature generation/extraction and classification techniques.

Chapter 3 – Method Outlines the methodologies used in each phase of this project and gives implementation details of the corresponding methods respectively.

Chapter 4 – Experimental Results Gives an overview of the results obtained by different approaches used for solving the detection problem, the design of features as well as classification rates.

Chapter 5 – Discussion Provides a comprehensive discussion where advantages and disadvantages for each part of this project is more thoroughly evaluated.

Chapter 6 – Conclusions Gives a concluding summary of our thoughts on the obtained results and how the initial problems were solved.

Chapter 2

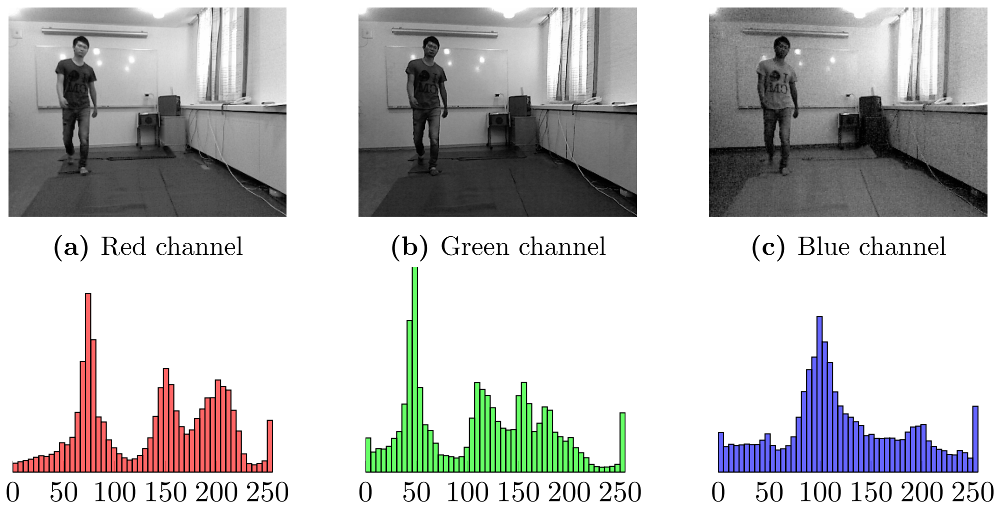
Theory

The aim of this chapter is to present the basic theory on the subject of image processing. It will also outline descriptions of the methods and techniques used throughout this thesis. Initially the general concept of RGB-D (Red,Green,Blue,-Depth) images and some basic operations used for processing is explained. A description of methods usable for object detection is presented followed by several sections introducing methods used for extraction of image features. Finally the concept of classification using support vector machines (SVM:s) is described.

2.1 RGB-D images

Digital images are built up by pixels, i.e. small picture elements arranged in two dimensional space. Pixels are represented by numerical values, the RGB pixel is comprised of three numerical values representing the intensity of the red, green and blue colors respectively. These values are often 8-bit numbers, i.e. ranging from 0–255 where a larger number corresponds to a higher intensity. An RGB pixel comprised of three 8–bit values can represent $256^3 \approx 16.7 \cdot 10^6$ different colors. In Figure 2.1 an RGB image[1] has been split into its three components to illustrate the intensities in each color band. The intensity is represented with a gray scale image and the corresponding histograms show the distributions of pixels over the range [0,255]. Kinect uses an infrared laser projector coupled with a monochrome camera to generate the depth-map that functions under any ambient light conditions. The depth-map is in standard VGA resolution, i.e. 640×480 pixels. Each and every pixel contains a value which represents the distance between the camera and the object in millimeters. The various pieces of furniture and educational appliances, e.g. the overhead projector, depicted in the corner of the room in Figure 2.2a are not distinguished in the depth-map in Figure 2.2b even though the furniture is not at the same depth as the wall. This is due to the operational range of Kinect, which is 0.7-6 meters and the appliances are at a distance greater than 6 meters from the camera [3].

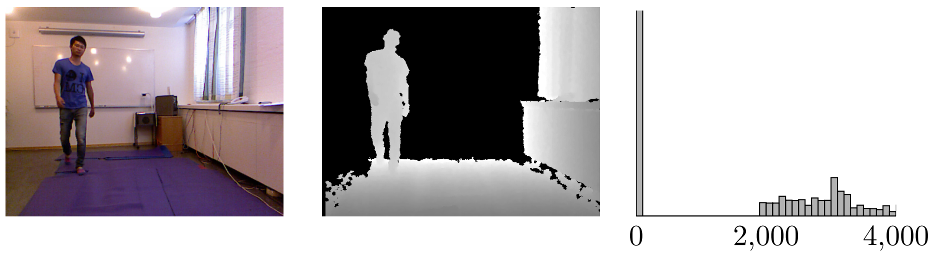
[1]The original RGB image is illustrated in Figure 2.2a.



(a) Red channel **(b)** Green channel **(c)** Blue channel

(d) Red histogram **(e)** Green histogram **(f)** Blue histogram

Figure 2.1: Red, green and blue channels of an RGB image. The histogram representations corresponds to the gray scale intensity of each color band.

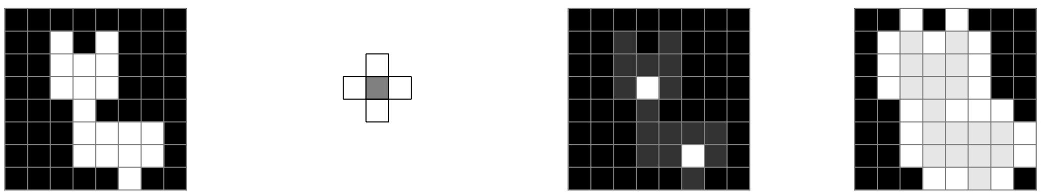


(a) RGB image **(b)** Depth map **(c)** Depth histogram

Figure 2.2: Components of an RGB-D image. The histogram represents the intensity in distance between the camera and the corresponding objects measured in millimeters. The depth map depicts the same correspondence in a gray scale image where black (or 0) corresponds to distances to far from, alt. to close to the camera.

2.2 Morphological Transforms

Morphological transforms, with regard to the subject of image processing, are operations that most often are applied to binary images. A binary image is an image constructed by pixels that are logically true/false (1/0 or white/black). The idea is to shrink or grow objects in images. Objects, also known as blobs, are represented as sets of connected (neighbouring) pixels that are logically true (1). In the following sections, true will imply logically true.



(a) Binary **(b)** SE **(c)** Erosion **(d)** Dilation

Figure 2.3: Figure 2.3a illustrates a binary image with a single object, i.e. neighbouring logically true pixels. By use of the structuring element depicted in Figure 2.3b the resulting images after applying erosion/dilation is illustrated in Figures 2.3c/2.3d.

2.2.1 Structuring Elements

A structuring element (SE) is a set of points arranged in a given shape, e.g. $N \times N$ points forming a square, line or a random configuration. The SE needs to specify a center point [4] and is applied to a binary image by projecting its center point onto every pixel (in parallel) that are true. The pixels are then marked true or false (depending on the operand) if every pixel covered by the SE is true. In other words, if every point of the SE maps to a pixel that belongs to an object, see Figure 2.3.

2.2.2 Erosion

Erosion, denoted $\ominus$, has the purpose of removing pixels that do not fit a given SE (shrinking an object). Applying the structuring element onto an image as mentioned above results in an image where pixels that fit the SE are marked true [4], see Figure 2.3c. Any pixel that is not marked true is removed from the object. The object is therefore likely to be comprised of less pixels after this operation.

2.2.3 Dilation

Dilation, denoted $\oplus$ has the purpose of adding pixels, i.e. grow objects. Applying the SE on a binary image therefore results in, as apposed to erosion, that the pixels that fit the SE are marked false [4], see Figure 2.3d. Later, any pixel that is marked true will have to mark some neighbouring pixels true in order for the SE to solely hit pixels that belongs to the object. In other words, when the SE is projected onto the set of pixels comprising the object, it should only hit true pixels. In general some pixels do not fit the SE and the object is therefore likely to be enlarged.

2.2.4 Closing and Opening

The terminology of closing and opening an object refers to the use of erosion and dilation in specified orders. If A represents a binary image and B is a SE, the operations is performed as follows. Opening, denoted "$\circ$" is the the equivalent to applying erosion followed by dilation, i.e. $A \circ B = (A \ominus B) \oplus B$. Closing, denoted "$\bullet$" is the opposite of opening, i.e. dilation followed by erosion, $A \bullet B = (A \oplus B) \ominus B$ [4].

2.2.5 Extended Maxima Transform

The different regional extremes in a picture can contain both interesting and irrelevant features. In order to make use of these features the Extened Maxima Transform can be used. The transform is preformed by extracting the regional maxima of the H-transform. This is done by first conducting the H-transform on the image. The H-transform is essentially a search for all maxima in an image that are greater then some value V. The H-transform filters the image by looking at the intensity values of the pixels with a contrast parameter. Any pixel that is V higher in intensity compared to its neighbours is considered a maxima. The Exteneded Maxima transform differs somewhat from the other kinds of morphological operations which instead use shape criteria set by the specified SE. [5]

2.3 Foreground Detection

One of the more important parts of computer vision is to prepare the video data for future processing. Not all parts of the video frames are of interest. Most often only a subsection of the frame contains the relevant information. In order to get usable data, the region of interest (ROI) has to be extracted. What parts of the video that belong to the foreground as well as what parts belong to the background, has to be determined. This evaluation can be done in a multitude of ways, each of which with their own strengths and weaknesses.

2.3.1 Color Thresholding

The process of thresholding an RGB image I_{rgb} by color is a simple process. The images of each color band, I_r (red), I_g (green) and I_b (blue) represents parts of the original image which upon separation makes it possible to mask out, i.e. use specific partitions $[\alpha_i,\beta_i]$ of the intensity range where $i = r,g,b$ and $0 \leq \alpha_i \leq \beta_i \leq 255$ for an image represented by 8-bit integers.

The values of α_i and β_i can be obtained by use of some interactive tool, e.g. Matlab's Color Thresholder. Once the values are obtained the masked images (I_{rm}, I_{gm}, I_{bm}) of each color band is obtained by

$$I_{im} = (I_i < \alpha_i) \land (I_i > \beta_i) \tag{2.1}$$

where $i = r,g,b$. The image mask I_M, containing the binary representation of the region of interest, is then constructed by the logical conjunction between the masked images from each color band as

$$I_M = I_{rm} \land I_{gm} \land I_{bm}. \tag{2.2}$$

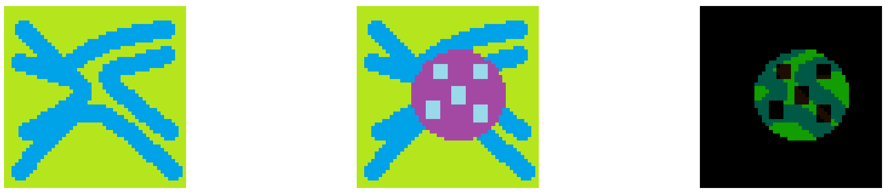
2.3.2 Background Subtraction

One of the most basic approaches in generating a ROI is to remove the background. Removing the background would in a perfect world just leave out the objects of interest. Unfortunately achieving this can be problematic, especially in a varied setting such as the ones found in real world applications of computer vision.

The process of background subtraction can be done in various ways. Depending on what type of video should be processed and its characteristics, e.g changes in illumination, different methodologies produce different results. Therefore there exists no universal rule as to which method should be used in any particular case.

One of the simplest techniques for performing a background subtraction is by manually subtract a frame, with only the background visible, from every other frame. By doing this, every pixel that belong to the background should be set to zero, i.e. becoming black since they contain the same data. The black pixels are considered to be irrelevant i.e. belonging to the background. Correspondingly, every pixel with contents greater than zero is set as foreground.

The method is only viable if an ”empty” frame, free of any objects or regions of interest, is accessible. This may or may not be the case, depending on what the computer is “watching”. If no such empty frame is available, the risk of introducing errors increases. The types of errors introduced are most often erroneous masks from the later steps in the method. The mask may, for example, get a ”phantom” silhouette of a object appearing wherever the faulty background contains ”non empty” data. However, when a proper background image is attainable, the subtraction should isolate any new objects entering the video. These objects can then, for example, be sifted through a color-threshold to obtain a ROI. Unfortunately this will not always give the desired results since it might be possible that the objects of interest contains similar pixels to the ones in the background which would make them black as well. This will in turn result in holes in the ROI after thresholding. An example of a manual subtraction resulting in a noisy ROI can be seen in Figure 2.4.



(a) Background. **(b)** Spotted ball in front. **(c)** Subtracted background.

Figure 2.4: In 2.4a the background is shown. A frame extracted from the video when a spotted ball passes over the background is shown in 2.4b. Subtracting 2.4a from 2.4b results in the picture shown in 2.4c. It can be seen that the subtraction creates ”holes” were the spots should be.

A similar approach is background subtraction in HSV color-space. HSV presents the pixel-value more intuitively than RGB. Instead of a red, green and blue channel HSV has a hue, saturation and brightness (value) channel. The saturation channel is of prominent interest since human skin more often than not tends to be relatively highly saturated, as shown in Figure 2.5b. Background subtraction in HSV color-space does not differ much from manual background subtraction, but is more of a variation of it. It operates in the same manner, where the current image is compared to the background image to spot differences. The most defining aspect of this method is unsurprisingly that it is performed in the HSV color-space.

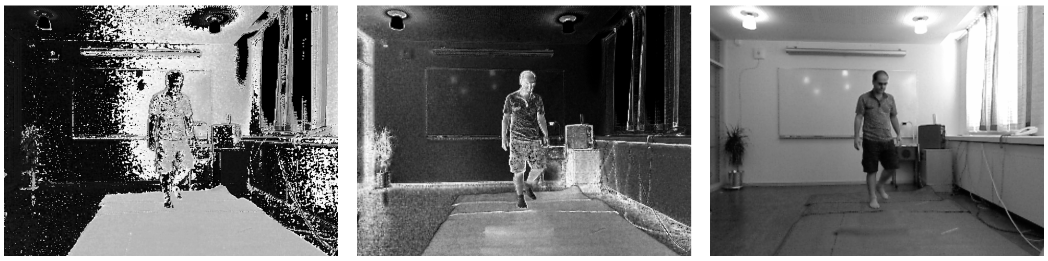

(a) Hue Channel **(b)** Saturation Channel **(c)** Brightness Channel

Figure 2.5: The hue channel does a bad job distinguishing the human from the mat, the saturation channel somewhat highlights the human and the brightness channel only acts as black and white version of the original image.

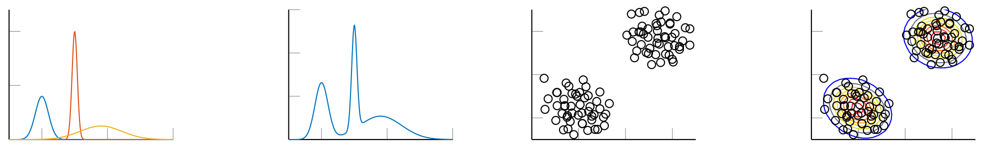
Another way of determining what parts of a video are in the background or in the foreground is to use a Gaussian mixture model (GMM). The purpose of the GMM is to to model the probability distribution of certain features, for example in a video stream. This can be used to create a model that describes the background. This model can then in turn be used to extract a ROI. The GMM consists of a weighted sum of Gaussian density distributions [6]. By concatenating a sum of Gaussians it is possible to model a more "uneven" density function as the one shown in Figure 2.6b. Using GMMs is a good choice if the gathered data points clusters together as can be seen in Figure 2.6c and Figure 2.6d. The GMM generally performs better than k-means clustering[2], especially if the data clusters differ in size and have some correlation between them [8].

A final approach, the method of eigenbackgrounds (EB), makes use of the method of principal component analysis (PCA). The process can be divided into training and execution phases.

The training phase could be conducted in a variety of ways, e.g. static by use of a single set of images or dynamic such that the model is re-trained during the execution phase [9]. The work flow of the training phase is structured in the following steps [10].

- A set of images $I = \{I_i : i = 1,2,\ldots,n\}$ with width w and height h is initially reshaped into a set of vectors $V = \{v_i : i = 1,2,\ldots,n\}$ where the vector v_i is constructed by vertical concatenation of the columns of I_i. The vectors in V is thus column vectors of size $m \times 1$ where $m = w \times h$.

