BRNO UNIVERSITY OF TECHNOLOGY

Faculty of Electrical Engineering
and Communication

MASTER'S THESIS

Brno, 2017   Bc. Gábor Árva

BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF RADIO ELECTRONICS

ÚSTAV RADIOELEKTRONIKY

EMBEDDED VIDEO PROCESSING FOR SURVEILLANCE SYSTEMS

EMBEDDED ZPRACOVÁNÍ VIDEA PRO DOHLEDOVÝ SYSTÉM

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR   Bc. Gábor Árva
AUTOR PRÁCE

SUPERVISOR   doc. Ing. Tomáš Frýza, Ph.D.
VEDOUCÍ PRÁCE

BRNO 2017

Diplomová práce

magisterský navazující studijní obor **Elektronika a sdělovací technika**
Ústav radioelektroniky

Student: Bc. Gábor Árva   *ID:* 154671
Ročník: 2   *Akademický rok:* 2016/17

NÁZEV TÉMATU:

Embedded zpracování videa pro dohledový systém

POKYNY PRO VYPRACOVÁNÍ:

Prostudujte dosavadní projekty/knihovny pro zpracování video signálů a počítačové vidění (např. OpenCV). Pro vhodnou platformu rodiny ARM navrhněte řetězec pro snímání statické scény a vyhodnocování obrazu s funkcemi pro dohledový systém vč. detekce pohybu a identifikace objektů, atd. Sestavte snímací systém a navrhněte způsob vzdáleného ovládání systému, vč. logování událostí, vyhledávání, apod.

Oživte celý řetězec zpracování reálných video signálů a proveďte detailní testování všech funkcí.

DOPORUČENÁ LITERATURA:

[1] OpenCV: Open Source Computer Vision [online]. 2016 [cit. 2016-05-24]. Dostupné z: http://opencv.org/.

[2] Raspberry Pi [online]. Raspberry Pi Foundation, 2016 [cit. 2016-05-24]. Dostupné z: https://www.raspberrypi.org/.

Termín zadání: 6.2.2017   *Termín odevzdání:* 16.5.2017

Vedoucí práce: doc. Ing. Tomáš Frýza, Ph.D.
Konzultant:

prof. Ing. Tomáš Kratochvíl, Ph.D.
předseda oborové rady

ABSTRACT

This diploma thesis deals with the design of an embedded surveillance system implemented on a Raspberry Pi 3 B device. The stated system includes motion and object detection algorithms realized with OpenCV functions. The evaluated information from the device is accessible via web server.

KEYWORDS

OpenCV, embedded surveillance system, Raspberry Pi 3 B, cascade classifiers

ABSTRAKT

Diplomová práca sa zaoberá návrhom embedded dohľadového systému, ktoré je implementované na Raspberry Pi 3 B zariadenie. Uvedený systém obsahuje algoritmy pre detekcie pohybu a detekcie objektov, ktoré sú realizované pomocou OpenCV funkcie. Vyhodnocené informácie sú prístupné na webový server.

KLÍČOVÁ SLOVA

OpenCV, embedded dohľadový systém, Raspberry Pi 3 B, kaskádové klasifikátory

DECLARATION

I declare that I have written my master's thesis on the theme of "Embedded video processing for surveillance systems" independently, under the guidance of the master's thesis supervisor and using the technical literature and other sources of information which are all quoted in the thesis and detailed in the list of literature at the end of the thesis.

As the author of the master's thesis I furthermore declare that, as regards the creation of this master's thesis, I have not infringed any copyright. In particular, I have not unlawfully encroached on anyone's personal and/or ownership rights and I am fully aware of the consequences in the case of breaking Regulation $\S\,11$ and the following of the Copyright Act No 121/2000 Sb., and of the rights related to intellectual property right and changes in some Acts (Intellectual Property Act) and formulated in later regulations, inclusive of the possible consequences resulting from the provisions of Criminal Act No 40/2009 Sb., Section 2, Head VI, Part 4.

BRNO   ....
                                                        (author's signature)

ACKNOWLEDGEMENT

I would like to thank doc.Ing. Tomáš Frýza, Ph.D. for professional guidance, consultation, patience and suggestions for the work.

BRNO ....

(author's signature)

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic
http://www.six.feec.vutbr.cz

ACKNOWLEDGEMENT

BRNO   . . . .
                                                        (author's signature)

MINISTERSTVO ŠKOLSTVÍ,
MLÁDEŽE A TĚLOVÝCHOVY

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

2007–13

OP Výzkum a vývoj
pro inovace

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

This thesis deals with the design of an embedded surveillance system realized on a Raspberry Pi 3 B minicomputer. The main part of this project focuses on the study of selected functions used by modern surveillance systems, like basic motion detection and algorithms used for detecting human faces. An external web-camera captures static scenes which are used as input data for the image processing algorithms. These algorithms analyze the images in real time, yielding information about moving objects and saving the video sequence if motion has been detected.

The system also includes remote access, making possible for the user to gain access to the embedded device through an internet connection. The remote access makes possible to get an insight to the actual events of the surveyed area, to browse among logged evens and to watch saved video files. The rest of the thesis is organized as follows.

The second chapter describes the used functions in the chain of video procession, discussing the process of motion detection in detail. Further, it deals with methods for cascade classifier based object tracking and the training process of a custom classifier for the detection of hands.

The third chapter deals with the design of the stated surveillance system in details. The main parts of the system are described in separate sections and their functions are presented with the help of commented source code parts, pseudo-codes and flow-diagrams.

In the last chapter, the experimental results are listed, encompassing the outcomes of the performed tests.

1.1 Surveillance system

The basic set-up of a surveillance system includes one or more cameras attached to monitors. The purpose of these systems are to give an overview of a huge area to the operators. Their task is to watch the monitors constantly then evaluate and react to the actual events. Video recorders can save the output of each camera and after an occurred incident the footage can be used as evidence. The disadvantage of this system is that it cannot be used for preventing incidents and a lot of time is required for finding the correct videos. The weakest element of these surveillance systems are the human operators. Since watching the monitors for a long period of time where nothing significant happens is a hard task and mistakes can be easily made.

The thesis includes the design of a surveillance system for indoor use, in which the tasks of human operators are substituted by a portable control unit equipped

with one or more cameras. The task of surveillance will be approached from the human perspective which includes the following activities:

- Detecting moving objects
- Recognizing objects
- Possibly tracking object movements

To automatize these task, basic Computer Vision algorithms are modified and applied to the real-time camera feed. The functions are provided by the open source computer vision libraries called OpenCV. [1]

Additionally, the video sequence is saved only in cases if a corresponding event is detected, thus saving storage space and making easier to find the desired video footage. The controller unit is equipped with remote access, so the user can also check the captured events, even the live feed from the camera from any distance. The information provided by the controller unit can be further used for calling events like triggering alarms, sending messages to the owner or calling the police. [1]

1.2 Computer vision

Humans perceive the three-dimensional structure of the world around them with apparent ease. They can easily specify the attributes of objects, recognize people in pictures even guess the emotions on their faces. Researchers in computer vision have been developing mathematical techniques for recovering the shape and appearance of objects in imagery. With them, one can track movement, rebuild 3D models of an environment and recognize people by their face. However, despite all of these advances, the idea about a computer that interprets images as a small child does, is still elusive. [2]

The basic of computer vision is to describe the world in one or more images and to specify and reconstruct its attributes, as color distribution and shape. Fortunately in today's digital world with the advent of powerful and affordable computing devices, it has never been easier to create the desired imagined applications. Huge amount of tools and libraries are offered for picture and video manipulation. For anyone who wishes to develop his own application, the OpenCV library is a considerable option to use.[3]

1.2.1 OpenCV

OpenCV stands for Open Source Computer Vision. It is an open source library containing over 500 optimized algorithms for image and video analysis and manipulation. OpenCV was introduced in 1999, since then, it has widely spread among researchers and developers as the main development tool in computer vision. It was originally developed at Intel to create vision-based CPU-intensive applications. The version 1.0 was launched in 2006, which was followed by the second major release in 2009 with OpenCV 2 that recommended new changes, like the C++ interface.

OpenCV is aimed at making computer vision accessible to programmers and users in the area of real-time human-computer interaction. There are available detailed manuals, source-code examples and tutorials which eases the work of the developers. It includes libraries for object/human/face detection, recognition, tracking and segmentation as well as camera calibration and 2D, 3D shape reconstruction. It has C++, C, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. The full list also detailed tutorials can be found on the official website [4].

1.3 Raspberry Pi

Raspberry Pi is a miniature computer with the size of a credit card developed by the Raspberry Pi Foundation. The main aim of this product was to create a device, which makes the programming courses for children easier, more enjoyable and all of this for a reasonable price. After the introduction however, this little computer gained favor against many programmers and developers due to its cheap price and wide range of utilization. Since the first model, which was introduced in 2012, these microcomputers have developed more and more mainly in hardware equipment and fortunately not in dimensions. [6]

The latest model is the third generation Raspberry PI shown in Fig 1.1, which is available since February 2016. The dimensions of this model are 8.5 x 5.3 cm, within this relatively small area the deice has a quad-core ARM processor with 1 GB RAM, a built-in Wireless LAN module and a long variety of ports.