[2]K-means clustering is a method for separating clusters in k classes and is a common method used for data mining [7].



(a) Three Gaussians. **(b)** The weighted sum of the Gaussians **(c)** Data points in distinct clusters. **(d)** Gaussian mixture model applied to clusters

Figure 2.6: 2.6a shows three separate Gaussians fitted to one-dimensional data. 2.6b shows the generated model which is the weighted sum of the previous curves. In 2.6c it is clear that the two-dimensional data belongs to two distinct clusters. The distribution of the different points are modeled by 2 Gaussians in 2.6d. It can be seen in the topography that the model captures the distribution of the data.

- An average vector $\bar{v}$ is computed as $\bar{v} = 1/n \sum_{i=0}^{n} v_i$.
- A matrix X is then constructed by horizontal concatenation of the normalized image vectors $x_i = v_i - \bar{v}$ as $X = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}$.
- Assuming that the images in I are very similar, a principal component analysis can be applied to represent the images in a vector space of lower dimension. Singular value decomposition (SVD) then yields $X = USV^T$ where U is an $m \times m$ orthogonal matrix, V^T is a $n \times n$ orthogonal matrix and S is an $m \times n$ diagonal matrix with the singular values of X. The first k columns of U approximates a basis of the column space of X if the k first columns of S contains non-zero singular values and the residual $n - k$ values is sufficiently close to zero.
- The basis $\widetilde{U}$ of the subspace then represents the EB model.

The execution phase is conducted by iteratively project new images $\tilde{I}$ onto the subspace as

$$\tilde{v}' = \widetilde{U}\widetilde{U}^T(\tilde{v} - \bar{v}) + \bar{v} \tag{2.3}$$

where $\tilde{v}$ is the new image represented as a column vector and $\tilde{v}'$ its projection onto the subspace. If $\tilde{I}$ contains significant deviations from the set of images in I, the absolute difference between the image and its projection at the corresponding locations would increase. By thresholding this difference

$$w = |\tilde{v} - \tilde{v}'| > T \tag{2.4}$$

where T is a given threshold, most anomalies would be detected. Finally to get the image representation of the deviations (moving objects), the values of w in intervals of h is concatenated horizontally into a two dimensional matrix (an image) [10], i.e. the first column consists of the first 1,...,h values, the second column of the $h+1$,...,$h+h$ values etc.

2.4 Histogram of Oriented Gradients

Histogram of Oriented Gradients, HOG, first described by Navneet Dalal and Bill Triggs in 2005, is an algorithm to identify specific features in an image. The first step to acquire the features of an image are by applying an 1-dimensional centered point discrete derivative mask,

$$[-1, 0, 1]. \tag{2.5}$$

The pixel evaluated is multiplied with the factor 0, the pixel left and right to it with -1 and 1 respectively. The sum of these three pixels are then the new value for the evaluated pixel. This is done both horizontally and vertically. When computed vertically the transposed version of the mask (2.5) is used. Every pixel determines a vote for an edge orientation histogram channel, where the vote is weighted on the gradient magnitude or a function of the gradient magnitude, e.g. the square-root of the gradient magnitude. All of the pixels that vote to the same histogram makes up a sub-image and these sub-images are referred to as cells. The shape of the cells can be set to either rectangular or radial, referred to as R-HOG and C-HOG respectively. An example of rectangular cells is shown in 2.7a where each cell consist of 9 pixels. The bins of the histogram are 20 degrees wide. Whether the gradient is unsigned or signed decides the spreading range of the histogram bins. When unsigned, the bins are spread evenly from 0 to 180 degrees and when signed, they are instead evenly distributed from 0 to 360 degrees.

HOG can be used on both colour and grayscale-images. The results are comparable between RGB and LAB colour spaces, but when restricting to grayscale, HOG performs slightly worse [11]. When HOG is applied on colour images each colour channel is calculated individually and the pixel's gradient vector becomes the channel with the biggest norm.

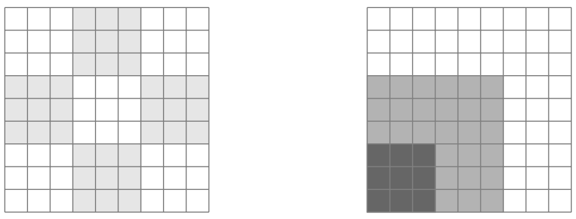
The histogram for each cell is used to normalize the contrast of larger spatial regions of the picture, referred to as blocks. The normalization of the blocks are overlapping, meaning that cells have the potential to be used several times. The overlap eliminates local inconsistencies of illumination. Otherwise these inconsistencies possibly can appear between block-boundaries. The block normalization schemes most commonly used by Dalal and Triggs are L2-hys and L1-sqrt [12].

Let all the histograms from a block make up the non-normalized vector $\boldsymbol{v}$, let $||\boldsymbol{v}||_k$ be the k-norm for $k = 1,2$ and let β be a small unspecified constant. The normalization scheme can then be defined as one of the following:

$$\text{L1-sqrt} \quad f = \sqrt{\boldsymbol{v}/(||\boldsymbol{v}||_1 + \beta)} \tag{2.6}$$

$$\text{L2-norm} \quad f = \boldsymbol{v}/(||\boldsymbol{v}||_2^2 + \beta^2) \tag{2.7}$$

L2-hys is a variation of L2-norm, where L2-norm is followed by a threshold which limits the maximum value of $\boldsymbol{v}$ to 0.2 and then re-normalized (with L2-norm).



(a) 9, 3 × 3 Cells **(b)** A 2 by 2 block consisting of 4 3-by-3 cells

Figure 2.7: The bottom left 2 × 2 block (light grey) is the only block that the bottom left 3× cell (dark grey) is part of in the 9 × 9 pixel image.

2.5 Optical Flow

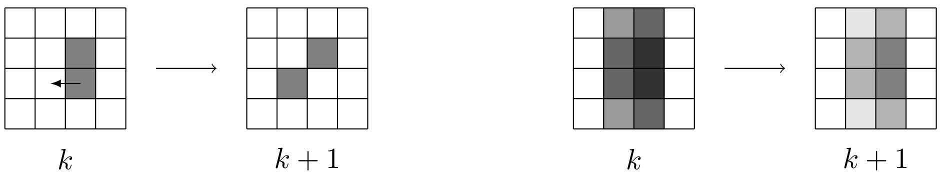
Most optical flow algorithms are based on a few assumptions, primarily *intensity constancy, gradient constancy* and *smoothness.* The intensity constancy assumption makes use of the assumption that pixel values do not change as a result of pixel displacements between frames (see Figure 2.8a). This could formally be stated as

$$I(x,y,t) = I(x+u, y+v, t+1), \tag{2.8}$$

where $I(x,y,t)$ denotes the image intensity function for a pixel (x,y,t) and $(u,v,1)^T$ is the displacement vector between images at time t and $t+1$ (see Figure 2.9) [13]. If displacements between images are assumed to be small (approximately one pixel), linearization of Equation 2.8 by a first order Taylor expansion is a valid approximation that yields the equation know as the *optical flow constraint* [13],

$$\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} = 0. \tag{2.9}$$

The optical flow constraint alone, as stated in [14] and [15], is not sufficient for the computation of velocity vectors since there exists two unknowns, u and v, and give rise to what is referred to as the *aperture problem* (see Section 2.5.4).



(a) Intensity constancy assumption **(b)** Gradient constancy assumption

Figure 2.8: Illustration of optical flow assumptions, (a) The illumination (intensity) of pixels are assumed to be constant when displacement occurs between frame k and $k+1$. (b) The image gradient is assumed not to change when change of illumination occurs between frame k and $k+1$.

The second assumption mentioned is stronger. It is stronger since it is invariant to changes in pixel intensities, i.e. gradient constancy make use of the assumption that the gradient of an image do not change due to changes in pixel intensities (see Figure 2.8b). Displacements of pixels are assumed not to affect the gradient [13], permitting the gradient constancy assumption to be stated as

$$\nabla I(x,y,t) = \nabla I(x+u,y+v,t+1) \tag{2.10}$$

where $\nabla I = (\partial_x,\partial_y)^T$ denotes the spatial gradient.

The third assumption, smoothness, is often used in order to help solve the aperture problem. It is stated in [14] that in most parts of an image, pixels tend to move similarly to their neighborhood, except of course, at object boundaries. A smoothness assumption therefore aids estimation of optical flow in parts of an image where gradients can not be estimated. Due to discontinuities at object boundaries, an assumption of a *piecewise* smooth flow, i.e. smoothness in small regions, often result in better estimates of the flow field [13],[14],[15]. In [15] it is also stated that filtering or smoothing (using low- or bandpass filters) prior to the computation of image differentials enhances the pattern signals relative to the noise inherent in the image and produces a more accurate structure.

2.5.1 Horn–Schunck

The Horn-Schunck method is a global method for estimation of optical flow and makes use of the assumptions mentioned earlier. It is denoted global since it introduces a global smoothing constraint. The smoothing constraint introduced is the square of gradient magnitudes [14]. The problem of estimating optical flow is defined as an energy minimization problem, giving rise to a linear system of equations that is solved by use of the numeric Gauss-Seidel method [14]. For a more thorough derivation of this method, see the original work of Horn and Schunck [16].

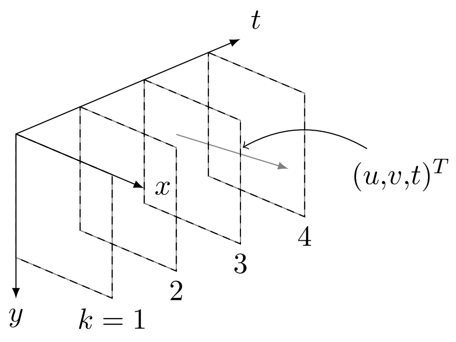


Figure 2.9: Illustration of the spatio-temporal space, between images at time t and $t+\Delta t$ (frame k and $k+\Delta k$) the displacement vectors $(u,v,t)^T$ shows how a pixel at (x,y,t), $t = 1,2,...$ moves.

2.5.2 Lucas–Kanade

The Lucas–Kanade method is a local method for estimation of optical flow. It is assumed that the optical flow field in small regions around each pixel is constant. It is denoted local due to the introduced smoothness assumption and solves the optical flow constraint in local regions of an image. Each of these regions give rise to overdetermined linear systems, systems that are solved by use of the least square method. For a more thorough derivation of this method, see the outline made by Lucas and Kanade [17].

2.5.3 The Coarse–to–Fine Approach

The coarse-to-fine approach can be viewed as a series of three steps, starting with an initialisation step followed by two iterative execution steps.

Initialisation Starts by re-sizing/down-sampling of a pair of images. Illustratively, by placing the down-sampled images on top of each other creates a Gaussian pyramid where images with coarser resolution appear upwards (see Figure 2.10). A pixel of an image above the base level is then a mean value of n pixels, where n depends on the re-sizing factor and the level, from the original image that forms the base of the pyramid. By starting optical flow estimation with an image of coarser resolution, large displacements will appear smaller. The images at the top level will be denoted i_{top1}, i_{top2} for image 1 and 2 respectively.

Execution step 1 Estimation of optical flow between images is iteratively calculated, starting at the top level where the flow is first estimated between i_{top1} and i_{top2}. The estimated flow field is then used to warp i_{top1} towards i_{top2}, creating a warped image, i^w_{top}. The next estimation of flow is then calculated between i^w_{top} and i_{top2} followed by the procedure of warping i^w_{top} towards i_{top2} again. This procedure of estimating optical flow and warping continues until the warped image converges with i_{top2}.

Execution step 2 Once i^w_{top} has converged with i_{top2} the estimated flow field is up-sampled to match the image at the next level, i.e. the same image with higher resolution. The up-sampled flow field is then used as an initial ”guess” to warp i_{next1} towards i_{next2}, where $i_{next1,2}$ are the images at the next level. Once the initial guess is done, the process is looped back to execution step 1, performing the same procedure on this level. This loop involving execution step 1 and 2 then continues to execute until the estimated flow of the original images have been retrieved.

2.5.4 The Aperture Problem

The rectangular area depicted in Figure 2.11 represents an area with a uniform texture, i.e. insufficient for gradient calculation. The aperture problem is that when looking

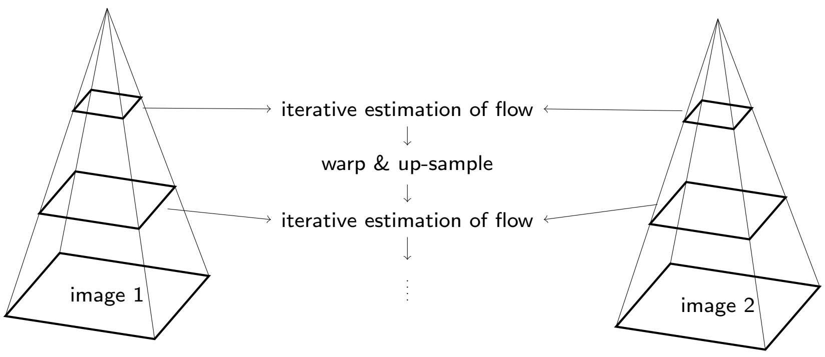


Figure 2.10: The coarse-to-fine approach of estimating optical flow. Gaussian pyramids depicts down-sampled images with coarser resolution upwards.

through an aperture, say 1, any horizontal movement of the rectangle would be impossible to register since only vertical pixel movement is seen. Aperture 3 depicts the same problem, being able to detect horizontal but not vertical movement. Through aperture 2 however, it is possible to detect both vertical and horizontal motion [14].

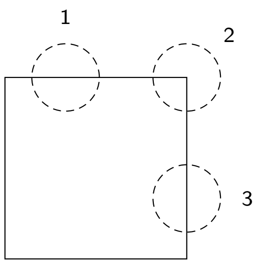
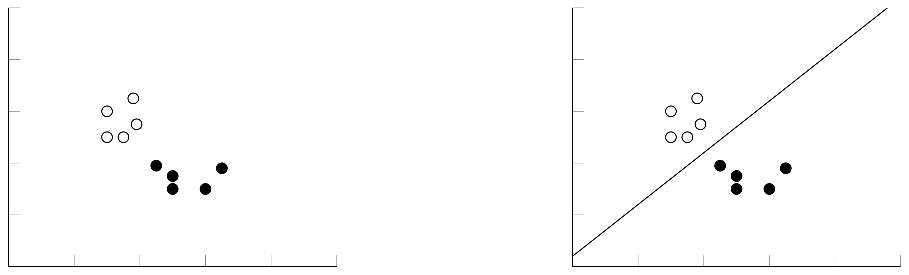


Figure 2.11: Uniform textured rectangular surface with three circular apertures.

2.6 Support Vector Machines

SVM or, "Support Vector Machine", is a method to systematically allow a computer to undergo supervised learning. The SVM generates on a given data-set, a classifier capable of separation of two classes of objects. The SVM is essentially a two object classifier [18] but in theory the possibility of cascading SVM:s together allows classifying into two or more categories.

The idea is that all entities to be classified are represented as a feature vector. This feature vector may be of an arbitrary dimension, that is, it can contain n elements. All feature vectors that are passed into the SVM needs to have the same size however. The elements in the feature vector may be some measure of certain characteristics on the object or something else that distinguishes the objects among each other. When plotted these vectors make up points in an n-dimensional space. The plot depicted by 2.12a shows an example of this using two-dimensional feature vectors. Assume that there are two possible classes, class A (black circles) and class B (white circles). As seen in 2.12a points from the same class, here having the same color, cluster together and are quite easily distinguished by the human mind. For a computer however, the task is not so trivial. Where one class ends and another begins has to be determined. One way to asses this is by using the SVM. The SVM-algorithm tries to separate the points into two classes via a hyperplane. Since there usually are quite a few such hyperplanes the SVM tries to choose the one that guarantees that the maximum distance betwixt the two closest data-points (also called support vectors) is achieved. 2.12b illustrates the separation of the two classes by the hyperplane. Since the demonstration-data is two-dimensional the hyperplane is reduced to a line. Any data that falls to either side of the line is assumed to belong to that class. This means that in the example below, any data point above the black line is considered to belong to class B and vice versa [19].

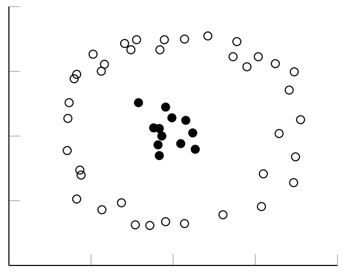


(a) Feature vectors in 2D space. **(b)** Hyperplane separating two classes.