For its proper function the device needs to be connected to an external power supply rated at 5 V with a minimal current of 2 mA. As a power supply any mobile charger can be used with higher current output than the required value, giving the Raspberry Pi users even more freedom. The maximal power consumption is stated as 3.5 W.
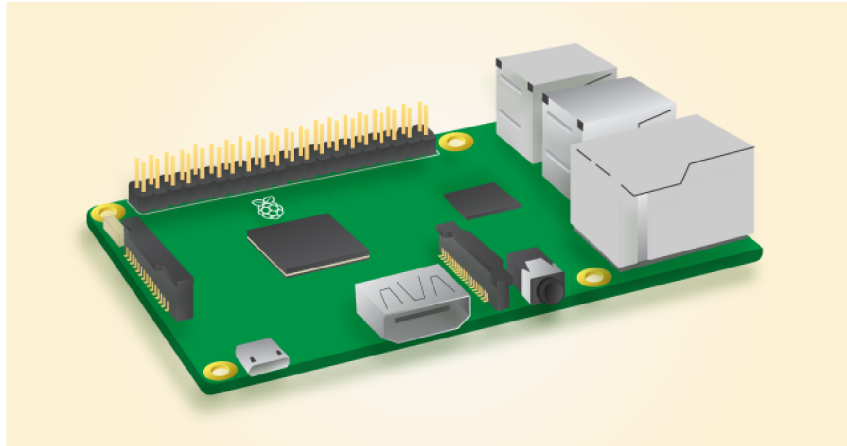
Fig. 1.1: Raspberry PI model 3 B [6]

The Raspberry PI 3 B has the following equipment and technical properties:[6]

- f A 1.2 GHz 64-bit quad-core ARMv8 CPU
- 802.11n Wireless LAN
- Bluetooth 4.1
- Bluetooth Low Energy (BLE)
- 1 GB RAM
- 4 USB ports
- 40 GPIO ports
- Full HDMI port
- Ethernet port
- combined 3.5 mm audio jack and composite video
- Camera interface (CSI)
- Display interface (DSI)
- Micro SD card slot
- VideoCore IV 3D graphics core

All these technical parameters makes this microcomputer an appropriate choice for this project. It has enough computational power for the manipulation of static image frames, while with a proper camera module the system still possess small dimensions and weight.

2 DESIGN OF THE SYSTEM

2.1 Hardware part

For the central control unit, the Raspberry PI 3 B was selected. Before one can start to work on the newly bought product, an operating system have to be installed on the device. The installation process requires an SD card and a downloaded version of one of the free operation systems available. The image file containing the operational system, a detailed guide for the image writing and an installation tutorial can be all found on the Raspberry homepage: [6]. For this project the Raspbian Jessie Lite operating system had been chosen.

2.1.1 Block scheme

The block scheme of the whole hardware part is shown in Fig. 2.1. The source code was developed directly on the mini computer with the assistance of the connected peripheral devices.

Fig. 2.1: Block scheme of proposed surveillance system

2.1.2 Installation of OpenCV libraries

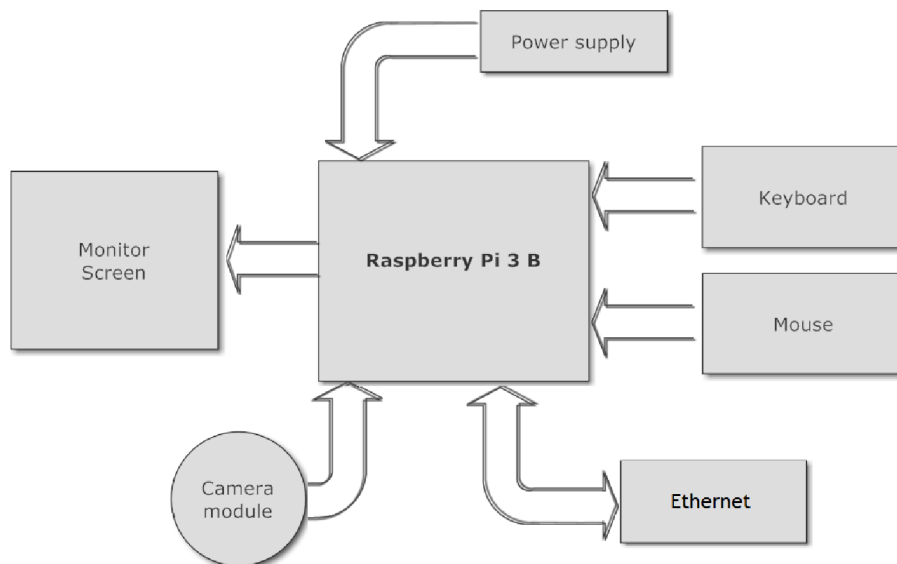For the correct function of OpenCV a few other packages have to be installed on the device. This can be done with typing a few commands to the terminal with an active internet connection. In the first place, the already installed packages and the firmware of the minicomputer have to be updated.

```
sudo apt−get update
sudo apt−get upgrade
sudo apt−get dist−upgrade
sudo rpi−update
```

Next step is to install the build system `CMake`, a free distributed version control system `Git` and a few other dependencies.

```
sudo apt−get install cmake git
sudo apt−get install build−essential
```

GTK development libraries used for the visualization of GUI (Graphical User Interface):

```
sudo apt−get install −y libgtk2.0−dev pkg−config
```

The other packages needed by the OpenCV can be found among the installation tutorials on the official OpenCV homepage [4]. The version of OpenCV libraries used in this thesis can be downloaded through the terminal with the following commands:

```
cd ~/
git clone https://github.com/Itseez/opencv.git
git clone https://github.com/Itseez/opencv_contrib.git
```

As the last step, a new folder is created for the build files. After the building process, the libraries are compiled and installed. The whole process can take up to 3 - 4 hours.

```
cd ~/opencv
mkdir build
cd build
sudo cmake −D CMAKE_BUILD_TYPE=RELEASE
−D CMAKE_INSTALL_PREFIX=/usr/local
−D OPENCV_EXTRA_MODULES_PATH=../../opencv_contrib/modules
−D WIHT_V4L=ON −D INSTALL_C_EXAMPLES=ON −D BUILD_EXAMPLES=ON
−D WITH_FFMPEG=OFF ..

sudo make −j2
sudo make install
sudo ldconfig
```

2.2 Software part

2.2.1 Motion detecting algorithm

The first task is to teach the Raspberry Pi how to detect motion in the captured video footage. There is a large amount of available algorithms on the world web already, the only thing what limits their potential is the hardware of the processing unit. The selected motion detecting algorithm is based on frame difference method [5], which is the most suitable choice for a static camera system.

The main idea is to compare two frames of the video file. Since the background is constant, the content of the frames are the same, except in the region of moving objects. The difference between these images provides information about the location of the moving objects. The detected motion in the video sequence is then highlighted with a rectangle drawn around it. An example for the result can be seen in Fig. 2.2.

To detect any motion in a given video sequence, a reference image is required. This image can be captured once containing only the scene background, or it can be replaced continuously during the process. The actual image is taken from the real-time camera feed what is then converted into gray-scale format showed in Fig. 2.3 (a). The reference image is then subtracted from the newly captured one, resulting in an output picture including only the pixel values of the moving object (b).

Fig. 2.2: Illustration of detected motion

In the next step a threshold function is applied to the output image (c), setting the intensity value of each pixel to maximum or minimum, considering a threshold value. This image still contains pixel errors caused by inadequate light conditions or the imperfection of the camera attributes. These errors mainly occur on the edges of objects and in bright areas of the image. The thresholded image is further processed and filtered by corresponding functions, resulting in image (d). The further description of these functions can be found in the following chapters.