Figure 2.12: The feature vectors can be seen in 2D space in 2.12a. It is clear that vectors belonging to the same class cluster together. In 2.12b the SVM has introduced a hyperplane that clearly separates the classes on each side of the plane (a line in 2D).

Unfortunately some cases exists where the data is not so well clustered. In that case the data may not be linearly separable as in 2.12b. What this means is that there does not exist a hyperplane such that it divides the points into two categories. Nevertheless it might be possible for the human imagination to find some sort of similarity between them. In 2.13a it is clearly displayed that the data shows some sort of correlation. All the black circles are clustered together in the middle and the white ones seems to have some sort of minimum distance from the origin. Separating them using a plane (line) is futile. In this case there are two possible alternatives accessible. One option is to choose the hyperplane that presents the smallest error. [18] In this case this is not possible since it would introduce an error so big that the classifier would lose its meaning, as can be seen in 2.13b.

The other option is to use the kernel trick. The kernel trick means that by some function called the kernel, the feature vectors are mapped into a higher dimensional space. There also exists another possibility if additional non-correlated features are available. By introducing the extra features into the feature vectors, it would increase their dimensionality (which is what the kernel essentially does). In the higher dimension the data might be linearly separable as seen in 2.14a. Thanks to the kernel function or


(a) Feature vectors in 2D space.

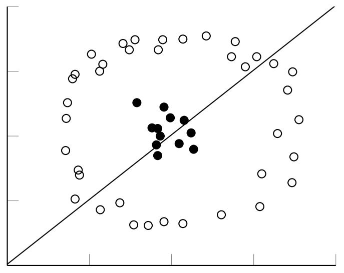
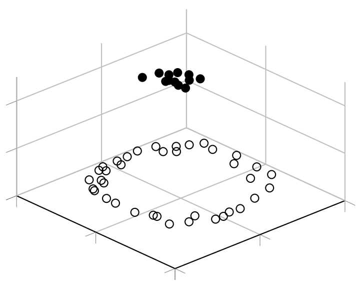

(b) Hyperplane separating the data.

Figure 2.13: As can be seen in 2.13a the data clearly clusters together in some way but it is also apparent that the data is not linearly separable. In 2.13b we can see a failed attempt at separating the classes. The data points are separated but the hyperplane introduces an unacceptable error which makes it useless for classification.


(a) 3D space view

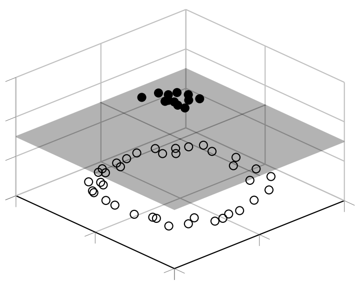

(b) Hyperplane in 3D space.

Figure 2.14: Using a kernel the data from 2.13a can be observed to be mapped to a higher dimensional space (in this case 3D) which is depicted by 2.14a. In this case it is obvious that there exists hyperplanes that can separate the data without introducing any errors. One such plane is shown in 2.14b

the increase of dimensionality on the feature vectors, it was made possible to achieve the state shown in 2.14a. In this case it is clear that the data is separable by a plane passing between the same colored data points. One such plane is shown in 2.14b.

In the cases where the kernel function is used it is usually best to utilize the basic ones, especially if the data set is large. This is because the execution of the program would otherwise be slowed down due to the costly complex calculations needed. The most commonly used kernel functions are linear, Gaussian or polynomial. There also exists the choice of the RBF kernel which usually gives satisfying results [20].

2.6.1 Training of the SVM

The training is done by allowing the SVM to analyze several data points and then try to find the dividing hyperplane, given by equation 2.11. w and b are model parameters and x_i is one of the feature vectors. All available training-data is mapped into a suitable

n-dimensional space, where n is the dimensionality of the data. The distance from the closest data point to the dividing hyperplane is calculated via equation 2.12. This distance, r, should be as large as possible and is basically a quadratic optimization under linear constraints [19]. This is a common, and known problem in mathematics and can be solved by maximizing L_p in equation 2.13 [18].

$$\mathbf{w}^T\mathbf{x_i} = 0 \tag{2.11}$$

$$r = \frac{(\mathbf{w}^T\mathbf{x_i} + b)}{||\mathbf{w}||} \tag{2.12}$$

$$\mathbf{L_p} = \frac{1}{2}||\mathbf{w}|| - \sum_{i=1}^{t} \alpha_i y_i(\mathbf{w}\mathbf{x_i} + b) + \sum_{i=1}^{t} \alpha_i \tag{2.13}$$

Here α_i ($i = 1,2,3\ldots t$) is a non-negative number such that the partial derivatives of L_p with respect to α_i are zero. t is the number of training cases (i.e. data points) that the SVM is feeded with, the vector $\boldsymbol{w}$ and the scalar b define the hyperplane as seen in equation 1. The value $\boldsymbol{L_p}$ is called the Lagrangian.

2.7 Predictive Analytics

To evaluate the performance of machine learning techniques, such as SVM classification, predictive analytics can be used.

2.7.1 Confusion Matrix

One technique that predictive analytics encompasses is a table of confusion, also referred to as a confusion matrix (CM). The confusion matrix is a $n \times n$ table layout which highlights where a supervised learning algorithm confuses one class for another, i.e. the CM visualizes the performance of the classifier. For a binary classification n is equal to 2 since there are two classes. As shown below in Table 2.1 the input data for the CM is true positives (TP), false positives (FP), false negatives (FN) and true negatives (TN). TP is where actual true instances are classified as true, whereas FN is where true instances instead incorrectly are marked as false. TN is where actual false instances are classified as false and FP is where false instances instead incorrectly are labeled as true.

Table 2.1: A 2×2 confusion matrix for assessment of binary classification rates.

	Is True	Is False
Lbl. True	TP	FP
Lbl. False	FN	TN

The content in Table 2.1 can then be used to acquire, inter alia, the correct classification rate (CCR) which is also referred to as accuracy (ACC), the miss classification rate (MCR), true positive rate (TPR), true positive rate (TNR), the false positive rate (FPR), the false negative rate (FNR). The calculation procedure of these rates are given by Equation 2.14–2.19.

$$CCR = (TP + TN)/(TP + FP + FN + TN) \tag{2.14}$$

$$MCR = (FP + FN)/(TP + FP + FN + TN) \tag{2.15}$$

$$TPR = TP/(TP + FN) \tag{2.16}$$

$$TNR = TN/(TN + FP) \tag{2.17}$$

$$FPR = FP/(FP + TN) \tag{2.18}$$

$$FNR = FN/(TP + FN) \tag{2.19}$$

2.7.2 Cross-Validation

Cross-validation (CV) is a technique used to estimate classification-errors. When a method needs to be calibrated on known data to later be tested on separate data and the available sample size of data is not of satisfying proportion, CV can be used. There exists several variations of CV and k-Fold Cross-Validation is one of those.

When applying k-Fold CV the data set is divided into k roughly equally sized groups. One group is used as validation data while the other $k-1$ groups are used as training data. This procedure is then repeated k times to make sure all groups of data are validated exactly once. Once all k sets of prediction statistics are generated, the mean of said statistics can be calculated. These mean values act as an estimation of the prediction statistics.

Chapter 3

Method

This chapter will focus on the process of designing algorithms and the methodologies used to extract the necessary data. Part 3.1 mainly consists of descriptions on how the detection of the foreground and ROI was performed. Part 3.2 describes the procedures used to generate the different features and the process of concatenating them into a single feature vector. Finally part 3.3 outlines the creation of the actual classifier, what different parameters that were used, how they were obtained, and the type of kernel used. One thing to note is that we have knowingly omitted most of exact parameter values since many of them are unique to this particular data-set.

3.1 Approaches for Human Detection

This section serves to give an outline of the algorithms used for video pre-processing.

3.1.1 Skeleton Tracking by Color Thresholding

A skeleton tracking approach for detecting motion was implemented. A video, captured by a Kinect camera, can be set up to project a skeleton onto the human object, connecting its joints. The skeleton is drawn with a distinct red color, making it possible to apply color thresholding in order to extract it. Detection of the skeleton from an RGB image was conducted in accordance to the method described in section 2.3.1. The RGB image was initially split in its respective color components and thresholded by a given value T. An initial image mask I_{M*} was then computed by the logical conjunction of the thresholded image components. To reduce the effect of excessive objects, pixel areas smaller than a given size N were removed from the initial mask. A bounding box of the object in I_{M*} was then extracted and to successfully fit the entire person, the box had to be scaled. The output was thus an image mask I_M, a rectangular object (area) of the image.

3.1.2 Isolation of the ROI by Manual Background Subtraction

To isolate the ROI from the rest of the image a method of background subtraction was tested. The background- subtraction procedure began by finding an "empty" frame, containing only the background setting, obtained by manual extraction from the video stream. This background-frame was then subtracted from every other frame in the video, making most of the data (pixel values) in the images zero. After that, every color band of each frame, red, green and blue, was thresholded on intensity. The intensity level had to be manually tuned depending on the type of video being processed. The results from the thresholding was then recombined via logical AND operation, producing a logical mask.

The logical mask obtained from the previous steps was then passed through a series of morphological operations to remove noise and tidy up the contours. In this case opening and closing. This was done using two different SE:s, one octagon of size 6 and one disk of size 4. The final mask was then used to remove everything but the ROI from each of the video frames. The same procedure could also be done on the depth map to instead obtain a mask generated from the D-part of the RGB-D data. In that case however, the thresholding could only be performed on the one band present, with a different intensity level tailored to the depth image.

3.1.3 Subduing the Effects of Inconsistent Illumination via Background Averaging

When the illumination of the video changes, a manual background subtraction might produce unreliable results. Therefore it could be of interest to generate a new background with updated lightning conditions every few frames. An attempt at one such method was implemented in the following way: by masking out the bounding box of the ROI from the current frame, leaving only the background, and then combining the newly masked image with the currently saved image of the background. First the ROI of the current frame was used to generate an inverted mask. That is, a mask that had ones everywhere but on the ROI-box. This mask was then used to extract the partial background from the current frame. The same area and location as the bounding box on the original background was then multiplied by two. Then the average of the new and old background was taken, producing a new one but with different illumination which was saved as the current background. This new background could then be used for manual subtraction. Here the reason for the multiplication by two in the previous step becomes clear. If it would have been omitted, a region much dimmer than the rest of the image would appear at the same position as the ROI because of the averaging.

3.1.4 Specframe

When analysing different ways to extract the human from the background, a promising method was found. Performing a manual gray-scale background subtraction did isolate the foreground (human) a bit but it also produced much noise as well. To smooth out the noise, a 2×2 median filter was applied. Despite of this some noise still remained. Realizing that it was much dimmer than the actual foreground a change in contrast and brightness was made to reduce the visibility of the noise by so much as to in fact set the intensity values of the ”noise-pixels” to zero. The actual foreground being brighter, would become dimmer, but not disappear. Increasing the contrast and brightness again after the initial reduction would then only affect the foreground, making it clearly visible, almost like a binary mask. This new high-contrast frame was denoted “specframe”, short for “special-frame”.

The change in contrast in order to obtain the specframe was found to be dependent on how the illumination changed and therefore had to be manually set to suit a particular type of video. As such it would be impossible to give a universal value that always generates good results.

3.1.5 Foreground Detection Using Gaussian Mixture Models

To eliminate the background from the objects of interest a GMM can be used. Usually quite a few frames of the background is supplied to train the model with. Unfortunately this could not be done in this case since only a very limited number of ”empty” frames from the data-set were available. Most of the time not even one. In order to circumvent this limitation the specframes were used instead. Almost being binary masks in their own right, the GMM could easily find the actual foreground. The GMM was set up to create the model with 2 separate Gaussians, using only the first frame (specframe) as training.

The other specframes generated from the video were then passed through the mixture model, producing a binary mask. The produced mask underwent some morphological operations to improve the ROI. The mask was filled so that any small holes in larger blobs were removed. Then a dilation with a SE in the form of a 4 pixel wide disk was performed. This was followed by a closing (using the same SE) and finally a blob-size filtering where any blob smaller than fixed pixel size (5000 in this case) was removed. The final mask and the ROI for that particular frame could then be used to extract important data later. Using a blob-analyzer object from Matlab’s computer vision toolbox, additional data, e.g. center of mass with respect to the mask could be obtained.

3.1.6 Extended Maxima Transform on Specframe

By performing the same grayscale manual background subtraction as above, followed by a contrast increase, the earlier mentioned “specframe” was produced. Then, by performing an extended maxima transform on the specframe, a true binary mask could be generated. The scalar level of the EMT was a case-specific parameter and as such it had to, like the rest, be manually tuned to suit different videos illumination characteristics. By letting the mask produced by the EMT trough a similar series of morphological operations as used in the method with the GMM, a more convenient mask with less noise and cleaner edges was received. This mask could then in turn be used to extract the relevant data from the video frames.

3.1.7 Blob-tracking in HSV color-space

The first step of this method was to perform a color-space transformation, where each and every element of an input array of RGB-images changes color-space from RGB to HSV. A background subtraction with regard to the saturation-channel was then performed. A threshold was then used to create a logical matrix. The logical matrix was filtered and only connected components with a size within a certain range were kept. In the hue-channel, cold, i.e. blue and green hue, got strained off. The cold hue threshold was implemented to reduce the risk of obtaining highly saturated areas with cold hue which, judging from the test data videos, most likely were not human skin. A mask was created from the filtered hue-channel and the logical matrix. The biggest connected component was then computed. The x- and y-coordinates of said component's mass-center was then put into a 1-by-n cell-array, where n was the number of frames represented in the input.

3.1.8 Motion Detection with Eigenbackgrounds

An EB model for detection of moving objects was implemented with both a static set of images as well as a dynamic approach for the computation of the eigenspace. Computation of the approximated basis of the column space were identical to the method outlined in section 2.3.2. For the static approach, an image that solely captures background (for the respective video) were reproduced as a set of images where each reproduction of the image had a gradual change in intensity. The approximated basis where thus computed ones and used to map every new frame. The dynamic approach were set up to iteratively calculate new approximations of the model. The approximation of the column space basis were thus calculated from N images prior to the currently evaluated image.

3.2 Design of Feature Vectors

In this section the process of feature design is described. Initially the different types of features are explained, followed by sections describing fusion at the feature level as well as normalization techniques.

3.2.1 Centroid Velocity- and Head-Centroid Relation

These features were generated by tracking the center of mass for the human blob and head, obtained from the masks generated trough the previous steps in section 3.1. By taking the Euclidean distance between two points (head or centroid) from two consecutive frames and multiplying by the frame rate (FPS), the velocity (pixels/s) of the moving points could be calculated. Omitting the x-coordinate meant that only movements on the y-axis would be taken into consideration for the velocity. In the case where the centroid might be lost in the next frame the old coordinate for the centroid was assumed to still be valid and used once more. The velocity for the center of mass between two consecutive frames was then added into a vector v_{CV}.

$$v_{CV} = FPS \cdot \begin{bmatrix} ||(C_{f_1}) - (C_{f_2})|| & ||(C_{f_2}) - (C_{f_3})|| & \dots & ||(C_{f_{n-1}}) - (C_n)|| \end{bmatrix} \tag{3.1}$$

where C_{f_i} denotes the centroid on frame i. Performing statistical measurements served to reduce the data size of the vector to a manageable one. The measurements used were the mean and the variance. This therefore resulted in a two-tuple f_{CV} on the following form

$$f_{CV} = \begin{bmatrix} E[v_{CV}] & Var(v_{CV}). \end{bmatrix} \tag{3.2}$$

Another similar feature was created by calculating the Euclidean distance between the center point of the human blob and the center point of the head. Utilizing two pre-process methods, blob-tracking HSV-background subtraction and ROI-extraction. For every frame where any of these points were unavailable, the values missing defaulted to the middle of the picture. For example if the data used was a video with a resolution of 640×480, the replacement values would be 320 and 240. To reduce the size of the feature the mean and the variance was applied to the vector v_{HC} that held the head-to-centroid relation of each frame.

$$v_{HC} = \begin{bmatrix} ||(C_{f_1}) - (H_{f_1})|| & ||(C_{f_2}) - (H_{f_2})|| & \dots & ||(C_{f_n}) - (H_n)|| \end{bmatrix} \tag{3.3}$$

where C_{f_i} denotes the centroid on frame i. Taking the statistical measures then gives the feature vector f_{HC} as a two-tuple on the form

$$f_{HC} = \begin{bmatrix} E[v_{CV}] & Var(v_{CV}). \end{bmatrix} \tag{3.4}$$

3.2.2 HOG Features

The process of generating features from the HOG began by extracting out only the contents present in the ROI generated by the detection algorithm using Matlab default parameters. Then, by taking the magnitude of the difference between the HOG of two consecutive frames it could be reduced to a single value. This value was then placed in a vector for further processing. By making statistical calculations, such as taking the variance, the dimensionality was reduced further. The process also served to extract the characteristics of the HOG as well. The statistical measurements used to generate the feature vector were: mean and variance in that order, creating a two-tuple for later use. The following equation explains the process of obtaining the two tuple:

$$v_{HOG} = \begin{bmatrix} ||\texttt{HOG}(ROI_{f_1}) - \texttt{HOG}(ROI_{f_2})|| & \dots & ||\texttt{HOG}(ROI_{f_{n-1}}) - \texttt{HOG}(ROI_{f_n})|| \end{bmatrix} \tag{3.5}$$

where f_i denotes frame number i. Performing the statistical measures then gives the total feature vector

$$f_{HOG} = \begin{bmatrix} E[v_{HOG}] & Var(v_{HOG}) \end{bmatrix} \tag{3.6}$$

3.2.3 Optical Flow Features

Optical flow was calculated between adjacent images in a set of frames (a video) corresponding to individual activities. Standard parameters were used for both Lucas-Kanade and Horn-Schunck as described in [21]. For every set of frames, calculations generated two sets of flow fields (x and y matrices), one for vertical movement (v_y) and one for horizontal movement (v_x). Average flow fields ($\bar{v}_y, \bar{v}_x$) was constructed by simply averaging the initial two sets in the temporal dimension. Feature components was then computed from $\bar{v}_y$ and $\bar{v}_x$. Initially two types of histograms was calculated, one known as histogram of oriented optical flow (HOOF) where the orientation of flow ($\arctan(\bar{v}_y/\bar{v}_x)$) was used as a criteria for assigning each flow vector to a bin. The magnitude of that vector was then used to assess the contribution to the histogram. The second type of histogram, denoted histogram of optical flow (HOF) simply used the magnitude for both assigning flow vectors to bins and assess their contribution. A non-histogram feature was constructed by use of a PCA on the average flow fields, projecting each field onto its first eigenvector, giving two vectors v_{EIGx}, v_{EIGy}. The feature was then constructed by taking the mean of these vectors resulting in a two-tuple feature f_{EIG} as

$$f_{EIG} = [\, E[v_{EIGx}],\, E[v_{EIGy}]\,].$$

Further component was the mean and variance of flow. For each flow field v_{xi}, v_{yi} where $i = 1,2,\ldots,n$ and n is equals the number of frames capturing an activity, the mean $E[v_{xi}], E[v_{yi}]$ and variance $var(v_{xi}), var(v_{yi})$ values was calculated. For each pair of frames (i and $i+1$) a scalar value for vertical/horizontal mean/variance thus gives $1 \times n$ vectors ($f_{VARx}, f_{VARy}, f_{Ex}, f_{Ey}$) with the corresponding values. The final features f_{VAR} and f_E was then generated by creating histograms of the vectors values as

$$\begin{aligned} f_{VAR} &= k\text{-bin histogram of } [f_{VARx}, f_{VARy}] \\ f_E &= k\text{-bin histogram of } [f_{Ex}, f_{Ey}]. \end{aligned}$$

The histograms were calculated by initially identifying the maximum v_{max} and minimum v_{min} values which was used as start/end points. Every edge in the histogram was then calculated by splitting the range $v_{max} - v_{min}$ into k bins. The content/value of each bin was then assigned with the sum of every value v_i that was mapped to that bin, i.e. every value that was in the range of specific bin. Finally a normalization of the histogram was conducted by dividing the value of each bin with the total value of every bin. k values used was 30 and 9.

3.2.4 Choosing the Type of Pre-Processing for Each Feature

There were some options for how each feature was pre-processed. i.e. how the human was detected. To decide what settings to use a table was set up, displaying the correct classification rate (CCR) for each feature's different pre-process-method. That means a

SVM was trained and tested for a single feature and this procedure was then repeated until all features had been tested. The few that had a CCR above 80% were then taken in consideration for usage in the final set of features.

3.2.5 Normalizing and Linear Scaling of Features

Since the data inside the different feature tuples sometimes had a drastic size difference and were often quite big overall, they were normalized in order to speed up the calculations in the SVM and also to improve its classification rate. This normalization was done using two different methods. The first one, and perhaps the most commonly used, involved calculating the mean and standard deviation for each column in the total feature matrix F. Then each individual value in the vector was normalized as follows:

$$\hat{F}_{ij} = \frac{F_{ij} - \texttt{mean}(F_j)}{\texttt{std}(F_j)} \tag{3.7}$$

where F_{ij} denotes the item at row i and column j and F_j the entire column

For example: consider a feature matrix F very similar to one generated from the centroid:

$$F = \begin{bmatrix} f_{CV_1} \\ f_{CV_2} \\ f_{CV_3} \end{bmatrix} = \begin{bmatrix} 845 & 7531590 \\ 778 & 6547530 \\ 875 & 9852014 \end{bmatrix} \rightarrow \begin{bmatrix} 0.304 & -0.322 \\ -1.348 & -1.032 \\ 1.044 & -1.353 \end{bmatrix} \tag{3.8}$$

Using this method, each individual value was normalized to lie close to the range [-1 1] which simplified the calculations in the SVM. The other method followed a more heuristic approach in the sense that it only served to reduce the size of the data. In this method, every value in the total feature vector was rounded up/down to the closest power of 10. Then the inverse of the most frequent power for each respective column was used to linearly scale down each value in that column. This process, served to reduce each value close to a range [0 1] whilst still retaining the original difference between the falls and lies. A small example (3.9) using the same feature matrix as in equation 3.7.

$$F = \begin{bmatrix} f_{CV_1} \\ f_{CV_2} \\ f_{CV_3} \end{bmatrix} = \begin{bmatrix} 845 & 7531590 \\ 778 & 6547530 \\ 875 & 9852014 \end{bmatrix} \rightarrow \begin{bmatrix} 0.845 & 0{,}753 \\ 0.778 & 0{,}655 \\ 0.875 & 0{,}985 \end{bmatrix} \tag{3.9}$$

3.2.6 Feature Fusion

As mentioned above, several types of features were generated in order to create accurate ”models” describing different activities. In order to avoid that certain types of features wrongfully classified one activity as something different, the combination or ”fusion” of several features was used as the primary model (feature vector) for classification. For example, the centroid velocity feature, mentioned above, might classify a falling object, such as a ball, as a ”falling human” since it describes the movement of the center of mass. By fusing diverse features f_i where $i = 1,2,\ldots,n$, that describes different aspects of an activity, a combined vector $f = [\, f_1, f_2, \ldots, f_n \,]$ will probably give a more accurate result. As will be discussed later on, the feature vector giving the most satisfying result was built using five types of features, which included: centroid velocity (f_{CV}), head-to-centroid distance (f_{HC}), HOG difference (f_{HOG}), an optical flow variance histogram (f_{VAR}) and finally a feature based on PCA of the temporal mean of optical flow (f_{EIG}). Resulting in a feature vector f as

$$f = [\, f_{CV},\, f_{HC},\, f_{HOG},\, f_{HOF},\, f_{EIG}\,].$$

3.3 The Classification Model

In this section the method of how the classifier was trained, the measurement of its accuracy and false alarm rate is described. Also a small outline of the process of down-sampling videos is given.

3.3.1 SVM Classifier

The SVM model was created using the package from libSVM [20]. The type of SVM used was a standard binary classifier. Using the normalized features from the data-set, the SVM was trained on 383 lies and 383 falls. A function was also created to draw 373 random samples from the data-set to train the SVM with. The randomizing process was created using the Matlab-native function `rand()`. The remaining parts of the data-set was then used to test the SVM during the initial trials. By trying (training the SVM and testing it) each feature individually, the ones with promising CCR could be sorted from the ones with poorer performance. The promising features were then fused together and a new SVM was trained using the fused data. When the decision of which features to use had been made, the process began of tuning the SVM so as to not over-fit the data. Since the kernel was RBF and the SVM of type C-svm, there were two parameters, c and g, to optimize. At first this was done using trial and error. Setting the SVM to perform a 100-fold cross-validation, different values of c and g were tried and the ones that generated the best results i.e. gave the best CCR were recorded. Just a few trials were made however, enough to get a general sense of right direction. To make sure that the real optimal choice was made, or at least very close to it, an exhaustive search (grid search) was programmed. Letting c run from 0.05 to 2.55 in increments of 0.1 and for each such value try g between 0.005 to 0.5 with increments of 0.005, good values for c and g were found. The same procedure was used to find the most suitable c–value for the SVM when using the linear kernel. The two SVM:s where then compared and the one with the highest CCR was picked as the final classification model.

3.3.2 Confusion Matrix and Cross-Validation

To acquire the CCR, MCR, TPR and TNR, the built-in cross-validation function of the libSVM package was used. The variation of cross-validation used was 100 fold. This function presented the results in percentage. However the cross-validation did not calculate the FNR and FPR. Both FNR and FPR were of uttermost importance and were acquired utilizing confusion matrices. As can be observed in Equation 2.16 and 2.17, TP, FP, FN and TN can be extracted. These four values are required for setting up a CM.

When calculating CCR and MCR, the whole data-set was used for cross-validation. This was not the case when calculating FNR and FPR. Instead 20 instances of 100 fold cross-validation on approximately 250 randomly selected data were used, i.e. CV was performed on chunks of data and not the complete set. For each instance the confusion matrix was set up and FNR and FPR acquired. The mean of these 20 values of FNR and FPR gave an estimation of the actual FNR and FPR.

3.3.3 Temporal Down Sampling

To reduce the computing time and also increase the classification accuracy of videos with high frame-rate, a temporal down-sampling method was implemented. Quite simply the method was set to consider only every other frame. For example removing every third frame would reduce the effective number of frames to compute by a third. Since the data-set used in this project consisted of videos with a frame rate $\zeta_{20} = 20$ frames per second (FPS), any other video had to be down sampled to this FPS in order to generate features that captured the characteristics of an activity in the same way. For example if a video $V = [I_1, I_2, \ldots, I_n]$ with n frames has a frame rate $\zeta_x \geq \zeta_{20}$ FPS, then a down sampled version $V_\downarrow$ would be the set of frames

$$V_\downarrow = \left\{ I_i = V\left[\left\lceil \frac{\zeta_x}{\zeta_{20}} \times i \right\rceil \right] \;,\; i = 1,2,\ldots,N \right\}$$

where $N = \frac{\zeta_{20}}{\zeta_x} \times n$.

Chapter 4

Experimental Results

In this chapter, the results obtained in the pre-processing stage, feature design/ generation and classification process is outlined. In Section 4.1 the outcome generated by different pre-processing approaches will be given and the results, with respect to ”degree of detection”, will be illustratively presented. Section 4.2 shows feature characteristics and average values (vectors) are used as a result measure. Section 4.3 gives the obtained classification rates and run-time results for various setups.

4.1 Image Pre-processing

The performance measurements of the detection methods are outlined in Table 4.1 and the frame references corresponds to Figure 4.2. Figure 4.1 shows the set of images used to compare the different detection techniques. It is also stepped 2 frames at at time in order to cover most of the process of falling.

The results of color thresholding can be seen in frame 1–4 of Figure 4.2. The enclosed area given by the bounding box (BB) fits the entire person and gives consistent results for most sequences capturing the skeleton. The execution time was quite fast but the notable downside was, however, the relatively large amount of background data that gets interpreted as foreground.

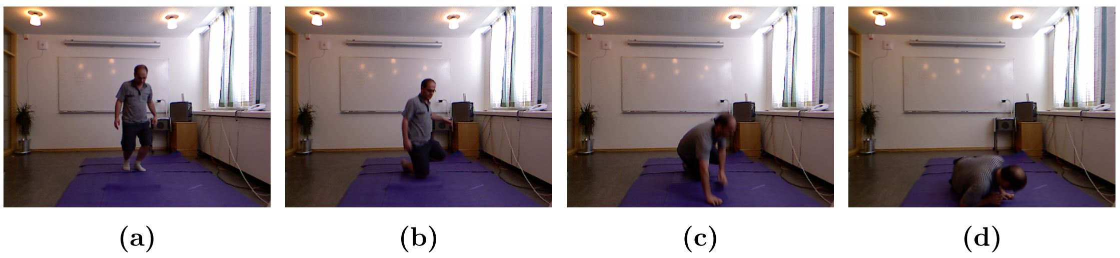


Figure 4.1: Image set used for visual comparison of proposed methods, results are shown in Figure 4.2. These images are extracted with a gap of one frame, i.e. frame $n \rightarrow n+2 \rightarrow n+4 \rightarrow n+6$ in order to illustrate the majority of a fall.

Eigenbackgrounds, both static (frame 5–8) and dynamic (frame 9–12) of Figure 4.2 generated average to poor results. The static approach had a reasonable run-time but was probably the one most sensitive to uneven illumination change. The run-time given in Table 4.1 is an approximation where the background model had been computed from a set of 40 images. The initial computation time of this model was approximately 3s. The dynamic approach was slower, Table 4.1 gives the result where the dynamically computed model consisted of three images prior to the one currently being analysed. This model was especially sensitive to the speed of motion. If slow motion occurred (or no motion at all) the model would interpret almost everything as background. If on the other hand fast motion, i.e. large displacements between frames occurred, the model would capture the trajectory.

As can be seen in frame 17–20 of Figure 4.2 the GMM on speccframe method performs quite well. It does however produce a slight error in the first picture. Fortunately this error is caught elsewhere and as seen the ROI only covers the part of the mask that contains the human. Also the centroid closely follows the actual middle of the person. The mask has an overall good bound on the human although it loses some parts of him at the end. Its resource consumption, shown in Table 4.1, shows a quite short computation time but a somewhat high memory usage.

The Manual background subtraction frames (13-16) shows quite the clear capture on the first two cases but rapidly deteriorates in the later frames, only covering parts of the human. It does consume a little less memory than the GMM but takes a little bit longer to perform. The depth method also captures the human quite well in the initial frames but like the rest, loses its accuracy during later frames. It also takes significantly longer to process than GMM or manual subtraction. Most of the methods also have the problem of accurately capturing the persons feet.

The HSV subtraction brought various results. The algorithm sometimes completely filtered away the head and reported ”nothing”. However, as shown in frame-sequence 25–28 it has the potential to work as intended, tracking the head. Out of all the detection methods implemented, HSV subtraction had the longest run time as well as the highest memory usage by far.

When considering the stated performance measurements in Table 4.1 the parameter regarding memory consumption should not be taken at face value since the method used to measure it was quite crass.