(a) Grascale frame

(b) Absolute difference of frames

(c) Applied Threshold

(d) Applied blur
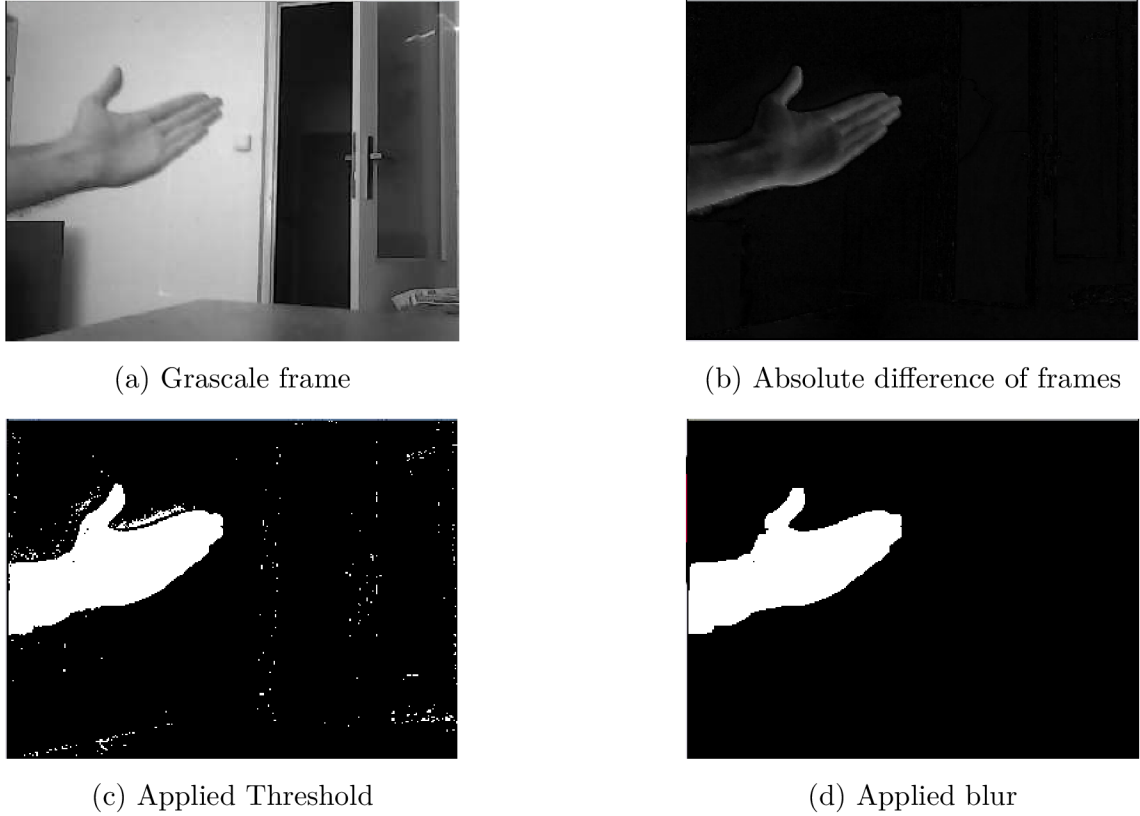
Fig. 2.3: Image procession for motion detection algorithm

If the algorithm would use a single reference image captured at the start-up of the system, all objects which were not present at the given time would cause a "false" motion, regardless if they are moving at the moment or staying still. To remove this unwanted feature, at a given time, two or more consecutive images can be selected from the video file to apply the described process to them. The disadvantage of this method is that the sensitivity to moving objects is highly depending on how many frames are captured in one second. If the fps is too high, the algorithm may not react to slow movements, but low fps degrades the quality of the video. To eliminate this problem, the reference image is not replaced with new images, but is blended with the actual image. This process is called linear blending, which adds the two input images with weighted values based on the following equation:

$$dst = \beta \cdot reference\_img + \alpha \cdot actual\_img \qquad (2.1)$$

$$\beta = 1 - \alpha \qquad (2.2)$$

This modification results in an increased sensitivity for the slower motions, since now the alpha value determines the alteration speed of the reference image. Finding the right value is vital for the correct operation and it is highly dependent on the actual fps. The flowchart of the whole algorithm can be found in Fig. 2.4.
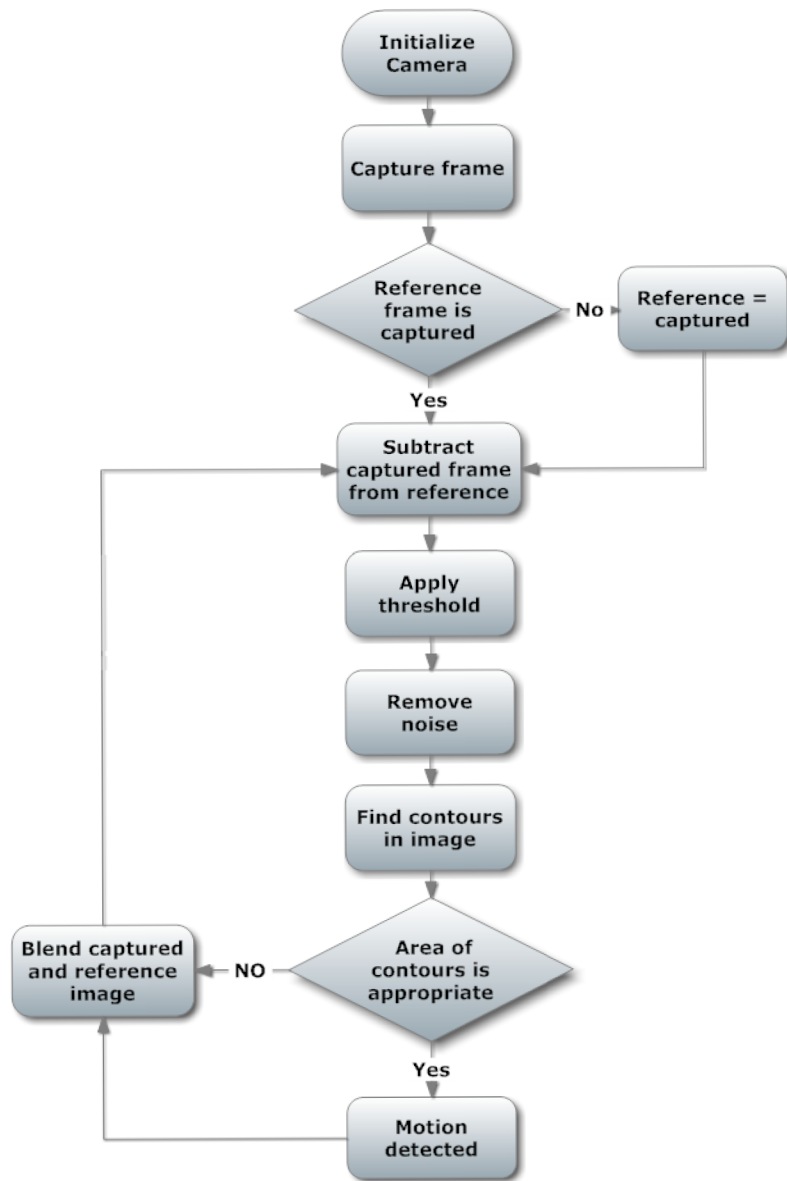