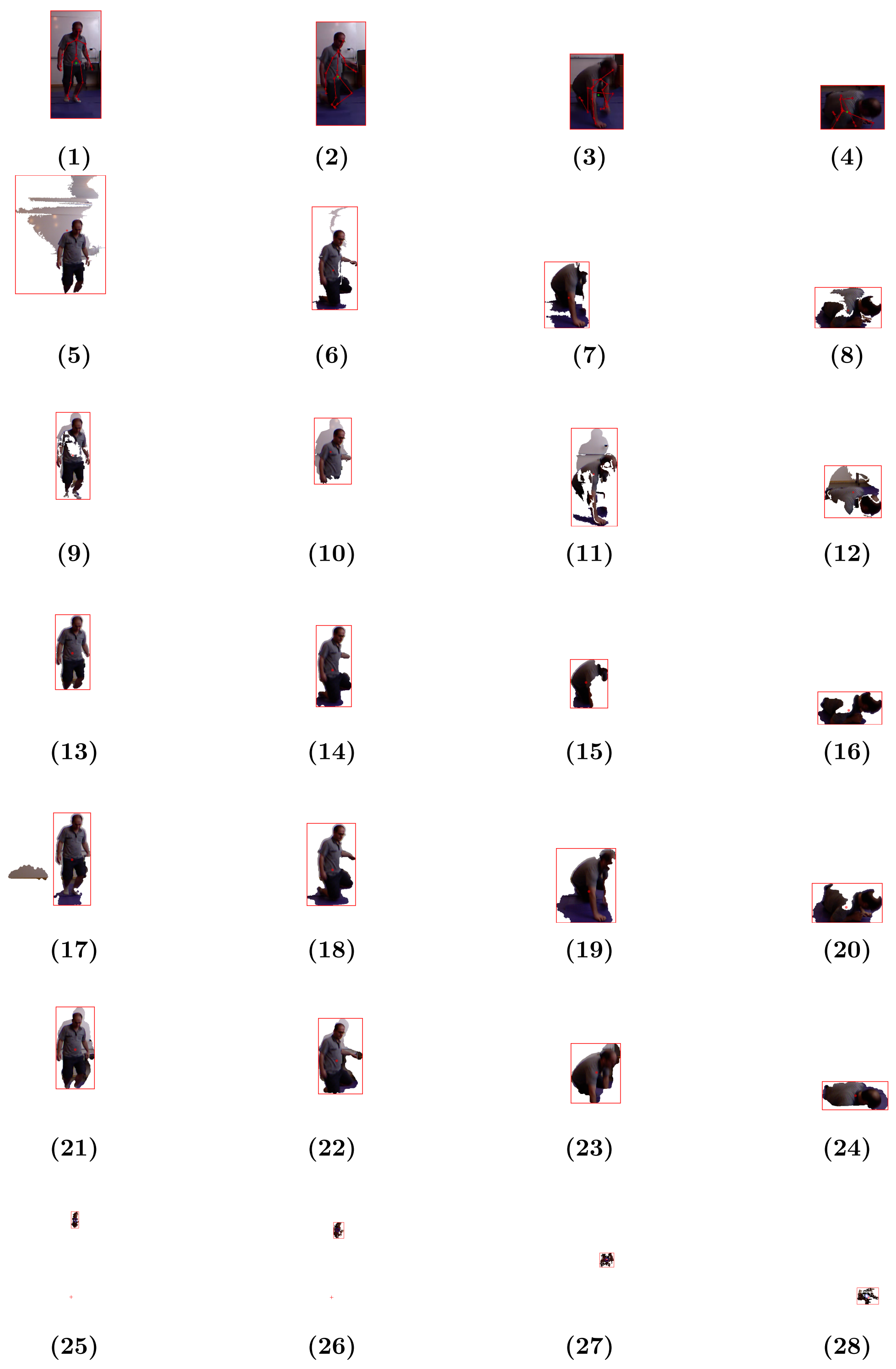


Figure 4.2: Different methodologies for determining the foreground in a video stream.

Table 4.1: Performance of the implemented detection algorithms. Run time results are average values (of approximately 300–400 frames) calculated by `tic/toc` function in MATLAB. Memory consumption was watched via the resource monitoring tool in Windows-systems. The tests were conducted on a machine with a Intel core i5-4200M 2.50 GHz processor, 8 GB of RAM clocked at 800MHz, running MATLAB R2014b on Windows 8.1.

Algorithm	**Results**	**Run Time**	**Memory Usage**
	[frame no. in Fig 4.2]	[s/frame]	peak/standby [MB]
Skeleton Color Threshold	1–4	0.013	610/580
Eigenbackground Static	5–8	0.042	809/803
Eigenbackground Dynamic	9–12	0.055	825/801
Manual Background Subtraction	13–16	0.078	1206/655
Gaussian Mixture Model	17–20	0.067	1270/678
Background subtraction via Depth	21–24	0.110	1223/652
HSV Subtraction	25–28	0.780	1971/1458

4.2 Features

Results regarding features will be given with respect to their individual classification rates as well as average measures on how big of a "difference" a feature produces for the different activities. two-dimensional (2D) features will be illustrated as scatter-plots in order to visualize this "difference", which in turn is interpreted as the relative difference of cluster centers or the average distance from the origin for different activities. Average histogram plots, based on various optical flow measures, is also given in order to determine the shape of such distributions. Table 4.2 displays the top performing individual features with a specific pre-processing method and SVM-type. Feature notation are

f_{CV} – Centroid velocity (see Section 3.2.1)

f_{HC} – Head coordinate relative to the center of mass for the body (see Section 3.2.1)

f_{HOG} – HOG difference feature (see Section 3.2.2)

f_{VAR} – Variance histogram of optical flow (see Section 3.2.3)

f_{ANG} – Optical flow histogram based on flow angle (see Section 3.2.3)

f_{MAG} – Optical flow histogram based on magnitude of flow (see Section 3.2.3)

f_{E} – Optical flow histogram based on temporal mean (see Section 3.2.3)

f_{EIG} – Optical flow feature based on PCA (see Section 3.2.3).

and the utilized pre-processors are

GMM – Gaussian Mixture Model (see Section 3.1.5)

Depth – Foreground detection using a depth-map (see Section 3.1.2)

EMT – Extended Maxima Transform (see Section 3.1.6)

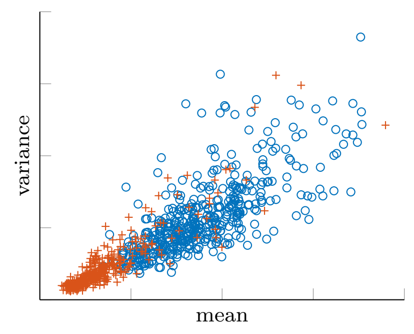
k-**HS** – k-binned optical flow histogram based on Horn-Schunk (see Section 3.2.3)

k-**LK** – k-binned optical flow histogram based on Lucas-Kanade (see Section 3.2.3)

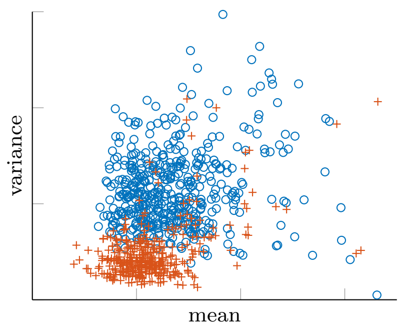
Table 4.2: Estimated CCR of individual features for various setups of pre-processing and classification models.

Feature	**Pre-proc.**	**Norm.**	**SVM**	**Kernel**	**Weight**	**CV** (100-fold)
f_{CV}	GMM	no	C	RBF	1	92,04%
f_{HC}	GMM	yes	C	Linear	100	91,90%
f_{HOG}	GMM	no	C	RBF	1	91,90%
f_{HC}	GMM	yes	C	RBF	1	91,78%
f_{HOG}	GMM	no	C	Linear	1	91,78%
f_{HOG}	GMM	yes	ν	RBF	0.35	91,76%
f_{CV}	GMM	yes	C	Linear	100	91,64%
f_{CV}	GMM	yes	ν	RBF	0.35	91,51%
f_{VAR}	30HS	n/a	C	Linear	10	90,60%
f_{VAR}	30HS	n/a	ν	RBF	0.25	90,47%
f_{VAR}	30HS	n/a	ν	RBF	1000	90,43%
f_{VAR}	30HS	n/a	C	RBF	1000	90,34%
f_{CV}	Depth	yes	ν	RBF	0.3	90,01%
f_{HC}	GMM	yes	ν	RBF	0.35	89,95%
f_{CV}	Depth	yes	C	Linear	1	88,96%
f_{CV}	Depth	yes	C	RBF	1	88,43%
f_{VAR}	9HS	n/a	C	RBF	10	86,55%
f_{VAR}	9HS	n/a	C	Linear	1	86,55%
f_{CV}	EMT	yes	C	Linear	100	86,11%
f_{CV}	EMT	yes	C	RBF	1000	85,60%
f_{HOG}	Depth	yes	ν	RBF	0.39	84,60%
f_{HOG}	Depth	no	C	Linear	1	84,30%
f_{HOG}	Depth	no	C	RBF	1	84,10%
f_{EIG}	30HS	n/a	ν	RBF	100	81,72%
f_{EIG}	30HS	n/a	ν	RBF	0.43	81,59%
f_E	30HS	n/a	C	RBF	1000	78,99%

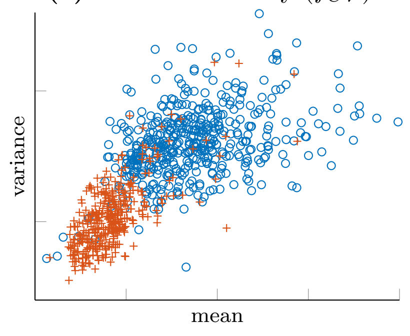
As mentioned earlier, GMM, Depth and EMT are all variations of the same detection method (background-subtraction), whereas HS and LK are variations of optical flow. Optical flow histograms of bin size $k = 9$ and $k = 30$ was used. The SVM model is either C or ν (nu) [20] with a linear or radial basis function (RBF) kernel. The weights vary quite a lot, which stems from how they were determined. A small trial-and-error process was utilized in order to acquire as high CV-results as possible. Most weights were decided on a power of 10 level, although some were even more fine-tuned.



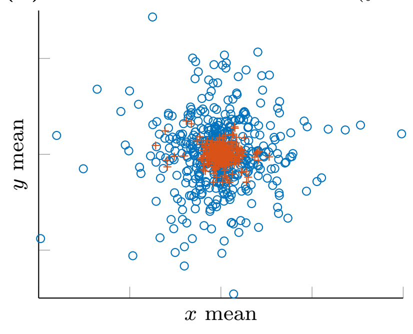
(a) Centroid velocity (f_{CV})



(b) Head-to-centroid relation (f_{HC})



(c) HOG difference (f_{HOG})



(d) PCA of avg optical flow (f_{EIG})

Figure 4.3: Scatter plots of individual features. Each point corresponds to a feature vector of an activity, i.e. one point for each video where a plus (+) denotes a person lying down and a circle (o) denotes a person falling down. Each plot contains 383 lie sequences and 473 fall sequences.

As can be seen in Figure 4.3, each feature plot (4.3a–4.3d) shows clustered regions for each activity. A vector $\bar{f}$ representing the average length or the "center of mass" of a cluster is used as a measure for how "well" a certain feature can distinguish one activity from the other. The larger the difference, the easier it would be to classify an instance as one class or the other.

The average centroid velocity vector $\bar{f}_{CV}$ was, with respect to both mean and variance approximately 2.5 times larger for the average fall sequence compared to the corresponding lie sequence as can be seen in Equation 4.1.

$$\bar{f}_{CV} = [\,E[f_{CV_x}],\, E[f_{CV_y}]\,] = \begin{cases} [\,0.9105,\, 0.2364\,] & \text{(fall)} \\ [\,0.3593,\, 0.0917\,] & \text{(lie)} \end{cases} \tag{4.1}$$

The average head-to-centroid distance $\bar{f}_{HC}$ had relatively even mean values but differed considerably with respect to the variance value. A fall sequence had an approximately 2.4 times larger variance than an average lie, as seen in Equation 4.2.

$$\bar{f}_{HC} = [\,E[f_{HC_x}],\, E[f_{HC_y}]\,] = \begin{cases} [\,0.1371,\, 0.1166\,] & \text{(fall)} \\ [\,0.1134,\, 0.0487\,] & \text{(lie)} \end{cases} \tag{4.2}$$

$\bar{f}_{HOG}$, the average HOG-vector, showed characteristics similar to that of the centroid velocity feature. An average fall had a mean value approximately 2 times greater than a lie while the difference in variance was approximately 1.5 as shown in Equation 4.3.

$$\bar{f}_{HOG} = [\,E[f_{HOG_x}],\, E[f_{HOG_y}]\,] = \begin{cases} [\,0.3489,\, 0.8030\,] & \text{(fall)} \\ [\,0.1675,\, 0.5175\,] & \text{(lie)} \end{cases} \tag{4.3}$$

The PCA of optical flow feature $\bar{f}_{EIG}$ showed different characteristics than that of previous features mentioned. Instead of looking at the center of mass (which essentially is the same for both falling down and lying down), the average distance to the origin was used as a measure of difference. This difference is given in Equation 4.4 as

$$\bar{f}_{EIG} = \sqrt{f_{EIG_x}^2 + f_{EIG_y}^2} = \begin{cases} 0.9200 & \text{(fall)} \\ 0.2316 & \text{(lie).} \end{cases} \tag{4.4}$$

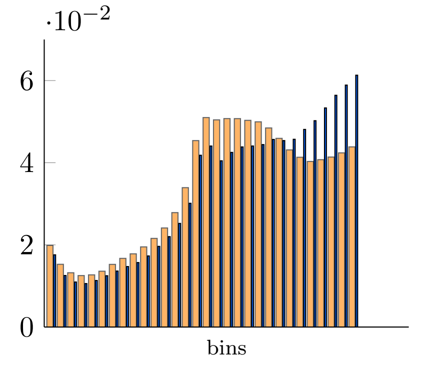
Figure 4.4 illustrates the average appearance of histogram features based on optical flow. Utilizing 100-fold CV, the highest CCR was obtained from the variance histogram f_{VAR} with approximately 90,6% correctly classified falls. Histograms based on temporal mean values, f_E, reached a 79% CCR while angle f_{ANG} and magnitude f_{MAG} based histograms resulted in a CCR of 67,5% and 61,2% respectively.

4.3 Classification

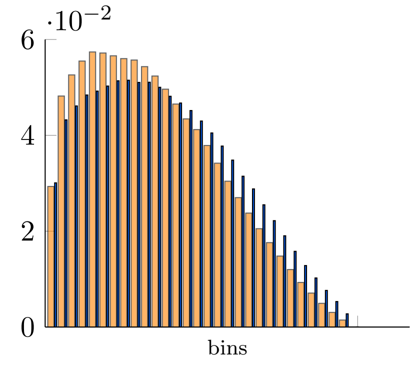
In this section we will state the results obtained from the final classifier as well as give its time-performance and further classification rates such as the FNR and FPR.

4.3.1 Cross-validation results

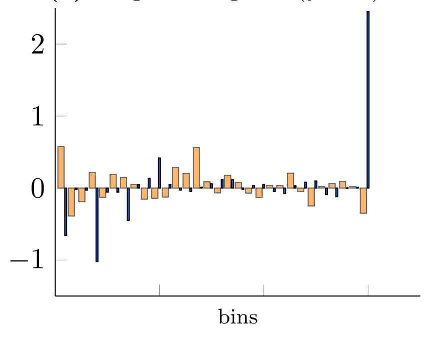
When the features had been fused together, normalized using the two different methods and the correct parameters for each SVM had been found via the exhaustive search, Table (4.3) depicting the performance could be constructed. Each SVM in Table 4.3 was tested using a 100-fold cross validation on the 383 falls and lies. The parameters used to obtain the results as well as what type of kernel is also presented.



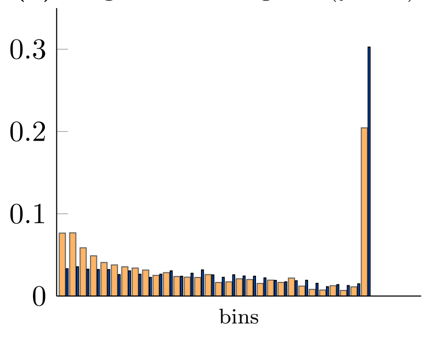
(a) Angle histogram (f_{ANG})



(b) Magnitude Histogram (f_{MAG})



(c) Mean histogram (f_E)



(d) Variance histogram (f_{VAR})

Figure 4.4: Average histograms of optical flow based on angle (f_{ANG}), magnitude (f_{MAG}), mean (f_E) and variance (f_{VAR}). Wide bars depict the average height of each bin for 383 lying down sequences and thin bars (overlaid histogram) depicts the corresponding average values for 473 fall sequences.

Table 4.3: 100-Fold Cross-Validation of 4 different SVM:s

Kernel	**c**	**g**	**Scaled/normalized**	**Run Accuracy (%)**
Linear	1.6	n/a	Linear scaling	93.73
RBF	0.1	0.6	Linear scaling	93.73
Linear	0.3	n/a	Normalized	92.95
RBF	1.85	0.005	Normalized	93.34

From Table 4.3 it is clear that certain ”combinations” of the SVM:s parameters and the used normalization methods results in different values on the accuracy of the classifier.

4.3.2 Run-time Estimation

When timing the final classifier on some of the videos, the data presented in Table 4.4 was obtained. An estimated run-time t for the classification process is then $t = t_{avg}/f_{avg} =$

16.03/26.33 ≈ 0.6s/frame, where t_{avg} and f_{avg} are the average computation time and number of frames given in Table 4.4.