Fig. 2.4: Flowchart of motion detection

2.2.2 Object detecting algorithm

Motion detecting methods provide information about motions taking place in the video footage, but they return no information about the features of the moving object itself. The main part of this thesis deals with the problems and solutions of object detection otherwise object tracking. The aim of these algorithms is to recognize the predetermined database of objects in the given video sequences, or in this case, to apply it on a real-time camera feed. In the case of a surveillance systems, the detected objects are often faces of individuals appearing on the screen or license plates of the passing cars. In fact, the detected object can be almost anything and is only limited by the user's needs.

Object detection using Haar cascades

The first algorithm for object detection included an applicated cascade of simple classifiers. To detect the specific object in images, the first step is to learn its features. However, working only with the intensity values of RGB pictures made the feature calculation computationally expensive. As a result in 2001, Viola and Jones developed a framework using the Haar-like features. They learned that feature based classifiers learn easier from a finite quantity of data and they operate much faster than pixel based ones [7]. The Haar features can be obtained by monitoring adjacent rectangular areas in a detection window and summing up the pixel intensities in each region. Then differences are calculated from the intensity sums, which are used to categorize the subsections of the image. These rectangle features seems to be primitive comparing to steerable feature filters, but their efficiency and speed are compensating the deficiencies.
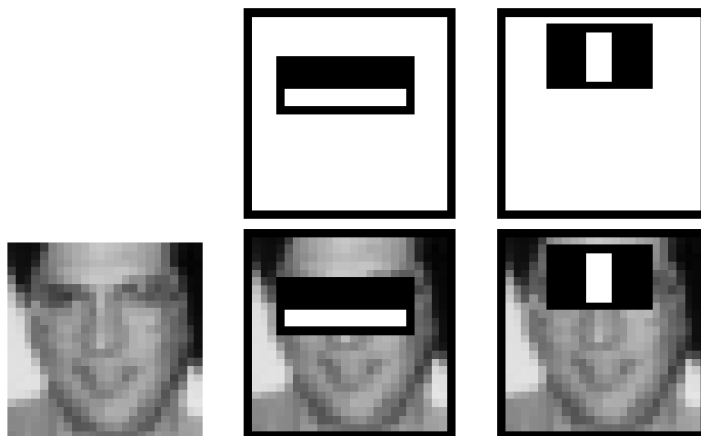
Fig. 2.5: The features selected by AdaBoost [8]

The first problem is, that the 24 x 24 pixel base resolution of these sub-windows return an exhaustive amount of 180,000 rectangle features, however only a few of these features can be combined to form an effective classifier. For this task, the developers implemented a learning algorithm to their project called AdaBoost. In the case of face detection, for which purpose this framework was originally developed, the learning algorithm selected 2 main features shown in Fig. 2.5. As last, the classifiers are gathered into cascades. With this step, the algorithm achieves increased detection performance with less computation time. A series of classifier are applied to the sub-windows starting with the most simple ones, and a positive result from these triggers the next more complex classifiers. The decision tree of the process can be seen in Fig. 2.6. Further information and test results are available in the literature [8].