Table 4.4: Time-consumption of the classification algorithm

Activity	**# Frames** (st)	**Compt. Time** (s)
Fall1	9	5.52
Fall2	12	7.92
Fall3	11	8.45
Lie1	61	36.1
Lie2	34	20.1
Lie3	31	18.1
Avg	26.33	16.03

4.3.3 Further Classification Rates

Table 4.5: 6 tables of confusion used for estimation of FNR and FPR. Average values represents classification rates estimated from a set of 20 confusion tables (see all estimates in Table 4.6 and 4.7.

	Acc Fall	**Acc Lie**	**FNR**	**MCR**	**CCR**	**FPR**
Pred Fall	159	11	6,47%	9,24%	90,76%	16,18%
Pred Lie	11	57				
Pred Fall	159	12	4,22%	7,60%	92,40%	14,29%
Pred lie	7	72				
Pred Fall	166	6	2,92%	4,37%	95,63%	7,41%
Pred Lie	5	75				
Pred Fall	162	3	8,47%	7,23%	92,77%	4,17%
Pred lie	15	69				
Pred Fall	153	5	8,93%	8,16%	91,84%	6,49%
Pred Lie	15	72				
Pred Fall	147	5	5,16%	5,39%	94,61%	5,81%
Pred Lie	8	81				
Average			5,79%	6,93%	93,07%	9,43%

Table 4.5 gives 6 out of 20 of the estimated results for FNR, as described in section 3.3.2 (more detailed results can be seen in Table 4.6 and 4.7). The average values, at the bottom of the table, are computed from the complete set of the 20 cases. Additionally, note that the table is not a full cross-validation computation. The input for the CM are

Acc Fall (actual fall), Acc Lie (actual lie), Pred Fall (classified to be fall) and Pred Lie (classified to be lie) where fall and lie are activity 1 and activity 2 respectively. FNR is therefore the rate at which activity 1 is classified as activity 2, similarly FPR is the rate at which activity 2 is classified activity 1.

Table 4.6: Confusion tables 1–10 for FPR/FNR estimation.

	Acc Fall	**Acc Lie**	**FNR**	**MCR**	**CCR**	**FPR**
Pred Fall	152	8	6,17%	7,17%	92,83%	8,99%
Pred Lie	10	81				
Pred Fall	157	10	4,85%	7,26%	92,74%	12,05%
Pred Lie	8	73				
Pred Fall	159	8	1,85%	4,82%	95,18%	12,12%
Pred Lie	3	58				
Pred Fall	164	9	4,09%	6,58%	93,42%	12,50%
Pred Lie	7	63				
Pred Fall	142	7	5,33%	6,73%	93,27%	9,59%
Pred Lie	8	66				
Pred Fall	174	9	3,87%	6,23%	93,77%	11,84%
Pred Lie	7	67				
Pred Fall	164	4	8,38%	7,76%	92,24%	6,06%
Pred Lie	15	62				
Pred Fall	159	11	6,47%	9,24%	90,76%	16,18%
Pred Lie	11	57				
Pred Fall	159	12	4,22%	7,60%	92,40%	14,29%
Pred Lie	7	72				
Pred Fall	166	6	2,92%	4,37%	95,63%	7,41%
Pred Lie	5	75				

Table 4.7: Confusion tables 11–20 for FPR/FNR estimation.

	Acc Fall	**Acc Lie**	**FNR**	**MCR**	**CCR**	**FPR**
Pred Fall	102	10	4,67%	8,06%	91,94%	12,66%
Pred Lie	5	69				
Pred Fall	158	13	5,95%	9,47%	90,53%	17,33%
Pred Lie	10	62				
Pred Fall	152	7	2,56%	4,89%	95,11%	10,14%
Pred Lie	4	62				
Pred Fall	152	9	3,80%	6,38%	93,62%	11,69%
Pred Lie	6	68				
Pred Fall	156	11	1,89%	5,79%	94,21%	13,25%
Pred Lie	3	72				
Pred Fall	154	8	4,94%	6,61%	93,39%	10,00%
Pred Lie	8	72				
Pred Fall	152	5	5,59%	5,81%	94,19%	6,25%
Pred Lie	9	75				
Pred Fall	162	3	8,47%	7,23%	92,77%	4,17%
Pred Lie	15	69				
Pred Fall	153	5	8,93%	8,16%	91,84%	6,49%
Pred Lie	15	72				
Pred Fall	147	5	5,16%	5,39%	94,61%	5,81%
Pred Lie	8	81				

4.4 Developed Software

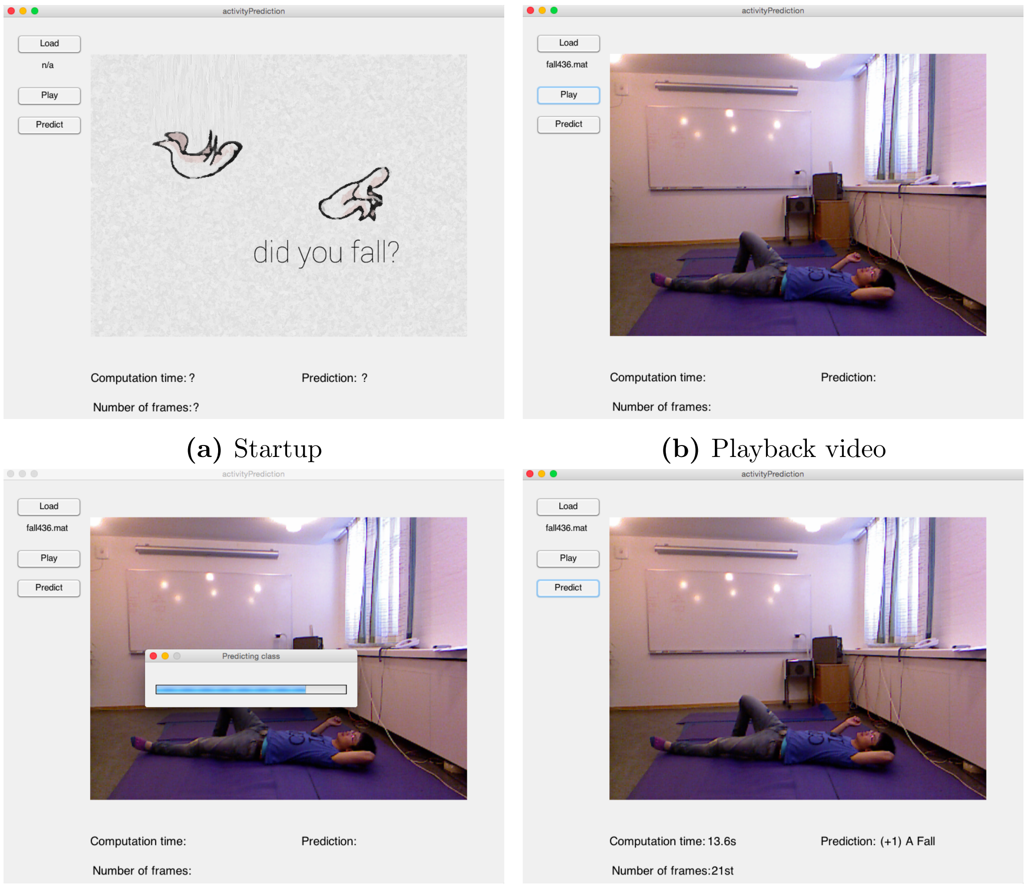
This section gives an overview of the functionality and interfaces of the developed software used for activity classification.

A wrapper function for the classification process, `didYouFall(`V`)`[1], was developed that initially calls the GMM and head-to-centroid pre-processing models, generating the ROI and head/centroid coordinates for each frame $\in V$. Then calculates the velocity, head-to-centroid, HOG, PCA- and variance of optical flow features. These features are then normalized and fused as described in section 3.2 and finally classified by a SVM model described in Table 4.3. The output of this function is thus the predicted result, i.e. 1 if the video contains a fall and 0 otherwise.

4.4.1 Graphical User Interface Design using Matlab

In order to get a simple way of using the `didYouFall(.)`-function and introducing additional functionality, a graphical user interface was implemented using Matlab. The different attributes of this interface can be seen in Figure 4.5. The core functionality of the GUI is essentially the classification process (a call to the `didYouFall(.)`-function) but, in addition, the GUI enables the user to load and playback a video prior to or after the classification process. This functionality was implemented to give the user an easy way of confirming the predicted results.

[1]V is an avi, mp4 or mat video file.



n/a

Computation time: ? Prediction: ?

Number of frames: ?

fall436.mat

Computation time: Prediction:

Number of frames:

fall436.mat

Computation time: Prediction:

Number of frames:

fall436.mat

Computation time: 13.6s Prediction: (+1) A Fall

Number of frames: 21st

(a) Startup **(b)** Playback video **(c)** Classification **(d)** Output results

Figure 4.5: Overview of the graphical user interface illustrating different views.

Chapter 5

Discussion

In this chapter we will discuss the three main components of the project and their respective sub-components, Detection, Features and Classification. We will mainly focus on our implementation of the algorithms and their final results. In addition we will also discuss possible ideas of improvement as well as other areas that we took special interest in.

5.1 Image Pre-processing

In this section we will focus on the procedures used to detect the foreground (the human) in order to generate the binary mask and the ROI.

5.1.1 Thresholding on Color

A few main issues were encountered when motion detection was conducted by color thresholding of the Kinect skeleton. The first issue was that the projection of the skeleton, i.e. the joint coordinates, did not map accurately onto to the human for certain orientations/poses. This problem mainly occurred when the person was lying on the floor. Then the skeleton would often cluster to some part of the body, e.g. the legs. This was probably due to the significant difference in shape between a person standing up and one lying down. The entire set of joints was not always ‘’visible” when a person was lying down and the Kinect seems to have problems estimating the location of the ‘’non-visible” joints in such a setting. Other faulty mappings also occurred, e.g. a joint could sometimes be mapped to a random coordinate of the image. The later issue was assumed not to pose much of a problem since erroneous estimations of the ROI only would appear in a few images. The fundamental flaw of this approach was, however, that the images mask produced did not capture the whole person, only a ‘’stick man” or skeleton representation of the pose. A solution to this problem was to produce a bounding box (BB) of the mask and extract a rectangular ROI. Further difficulties led to additional processing. One such difficulty was that the ”head” coordinate was mapped to the centre of the head. In order for the BB to fit the entire person, it therefore had to be scaled. The scaling was also important in order to solve the first issue, i.e. to fit the BB to the person when the skeleton clustered together. The enlarged ROI was a significant drawback of this method because the amount of background data included got significantly larger as apposed to background subtraction approaches. Discernible advantages was, on the other hand, the invariance to illumination change and the ability to detect a person even if no motion occurred. These were issues that showed to be critical problems in some of the other approaches.

5.1.2 Manual Background Subtraction

The method of manually subtracting the background from every frame was early adopted. The process was deemed feasible since the camera capturing the videos of our dataset was fixed at the same place. This meant that only one frame containing the background had to be extracted which could then be used multiple times. Also, because of its simplicity the manual background subtraction was a good starting point for the foreground detection before moving on to the more complex strategies. The initial results, in the form of binary masks and the ROI, from the manual background subtraction showed, tough crude, that the method was worth pursuing. At first the generated binary mask contained much noise and other unimportant debris. Unfortunately some of the persons in the videos also wore clothing with color similar to parts of the background. Whenever one such video was processed, large parts of the person would be lost to the background. In some cases this meant that the ROI got split up in smaller ROI:s containing the body parts that did differ from the background. In order to achieve a cleaner mask as well as getting a better ROI, some image enhancements were investigated. At first using a Laplace filter to sharpen the images before the background subtraction, was considered. The method was not pursued however, mainly because other pressing matters took precedence and after that the idea was forgotten.

The first attempt in enhancing the image to prepare it for background subtraction was to enhance the colors by performing a decorrelation stretch. At first this improved the masks generated but since it did not contribute much when the images were converted to grayscale it was later dropped. In connection with using the grayscale image for background subtraction an interesting effect was produced. The subtracted image still contained some noise and debris but now a human-like shape could vaguely be seen. To make use of this special frame, denoted “speccframe”, the contrast was increased. This resulted in an image with decent properties for thresholding. Under some circumstances the speccframe was also quite noisy. To alleviate the effects of the noise a median filter was used. The parameters were manually tuned to suit our videos by testing. The use of a Gaussian blur was also tried out. Although it made the frame smoother and seemed to reduce the noise, it did not work properly later in the process in conjunction with the EMT, and was therefore removed. The final specc-frame sadly had one annoying disadvantage. It was still a bit susceptible to changes in illumination. Due to the nature of the videos these changes happened often. Because if this, the speccframe would become completely white and in turn resulted in a binary mask which basically covered the whole frame. It goes without saying that such a mask would have been useless. However, after continued tweaks and the choice of detecting method to GMM the specframe showed good results.

5.1.3 Use of Extended Maxima

By generating the specframe from each color channel individually, performing an EMT on each and finally logically OR them together created a mask with reduced susceptibility to changing lightning conditions. Unfortunately, this procedure tripled the computational load and was therefore considered a quite high price to pay for the improvement. Nevertheless, it was considered to useful to neglect and was therefore left in the code as an available option. Unfortunately the EMT had a tendency to make false positives on moving shadows e.g the shadow cast by the persons body. This would most often mean an enlargement of the blob and by extension of the ROI as well. Since this were most likely to happen when the person was just roaming around the room and not during the falls or lying-downs it was not considered to cause any major problems and was therefore not remedied. Over the course of the implementation the method was in some sense ”forgotten” and it did not receive as much tweaks as the other methods. It was perhaps because of this that the method struggled a little to keep on par with the others during the testing. At the final stage of the project the method was removed since it, as mentioned, did not manage to keep up with the others. If more work had been done on it it might have become more useful. If we had had the time it would have been interesting to return to it and make improvements.

5.1.4 Estimating a new Background

Due to the sensitivity to changes in illumination new possible methods of foreground detection were investigated. A method of estimating a new background every n number of frames was considered and a first attempt was also implemented and tested. If a new background with updated lightning conditions could be obtained much fewer errors would be able to propagate trough to the final mask and would therefore most likely have improved the results. The main problem encountered with extracting a new background was that a person could already be present. In that case extracting a new background would only have made matters worse. In order to avoid this, the ROI on the previous frame was enlarged a bit and assumed to still contain the person. This part was then cut out from the frame and the remains was merged with the previous background data as an average. In theory the idea seemed quite good but in practice the results were mediocre to poor. On some videos the technique worked but on some, extremely poor results were produced. Due to this inconsistency in performance coupled together with the high risk of the accumulation of small errors over time in the averaging, the method was not pursued further. If more time was available it might have evolved to something better.

5.1.5 Gaussian Mixture Model

Another method of distinguishing what parts of the video belonged to the foreground was by using a GMM to model the background. The idea to use GMM first came when a mathworks webinar on the computer vision system toolbox briefly described its use in detecting moving cars on a street view [8]. It was then realized that the GMM should be suitable for detecting moving humans as well. The first test were done by passing the original frames right through the GMM in order to establish a model of the background. The initial results showed much promise so further investigations were performed. By conducting further literature studies as well as running more tests, better and better results were achieved. The produced binary masks generated via the GMM had some areas in them which did not contain any interesting information. A false detection of the foreground so to speak. This error was probably due to faulty initial parameters and an unsuitable number of training frames. Over time, some problems as for example, how many Gaussians to use or how many training frames the model should be based upon was found by a trial and error process. There was however one flaw with the GMM. Just as the other methods, it too suffered from a minor, but nevertheless present, susceptibility to illumination changes. Since there only were a few frames of background which the model could be based on, typically around five, the model was not as robust as we would have hoped. An interesting thing to try out would have been to train the model with at least 50 frames, with significant changes in illumination, and see if this would have decreased the susceptibility. Unfortunately no such data were available and therefore the first few frames had to suffice.

Eventually the idea came to let the GMM operate on the specframe. Since the specframe had very little intensity difference, (it was essentially black or white) only two Gaussians were needed to create a reliable model. We considered that since there was much less varied data in the background of the specframe, the modeling would be simpler and produce a more precise mask. The tests showed good results as well as good performance. It also had the big advantage of not being as susceptible to the aforementioned illumination changes. Consequently, it was quickly implemented and only received minor tweaks to its parameters from there on out.