Fig. 2.6: Depiction of the cascade classifiers [8]

Object detection using local binary patterns

The discovery of Local Binary Patterns allowed a more modern approach to the field of object detection. Compared to the Haar cascade based algorithm, it scans the input image more faster and requires less time to train a working classifier, while the main i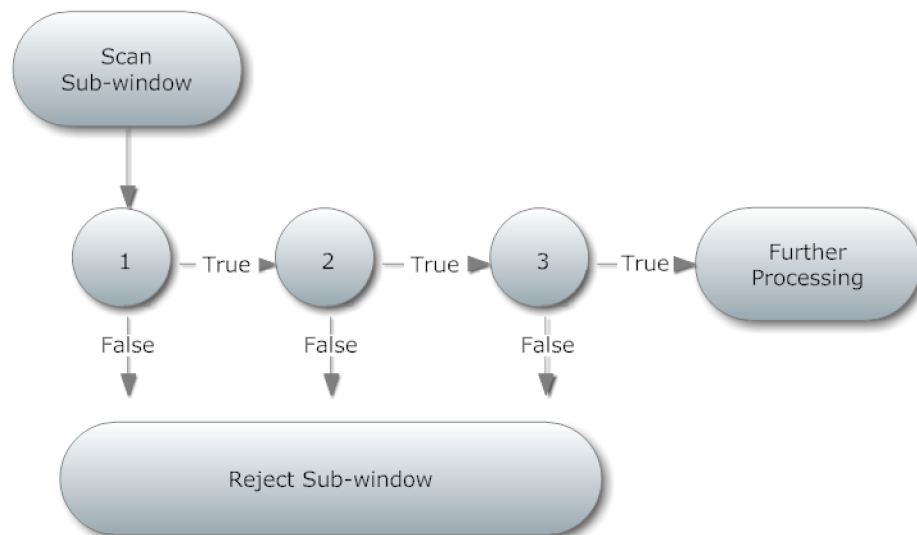dea remains the same. The algorithm categorizes the pixels of an image by applying threshold function to them in a 3 x 3 sub-window with a center value. This step is applied to each pixel of an image and the results are saved as 8 bit binary strings or decimal values containing the LBP features. The approach of the LBP operator is shown on Fig. 2.7.
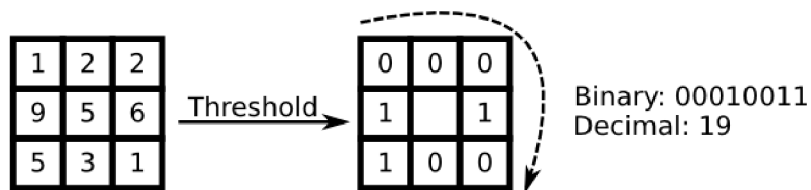
Fig. 2.7: Approach of Local Binary Patterns [9]

The method's scantiness manifests in the features captured in the small 3 x 3 area. These sub-windows cannot capture larger scale structures which may contain the dominant features of the detected object. To overcome this obstacle, a new representation was proposed by a group of Chinese engineers called Multi-scale Block Local Binary Patters (MS-LBP). The basis remains the same so each pixel is labeled and thresholded in the 3 x 3 sub-window with a center value considering the results in a binary string. The histogram of these labels are then further used as a texture descriptor. In each sub-region an average sum of image intensity is computed and another threshold function is applied to them by the center block, resulting in a MS-LBP process. The output images are influenced by the scale of chosen blocks: images filtered with small scale values represents more details and micro patterns, while big scale values reduce pixel noise and focuses more on the dominant features of the object. In Fig. 2.8, two faces are filtered with different scales. The (a) is the original gray-scale image, (b) is filtered by 3 x 3 MB-LBP, (c) is filtered by 9 x 9 MB-LBP and (d) is filtered by 15 x 15 MB-LBP. [9]

Fig. 2.8: MB-LBP application to random faces [9]

This new method is more robust then its predecessor, encoding not only micro-structures located in the default 9 pixel area but also macro-structures of image patterns, resulting in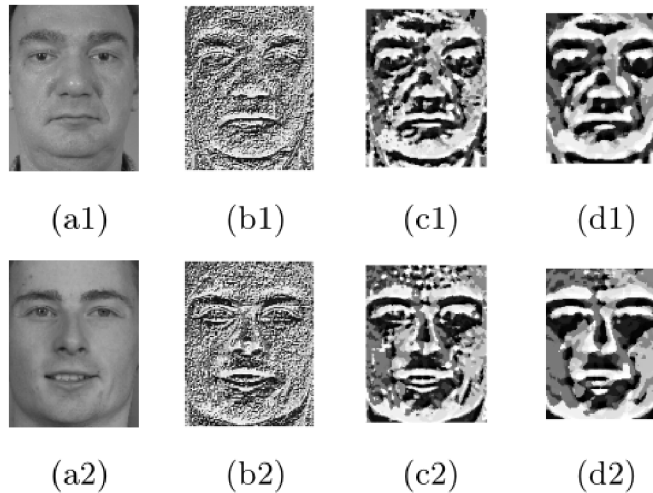 a more complex image representation model. The next task is to construct a classifier from the gathered features. Due to their huge amount, these measures contain much redundant information which have to be removed. For this purpose a Gentle AdaBoost learning algorithm is used to select the correct features from the many [9].

2.2.3 Cascade classifier training

The OpenCV library offers predefined cascade classifiers for face, eyes or mouth detection. However, if one requires a working classifier to detect other objects, the classifier needs to be trained individually. Fortunately, the OpenCV library includes the tools and functions to generate a custom cascade classifier based on Haar or LBP features.

The training time of a Haar cascade classifier can take up to weeks, while an LBP classifier for the same object can be trained in a few hours. Due to this reason an LBP cascade classifier will be trained for the detection of hands. To begin the training, a few other functions are needed beside the available OpenCV applications. Fortunately there is an open source directory available on GitHub including everything what is required for the training process [10]. It can be download by the following command in the terminal:

```
cd ~/
git clone https://github.com/mrnugget/
opencv-haar-classifier-training
```

The training process itself requires a set of positive sample images including the detected object, and a set of negative sample images containing possibly everything except it. According to researches [11], the amount of positive and negative samples may vary around a few thousand, however to acquire such a large amount of pictures about and without the desired object is hard to accomplish.

The negative samples, also called background images, are taken from random pictures omitting the detected object. Obtaining the sufficient amount of them is a time-consuming process. One way is to gather them manually or one can extract frames from video files and save them as separate images. In this project, the amount of negative images is 800. Once the required number is reached, the images have to be copied to the `./negative_images` directory of the cloned repository and run the command below, which creates a text file including the names of all negative images. [10]

```
cd ~/
find ./negative_images -iname "*.jpg" > negatives.txt
```

For a good classifier, a relatively large amount of positive samples is required. However, for this custom made classifier 100 pictures will be satisfactory including the hands of 5 people in different position and with different backgrounds. A set of the used positive samples are displayed in Fig. 2.9. After collecting them, the images are copied to the `./positive_images` directory and another text file is created containing the names of all positive images.

Fig. 2.9: Set of positive images

```
cd ~/
find ./positive_images -iname "*.jpg" > positives.txt
```

A specific OpenCV function, the `opencv_createsamples` is used to significantly increase the number of positive samples. This function generates a large amount of samples from the existing images by applying transformations and distortions to them. A short script written by Naotoshi Seo combines each positive image with a random negative image and runs them through the `opencv_createsamples` function, obtaining another 1900 positive samples [11]. The process results in a `*.vec` file used for the cascade training. This method is time optimal and does not need a lot of preparations, but will never be as effective as a classifier trained with real positive sample images.

```
perl bin/createsamples.pl
positives.txt negatives.txt samples 2000
opencv_createsamples −bgcolor 0 −bgthresh 0 −maxxangle 1.1
−maxyangle 1.1 −maxzangle 0.5 −maxidev 40 −w 40 −h 40"
```

The Perl script above returns 100 `*.vec` files to the Samples directory, which have to be merged to one file. To do so, a text document is created including the collected file names and the `mergevec.py` Python script merges the `*.vec` files.

```
cd ~/
find ./samples −iname '*.vec' > samples.txt
python tools/mergevec.py −v samples −o samples.vec
```

The resulting `samples.vec` can now be used to train the classifier. The application for the cascade training is the newest version of `opencv_traincascade` **greatest** compatible with the latest OpenCV version. The role of main parameters are described in this section, a detailed description about the other parameters and settings can be found in the official OpenCV site [12]. The number of the negative and positive samples are specified by `-nemNeg` and `numPos`, the `-w` and `-h` gives the sample size and the `-precalcValBufSize` and `-precalcIdxBufSize` determine the memory size used for the training process. The combined value of the last two parameters may not be larger than the available RAM of the computer. The `-numStages` parameter gives the iteration number the training process. Small number produce a weak classifier causing a lot of false positive matches, while a large number causes the classifier to end up over-trained and will be too selective. [11]

```
opencv_traincascade −data classifier −vec samples.vec
−bg negatives.txt −numStages 30 −minHitRate 0.995
−maxFalseAlarmRate 0.5 −numPos 1800 −numNeg 1000
−w 40 −h 40 −precalcValBufSize 256 −precalcIdxBufSize 256
−featureType LBP
```

The length of the whole training sequence depends on the number of samples and stages. With the previous parameters on a Raspberry Pi 3 B model, the whole process took 2 hours and 50 minutes.

3 REALIZATION OF THE SYSTEM

In this chapter one can find the described blocks located in the chain of image procession required for the concerned surveillance system. The individual sections include parts of source codes, pseudo-codes and flowcharts for better understanding. The flow chart of the whole chain is shown in Fig. 3.1.