During the final stages of the project we realized that since we only made a classification of video-clips containing one activity, the few empty background frames for the GMM would be unavailable. Not wanting to impose the restriction that every classification would need an empty background setting, we tried to tweak the GMM to only train on the first specframe that it received. After a while we managed to get satisfying results using only the first image as training. After having generated all of the features for the 500 video data set we realized how oddly the GMM actually behaved. Since each video clip now immediately contained a person the first specframe would also contain some representation of this in the form of a white area where the person should be. Passing this specframe into the GMM as the only training frame should have generated a faulty ROI due to the training frame containing white parts (a proper background should have been completely black in this case). Afraid that all generated features were faulty because of this error we ran around 30 tests on the GMM but to our astonishment found that not a single one generated a bad ROI. We were even more puzzled when the GMM generated extremely poor ROI:s when two or three frames were used as training. Being in the final stages of the project we did not have more time to investigate this behaviour. Perhaps it was due to how the GMM was programmed in Matlab or how the training image was preprocessed before being passed into the GMM. It could also be that the GMM plainly ignores the training frame if only one (1) is present, effectively emulating passing a black frame into it. Still, it was somewhat confusing.

5.1.6 Depth-map

As was already mentioned, the sudden changes in luminosity of the videos wrought havoc on nearly all of the implemented methods. Looking for ways that did not have this drawback, the realization came that by using the depth of the image, the interference of uneven illumination would be nullified. Since the objects in the video, other than the human, naturally remained at the same distance it should have been possible to perform a more exact manual background subtraction with the depth instead of with the RGB data. The frames obtained through this method should only have had non-zero pixels at the place of the person. This could then easily have been thresholded in order to obtain a mask of the ROI. Disappointingly, the depth-data captured by the Kinect only covered the interval [0,7-7]m from the camera. This resulted in depth-data that contained mostly zeroes. Therefore it was not to much use. Some workarounds to the conundrum was tried, however only one produced satisfying results. If complete depth data had been available the method would most probably have gotten great results but in this case the data was deemed too poor to motivate continued investigation. As such, all further work with the method was halted. The one technique that did produce some good results focused only on parts of the depth interval at a time and later reconstructed the whole image via logical OR operations. It sill suffered from most of the same problems as the original data did though. Whenever the person moved to far away or too close to the camera, the tracking capability was lost. During the final stages of the project we came to the realization that the loss of the person was not as big a problem as we had thought. Since we had made the restriction to only operate on pre-cut videos containing one activity and the random loss of the person only happened when they moved around to the back of the set and never during the actual falling or lying sequence, the masks produced by the depth-subtraction method should have been quite satisfactory. Unfortunately the realization came well after the final code had been packed and could therefore not be tested. Nevertheless the method could have created even better results and would have been very interesting indeed to investigate.

5.1.7 Eigenbackground

The static EB model shown to be highly sensitive to illumination change. The attempt of creating a set of images (from a single image) with varying intensities, did occasionally show some improvements. The difficulty was however, that changes in illumination often were caused by natural light, e.g. increasing/decreasing amount of sunlight through a window. The effect was thus that, illumination change occurred unevenly in the image. The uneven change did not match well with the set of images used to construct the model because this set had gradual, but even, change in intensity. As a result, image patches with uneven intensity got interpreted as foreground objects. Within a contained setting, e.g. a room without windows, where illumination change (by artificial light) could be assumed to change evenly, this method would have been advantageous in the sense that a foreign object would be detected even if it was still. The dynamic approach also had its disadvantages. Depending on the number of (prior) images N used to compute the model, different problems were observed. If $N = 1$ the model basically represent a straightforward background subtraction and therefore suffered from such problems as mentioned before in (5.1.2). If $N \geq 2$ the ROI would be represented as the entire trajectory of the object that occurred during those frames, i.e. a ”lag”. To capture the interesting region in a fall, the dynamic approach therefore had the disadvantage of capturing much of the trajectory (depending on the number of frames used for building the model).

5.1.8 Subtraction in HSV

In contrast to what sub-figures 25–28 in Figure 4.2 displays, background subtraction in the saturation-channel of HSV color space did not excel at continuously masking the head. The algorithm worked but did so inconsistently. It does seem to work fine during activity 1 and 2 but not when the human is performing ”random” activities. The cause of this unreliable behaviour requires a review of the algorithm. The human skin continuously had distinguishably high saturation in the test data videos. The goal with the algorithm was therefore to locate and separate the head, but the face had approximately the same saturation as the rest of the human skin visible, e.g. hands. This observation lead to the conclusion the connected components, blobs, needed to be filtered. The blobs were filtered for a certain range of size acquired by trial and error. A threshold was set up in the hue-channel to reduce the risk of including some non-human skin parts. The skin tone seemed to be orange to red, i.e warm colors. This meant that cold colors, i.e. green and blue were filtered away. This filter imposed an issue when a human was lying with the head towards the camera. The only ”skin” that could be observed was the scalp, which more often than not was covered by hair. Hair which both could have a low saturation and a hue that could be classified as cold. The observed person ran the risk of being ”decapitated”. The problem which occurred was that the hue filter either made the extraction more precise or completely removed the data obtained from the subtraction. What this meant was that at certain frames no data would be extracted. To nullify this fact under these very specific conditions the output could be set to contain two sets of data, computation of the biggest blob’s location before and after the filtering of cold hue. The two sets could be anticipated to include the coordinates for a rough center point of the torso (with a lot of noise) and the center point of the head (if it existed). However, to compute the biggest blob’s center point twice extended the run time of the algorithm quite significantly (approaching 1 second per frame). The ”safe”, extra coordinates of the torso were excluded due to the occasions where they were needed were few and far between. In addition to the risk of not having values, there was a risk of having the wrong values. If a highly saturated warm-colored object with an appropriate size would be present, e.g. a pattern on a t-shirt, this object's coordinates could be the values extracted instead of the real head's coordinates.

An desired result of an improvement for this algorithm would be to locate the head more continuously and reliably. Several options are available to achieve such a goal. Fine tune the values of the filters is one obvious step, another one is implementing an object recognition algorithm, may be to be used on the blobs. Another potential improvement would be to run the whole HSV-subtraction on the ROI instead of the original image. This would probably result in a lower run-time, since less data would be used in the calculation. It can be noted that ambient changes in illumination did not pose much of an impediment to this method. A probable explanation for this was the change of illumination mainly affected the brightness-channel which was left unused in the algorithm.

5.1.9 Morphological Operations

The masks produced from the earlier mentioned methods were all quite rough around the edges and also contained small specs of erroneous positives. Much of the blobs present also had tiny holes in them. There were also some cases where the blob containing the human was split up in 2 or more separate, but adjacent, blobs. In other cases the limbs of the blob with the person was unreasonably thin. A remedy for these problems was to make use of some morphological transformations. By using a function to fill in the small holes, better blobs were obtained. By manually looking through the masks it was deduced that the blob containing a person never shrunk to a size less than a couple of thousands of pixels. Therefore by removing any blobs smaller than around a fifth of this, a cleaner mask was obtained without losing any data from the larger ones. The mask was nevertheless still too spiky and sometimes too “thin” to be used effectively. Using dilation and erosion, the shape of the blobs were smoothed out and to a certain extent enlarged as well. Because of the enlargement of the blobs via dilation, the connectivity problem was solved too. The decision of which type of SE to use and its size was determined through manual testing to see which ones gave the most satisfying results. Due to the manual tweaking, improved results could most likely have been achieved but was nevertheless a time consuming process that was unnecessary to continue working on after acceptable results had been reached.

Since the above mentioned methods discussed throughout this section had different strengths and weaknesses and excelled at detecting different parts of the human, it seemed, at the time, wasteful not to use them all. Unfortunately no better way of combining their strengths than logically OR them together was found. In some cases this gave better results than using any one of them on its own. The results it gave to add them together did motivate the decision to at least implement the option of choosing “any and all of the methods” into the intermediate detection code. Using ”any and all of the methods” did however increase the computational load drastically and resulted in elongated running times. Although we did not have any actual restriction on how fast the program had to run or resource consumption, we felt that waiting three or four times as long for a small gain in accuracy was not worth it and the option was dropped in the final version of the software.

5.2 Features

In this section we discuss the various features that we used in the final classifier. We also mention the ones that did not get implemented. On both of the groups we try to reason what their respective strengths and weaknesses are and perhaps how they could be improved upon.

5.2.1 Centroid Velocity

The centroid velocity feature is perhaps the most simple one of the features used. At first the centroid was supposed to track the movements on the horizontal line as well as on the vertical, but we soon realized that a quick movement to the right or left of the image could generate a false positive. Seeking a way to prevent this from happening, the feature was changed to only consider vertical movement. This turned out very well in our controlled setting but in reality, the centroid feature could still generate false positives. For example, it is very plausible that a quick jump could generate a centroid feature value very similar to the one that would have been generated if it was a fall. Therefore one mustn't take the feature at face value but instead apply a more critical mindset in order to make qualified determinations on the nature of the activity that generated them.

Due to the computation of the magnitude of the difference between two centroids, the feature experienced a loss of information, That is, not all data the feature contained could be properly utilized. For example, the calculation of the feature resulted in a loss of the direction of the movement which meant that that information could not be used for classification anymore. In fact, the final centroid feature did not contain any information at all on the direction of the movement since it did not even support, for example, negative values for moving up. The feature generation was also tailored to suit the specific frame-rate of 20FPS that we had in our training data. Therefore a down-sample had to be made on any video with higher FPS (see Section 3.3.3). The feature generator for the centroid velocity also had the slight problem of generating different values for the velocity depending on how far away in the image the centroid was. The closer the centroid was to the camera the higher its value would be. This problem could have been remedied by the use of the depth-data but was not implemented due to time-restrictions.

The feature did have some good qualities as well. Thanks to its simplicity it took very little computational effort to create it which resulted in low performance costs and quick loading time. It is also quite intuitive to think about the movement of the center of mass as a descriptor of how a person (or object) is moving which makes it easy to grasp the concept of the feature. Furthermore, the feature also had some areas where improvements could be made, which perhaps could have increased its accuracy. For example, by instead of taking the mean of the vector, create a histogram, more information of the movement could have been retained and used by the SVM. Another possibility could have been to calculate the acceleration of the centroid instead of its velocity. The acceleration of a fall would most likely be close to the gravitational constant g and easy to compare to. Sadly, actually computing the centroid acceleration in m/s^2 would have been hard since $pixels/s^2$ translates to different values for the acceleration depending on how far away the person or object is in the image (as stated in the previous paragraph). It would have been possible to do (since we had access to the depth-data via the Kinect sensor) but we deemed it to time-consuming to implement. Because of this, the two mentioned additions to the feature were dropped but it would nevertheless have been interesting to investigate them. Lastly, as already mentioned, in our specific setting the feature did show good performance and was therefore kept in the final version of the classifier.

5.2.2 Head to Centroid Relation Feature

The head feature, or more specifically the relation between the head and the center of the body feature, is quite simple. For each frame the euclidean distance between the head and the body's center of mass is calculated. Since the head-trajectory pre-processing is a little prone to report ”nothing” (section 4.1) the head-feature assigned a default value whenever that happened. These values were the X- and Y-coordinates of the middle of the image. This created a situation where the person could be at the outskirts of the image in one frame but in next frame the data from the head-trajectory would be missing. The head-feature would assign the ”missing” values to the middle of the image. The result of such a situation would therefor be a abnormal long distance between what was assumed to be the head and body. Since this happened somewhat at random for both activities the feature did not get spoiled as can be seen in Table 4.2 and in the Figure 4.3b as well. In the table the rightmost column shows the performance, and the head feature (with the notation f_{HC}) is a top scorer. In the figure the two activities can be seen to be decently distinguishable along the variance-axis.

If the head-feature had handled missing data differently, for example by using the last recorded value before the missing data, another problem would arise. If a person moves in and out of the image, the coordinates would remain at the edge of the image where the person left the screen. However, this scenario is not relevant due to the restrictions imposed to the data, where the video snippets are assumed to completely contain a human performing an activity. This leaves us with four scenarios.

- If both the head- and body-data are present no adjustments are needed.
- If only the head-data is missing i can be assumed that there still is a body but the pre-processor for the head coordinates accidentally ”decapitated” the body (section 5.1.8) and head's coordinates would be replaced by the old or default values.
- If only the body data is missing it can be assumed that the head trajectory pre-processor has erred and detected a head in an ”empty” frame. This could possibly

be resolved by checking if a ROI is present for the whole body or not. Another solution could be to perform the head trajectory on the ROI instead of the whole image, as mentioned in section 5.1.8.

- If both head- and body-data are missing it can be assumed that there is no human in the image.

As mentioned earlier default values is one way to replace missing data. These default value would have to be set to something that minimize the interference with ”real” values. Zeroes is one possibility, minus one is another and the coordinates for the middle of the image (CMI) a third. In contrast to zeroes and minus one, the CMI acts as a cover up for missing data. Since it is in the middle of the image the majority of actions happens, CMI is a ”static estimation” where the human has a high likelihood of being. Since the replacement values only would be the values for one frame and the data delivered to the SVM is the mean and variance of a video snippet, it is hard to say how these different implementations would affect the classification accuracy.

The variance and the mean was implemented to reduce the number of data the SVM had to compute for every video snippet. That both the variance and the mean are used means that there is a possibility to distinguish certain scenarios. For example, if there is no data extracted from certain frames of a video snippet, the data are replaced with zeroes and the mean drops. If the very same video snippet generated some bigger values as well, the mean would not be much different that the mean of a video snippet generating strictly ”normal” values. So by looking only at the mean these two video snippets would not be distinguishable. However, there would be a difference in the variance and they would therefor be distinguishable.

5.2.3 HOG Features

The process of obtaining the feature from the HOG was quite similar to the process of obtaining the centroid and head feature. However, due to the nature of the HOG calculation itself it produced entirely different values. At first we tried to use the HOG in its raw form but the sheer size of the HOG vector prevented us from using it directly. Instead, the magnitude of the difference of to consecutive hogs was taken, reducing the size to a single value and losing some information in the process. Taking the statistical measures mentioned in section (3.2.2) reduced the size further to a one more fit for the SVM. Much like the case with the centroid, one could have opted to calculate a new histogram from the the intermediate HOG vector (Equation 3.5 section 3.2.2). However, due to the fact that our familiarity with the HOG was quite low, we chose not to pursue that method since the initial results with the two-tuple had proven to be quite satisfying right from the beginning. One thing we nevertheless did try was to check whether only computing the HOG on the contents inside the ROI was better than on the entire frame. In our case there was a big gain in performance as well as CRR when restricting the HOG to the ROI. One great thing about the HOG was that it did not depend as much on the speed of which tings were happening. Rather it seemed to in some sense actually "look" at the change in the image and responded accordingly. We drew this conclusion when we looked how the HOG performed on some videos with higher frame rate. Since it was quite unaffected by the frame rate in its accuracy it must be that it does not rely as much on the speed of which things are happening.

5.2.4 Optical Flow Features

Some variations of optical flow turned out to generate satisfying results. Among the different types of histogram features used, optical flow variance gave the highest CCR during tests. Variance of movement was expected to vary significantly between the activities under test, even though the type of motion was quite similar. At a fixed frame rate a fall must, by common sense, finish in fewer frames compared to a person lying down. For example, if a person in a standing pose falls to the ground, the body would approximately move with a constant acceleration (due to gravity), while if a person that intentionally lies down, surely would have a trajectory towards the ground with a lesser accelerated motion. Of course it is possible to lie down during a time frame equivalent to that of a fall but this would essentially mean that the person intentionally falls. If there are less motion between frames, the variance naturally gets smaller and since a lying down sequence has exactly these characteristics in comparison to a fall, the lying down histogram gets a greater amount of low variance values. That is, a larger part of the values gets binned in the left region of the histogram as seen in Figure 4.4.

The temporal mean histogram was, for the same reasons as variance, expected to produce noticeable differences when generated from different activities. The obtained results was not entirely as expected and the reason for this might stem from the calculation procedure. Unlike variance, which is a positive number based on the variety of flow, the mean values was simply calculated as the mean value of each flow vector, generating the average flow field. The flow vectors of both horizontal and vertical orientation was, as mentioned in Section 3.2.3, numeric values representing the magnitude of flow and was positive if the orientation of flow was down/right and negative if the orientation was up/left. This means that if the same amount of motion would occur towards the left as to the right or upwards and downwards during a video sequence, the temporal mean would be close to zero. A better approach would perhaps have been to look at the average magnitude of vertical/horizontal flow. If this method were to be adopted, this approach would still differ from the magnitude histogram f_{MAG}, which computes the 2-dimensional magnitude $(\bar{f}_x^2 + \bar{f}_y^2)^{1/2}$ but might represent similar information. The information contained in both f_E and f_{MAG} essentially are the same with the only difference that f_E bins horizontal and vertical flow separately based on their individual size while f_{MAG} bins the size of the actual (2D) flow vector.

Since the magnitude histogram also was based on the temporal mean, this feature might have been improved by utilizing the idea of calculating the temporal mean of flow vectors as a temporal mean of flow vector size.

Using the angle of flow for assigning a flow vector to a bin turned out to generate histograms with a relatively poor classification rate. This might be the case since both a fall and a person lying down exhibits similar characteristics in terms of the angle of motion. If instead diverse activities were to be compared this approach would probably generate significantly better results.

The non-histogram feature based on PCA of the temporal average flow turned out to be non-linearly separable in 2D space as can be seen in Figure 4.3d. This phenomena made it necessary to use the kernel trick as mentioned in Section 2.6 by utilizing a RBF kernel for classification. A CCR of 81,7% could then be reached. This feature is somewhat cryptic in the sense that it is not directly intuitive as what it represents. The choice of using the projection of the data on the first eigenvector was made in order to reduce the dimension as much as possible, keeping only the most significant part [22]. Since the dimension of an eigenvector v_e of an $N \times M$ flow field still has a quite large dimension ($N \times 1$) the mean value of such a vector $E[v_e]$ for both horizontal and vertical flow was, as mentioned in Section 3.2.3, used as the resulting feature. The final step of taking average values most likely destroys much information but it still seem to generate distinguishable results.

5.2.5 Other Features Specific to Human Classification

When extracting activity-sequences from the test data a certain pattern was distinguished in the very beginning of activity 2 (lie down). People with pants tended to bring their hands towards their knees, most likely to adjust the pants and prepare for the activity. In comparison while performing the early stages of activity 1 (fall), the hands were more or less randomly flailing. If our detection methods had supported knee-joint and hand trajectory, tracking the relation between hands and knees would have been an interesting feature to investigate.

Tracking the relation between different body parts in general would be interesting. The features would help specify if it indeed was a human that was detected and performed an action or if it for example just was a cat strolling by. For example if the hand trajectory algorithm, intended for humans, managed to extract the coordinates for the cat's left hind limb and head and a knee trajectory algorithm extracted the cat's tail. The feature above would probable differ substantially between a cat and a human. The features would therefore complement the motion detector to recognize certain aspects of human movement pattern and as a result probably increase the correct classification rate. However the trajectory algorithms would need to be of highest reliability when a human is the cause of the detected motion. To conclude the example, it would most likely not matter if hands were extracted from every ”screen of motion” as long as the ”actual hands” were extracted as soon as a human was on screen, due to as mentioned above it would be the relation between several human body parts and not the specific coordinates nor existence of such coordinates that would be the deciding factor.

5.2.6 Scaling

When using the scaled and normalized vectors we found that the linearly scaled ones gave the best results when passed into the SVM. When performing a 100 fold cross evaluation the CCR almost reached 94% (93.7337 to be exact). When using the other set we were only able to reach 93.21 % CCR. Since the same videos were used to generate the training with the only difference being the normalization of the features, and using the “optimal” RBF parameters for the SVM we drew the conclusion that in this particular case, a linear scaling was better than the traditional one. One thing to note however is that this was done with a fairly small data-set. It could very well be so that if the data-set was larger and contained more varying individuals falling, that the conventional normalization would have reached higher CCR. Also, the RBF kernel essentially produced the same results with the only difference that it was a bit wore at classifying the features individually. We also tried another way of reducing the size (values) of the feature vector, namely by creating a unit vector pointing in the same direction and train the SVM on that. Unfortunately this did not help at all since the feature vectors only differed in magnitude and not in direction between the falls and lies. Normalizing them in this way only made everything cluster together in the same spot, making the classes indistinguishable. Because of this, the method was switched out in favor the linear scaling.

5.2.7 Fusion at Feature vs. Decision Level

Initial experimentation on individual features gave varied results, i.e. at the individual level the accuracy in classifying activities varied from approximately 50% to values exceeding 90%. Features providing a higher CCR was then naturally the ones used in forthcoming tests. As mentioned earlier, centroid velocity, head-to-centroid distance and HOG-difference was three types of features that showed promising results (CCR 90+%). As for optical flow, histograms based on variance gave the highest accuracy followed by the PCA-of-flow-feature giving CCR in the lower 80:s (81+%). By merging (fusing) the five best performing feature vectors f_i, $i = 1,2,\ldots,5$ into a combined vector $f = [\, f_1, f_2, \ldots, f_5\,]$, even higher CCR was reached. The motivation of utilizing fusion at the feature level, as mentioned in Section 3.2.6 was that features specifically design to look at certain aspects of an activity might not give sufficient results. This is since there might be other activities showing similar characteristics with respect to that aspect.

Another approach would be to apply fusion at the decision level, that is, to classify an activity with each feature individually, in our case, making five separate classifications and fuse the outcomes. A theoretical model was developed but due to time limitations it was not implemented. An outline of the model is given here

- Train n classification models m_i, $i = 1,2,\ldots,n$ with CCR $\alpha_i \in [0,1]$ for each type of feature f_i
- Compute sum of decision value $\alpha = \sum_{i=1}^{n} \alpha_i$
- Classify an activity A with each m_i, giving n classification values $p_i \in 0$ or 1
- if $\frac{1}{\alpha} \sum_{i=1}^{n} \alpha_i p_i > 0.5$ then A is classified to belong to the positive class

Since each model m_i contributes with a value α_i, a model m_+ with greater $\alpha = \alpha_+$ than a model m_- with a smaller $\alpha = \alpha_-$ would have a greater contribution (weight) in deciding the final classification.

An interesting observation is that, if the number of features is less than three ($i < 3$), fusion at the feature level might work fine for any value α_i while fusion at the decision level only would consider the model with the largest α_i. That is, for $i = 2$ and $\alpha_1 > \alpha_2$,

$$p_i = \begin{cases} \{1,0\} \rightarrow \frac{\alpha_1}{\alpha_1+\alpha_2} > 0.5 = \text{positive} \\ \{1,1\} \rightarrow \frac{\alpha_1+\alpha_2}{\alpha_1+\alpha_2} = 1 = \text{positive} \\ \{0,1\} \rightarrow \frac{\alpha_2}{\alpha_1+\alpha_2} < 0.5 = \text{negative} \\ \{0,0\} \rightarrow \frac{0}{\alpha_1+\alpha_2} = 0 = \text{negative} \end{cases}$$

5.3 Classification

In this part we discuss the classifying model itself and also the calculated performance measures.

5.3.1 On the Choice of Kernel and Tuning of Parameters

The choice of which kernel to use in the SVM was relatively easy. Since our feature vector was quite long we decided to give the linear kernel a try first since it in our case resulted in faster computation. The initial results were quite bad, not much over 50%. Thinking that this was due to the linear mapping we decided to try the RBF as well. Unfortunately this did not help much either since we still just barely reached over 50%. Luckily we realized that if we normalized the vectors we might get better results. After the normalization the CCR jumped up to around 93% the linear kernel and close to 92% for the RBF. Since this result was reached using only the default values of the parameters in the SVM we figured that some increase in CCR perhaps could be gained if the parameters were correctly adjusted. Finding these parameters manually seemed quite time consuming so the process was automated using an exhaustive search. By saving the parameters that gave the best results we could be certain that a close to optimal choice had been made. The process was not without flaws however. Even though the process was automated a complete exhaustive search with 100-fold cross-validation would have taken around 24 hours to complete. In order to reduce the running time only a 25-fold was used. The value for the parameters generated this way was then manually entered into the SVM and then slightly tweaked in order to see if a better result were to be found in the close proximity. Using this method we could at least be certain that we found good values and only perhaps missing out on a tenth of a percent. In the end we found out that in our case the use of a C-SVM (cost parameter $c = 0.1$) with RBF-kernel (exponent factor $\gamma = 0.6$) on the linearly scaled feature data performed best.

5.3.2 Reliability of Estimated Classification Rates

The mean of 20 FNR and FPR values from the CM, 5.79% and 9.43% respectively, were concluded to be reasonable estimations. This was because the estimated CCR, 93.07%, very much resembled the CCR of the 100-fold CV performed on the whole data set (global CCR). The global CCR was 93.73%. Although the global CCR was slightly greater than the estimated CCR obtained (0.66% to be precise), the deviation should not affect the legibility of the estimated FNR or FPR much. A lesser CCR implies increased FNR and FPR, but not necessarily by an equal amount. Since the global CCR obtained is greater than the estimated CCR, it is reasonable to assume that the global FNR and FPR would be even lower than the estimated values.

Chapter 6

Conclusion

In addition to the more “floating” boundaries of the goal and purpose, we did have some qualitative and quantitative performance measures imposed on the project. For the program to be considered useful and robust enough, we required that it at least reached a CCR of 85-90% and not have a FPR of more than 10%. Looking at Table 4.3 in section 4 we can clearly see that this criteria was fulfilled with some percent to spare as well. If we take a look in the problem definition we initially set up as a guide for the project we pointed out eight (8) sub-problems to the whole project.

1. How to extract manageable data from a video stream?
2. How to locate the object of interest in an image?
3. How to extract the ROI from the image? What shape?
4. What features should be used to enable accurate classification?
5. How should the distribution of training vs. testing data be selected?
6. How should the training of the classifier be performed?
7. How to test and verify the accuracy and robustness of the classifier?
8. How to test and verify the speed and execution time?

In the final stage of the project we realized that most of these problems were trivial. Points 1,3 and 6 were all solved using Matlab-native functions or functions provided in the libSVM package. The shape of the ROI in point 3 was also quite arbitrary chosen to be rectangular for simplicity reasons. Point 2 was a more complex matter, more than one technique was utilized in order to solve this problem but it was finally solved using the GMM detector (see section 3.1.5). Point 4 regarding the type of features to be used was, as outlined earlier, basically a heuristic approach where some possible features were created and then compared amongst each other to find the ones with highest accuracy. We finally settled using a fused feature vector presented in section 3.2.6. For point 5 and 7 we divided the data-set in equal parts (383 falls and lies) to make sure that our classifier would not be biased towards a specific class. We had to reduce the data-set from 500 to 383 due to the fact that the first 110 videos did not contain a lie, just a sit. Moreover some videos were found to be corrupt and could not be used. We used the cross-validation method on the 383 videos to obtain the performance measures of the classifier that allowed us to verify its robustness and accuracy. The final classifier reached a CCR or 93.7337%, a FPR of 9.43% and a FNR of 5.79%. For the final question, point 8, which connects to the informally stated time-performance criteria, we simply used the Matlab-native tic-toc function to time our program and then divide the result with the number of frames in the processed video. By looking at Table 4.4, we can see that the final program roughly requires ~ 0.6 seconds of computation time per frame in the video in order to classify it. This means that a 50 frame video would take around 30 seconds to process which at least feels acceptable. This means that we also managed to reach that performance criteria as well. Of course, if this value could have been reduced to a tenth, that is, taking ~ 0.05 seconds per frame we would have been able to classify a video with a frame rate of 20 in real time. This improvement could be well worth looking into in order to improve the usefulness of the program. Other areas to improve upon would have been the generation of the features since some of them lost a bit of information during their extraction. Finding more ”non-destructive” feature extraction methods or trying the ones presented in section 5 could perhaps have increased the CCR of the classifier.

Credibility assessment Considering that the used resources, Matlab and libSVM are commonly used, and have high credibility and extensive bug-tests of their own, we can at least be sure that these do generate correct results. Also the methods used throughout this thesis are known to work and in the cases where the methods are new we have tried to document them so that they easily could be repeated later on by some other party. Of course there is always the risk of us having chosen erroneously at a certain point in the project which in turn would have given us lower results than we might have gotten had we chosen “right”. However there should not exist a case where a “wrong” choice generates better results which implies that the presented CCR becomes a sort of lower bound for what is possible with this particular data-set and imposed limitations.

Looking back at the results and statements made in the previous chapters we feel quite satisfied with our effort and where it has led us. Back when the project started, the goal was to create a Matlab software (function) that could make a distinction between the two classes of activities, falling and lying down. Since we actually managed to create such a function with an acceptable CCR we feel that the goal had been reached. In addition we also created a graphical user interface which makes the function easy to use. We also feel, when considering the purpose of the project, that we have fulfilled the part of obtaining a general understanding of some of the methodologies in computer vision. When the project began none of us had any experience at all with the workings of computer vision and machine learning. Throughout the project we have investigated and tried some of the most common techniques, not to mention reached a satisfying result. Therefore we feel confident to claim that our knowledge have increased. As for if our work has contributed to the general growth of the accumulated knowledge in the field, it is not for us to decide. Most probably we have not come up with any new or groundbreaking, but hopefully our efforts can still serve as a proof that the methods used in this thesis, at least in this case, can generate acceptable results. And in a sense, if our results can be of use to someone else, then we actually have expanded the general knowledge in the field, fulfilling our initial hopes.

Bibliography

[1] Microsoft kinect, Online.
URL `http://www.microsoft.com/en-us/kinectforwindows/`

[2] Volvo city safety, Online.
URL `https://www.media.volvocars.com/global/en-gb/media/pressreleases/154717/city-safety-by-volvo-cars-outstanding-crash-prevention-that-is-standard-in-the-all-new-xc90`

[3] Microsoft kinect range, Online.
URL `https://msdn.microsoft.com/en-us/library/jj131033.aspx`

[4] R. Strand, Computer assisted image analysis i: Lecture 5, mathematical morphology and distance transforms (2014).
URL `http://www.it.uu.se/edu/course/homepage/bild1/ht14/L5_morphology.pdf`

[5] P. Soille, S. A. (e-book collection), S. (e-book collection), Morphological Image Analysis: Principles and Applications, Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[6] D. Reynolds, Gaussian mixture models.
URL `https://www.ll.mit.edu/mission/cybersec/publications/publication-files/full_papers/0802_Reynolds_Biometrics-GMM.pdf`

[7] J. MACQUEEN, Some methods for classification and analysis of multivariate data.
URL `http://www.umiacs.umd.edu/~raghuram/ENEE731/Spectral/kMeans.pdf`

[8] I. The MathWorks, Gaussian mixture model, online (1994-2014).
URL `http://se.mathworks.com/help/stats/gaussian-mixture-models.html`

[9] M. Piccardi, Background subtraction techniques: a review, Vol. 4, 2004, pp. 3099–3104 vol.4.

[10] N. Joubert, Background modelling and subtraction: for object detection in video, Master's thesis, Stellenbosch University (2009).
URL `http://dip.sun.ac.za/~wbrink/students/NJoubert2009.pdf`

[11] N. Dalal, B. Triggs, Histograms of oriented gradients for human detection, Vol. 1, IEEE, 2005, pp. 886–893 vol. 1.

[12] N. Dalal, B. Triggs, Object detection using histograms of oriented gradients, online (May 2006).
URL `http://www0.cs.ucl.ac.uk/staff/a.moore/mvpractical2.pdf`

[13] High accuracy optical flow estimation based on a theory for warping, in: T. Pajdla, J. Matas (Eds.), Computer Vision - ECCV 2004, Vol. 3024 of Lecture Notes in Computer Science, 2004.

[14] J. Tompkin, Machine vision - practical 2: Optical flow an introduction, online (March 2008).
URL `http://www0.cs.ucl.ac.uk/staff/a.moore/mvpractical2.pdf`

[15] Performance of optical flow techniques, International Journal of Computer Vision 12 (1).

[16] B. K. P. Horn, B. G. Schunck, Determining optical flow, Artificial Intelligence 17 (1) (1981) 185–203.

[17] B. D. Lucas, T. Kanade, An iterative image registration technique with an application to stereo vision, 1981, pp. 121 – 130.

[18] D. G, Pattern Recognition An Introduction, Springer Science+Business Media, London, 2013.

[19] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, D. Steinberg, Top 10 algorithms in data mining, Knowledge and information systems 14 (1) (2008) 1–37.

[20] C.-C. Chang, C.-J. Lin, LIBSVM: A library for support vector machines, ACM Transactions on Intelligent Systems and Technology 2 (2011) 27:1–27:27, software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[21] P. Dollár, Piotr's Computer Vision Matlab Toolbox (PMT), `http://vision.ucsd.edu/~pdollar/toolbox/doc/index.html`.

[22] L. I. Smith, A tutorial on principal components analysis, avaliable 2015-02-27 (2002).
URL `http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf`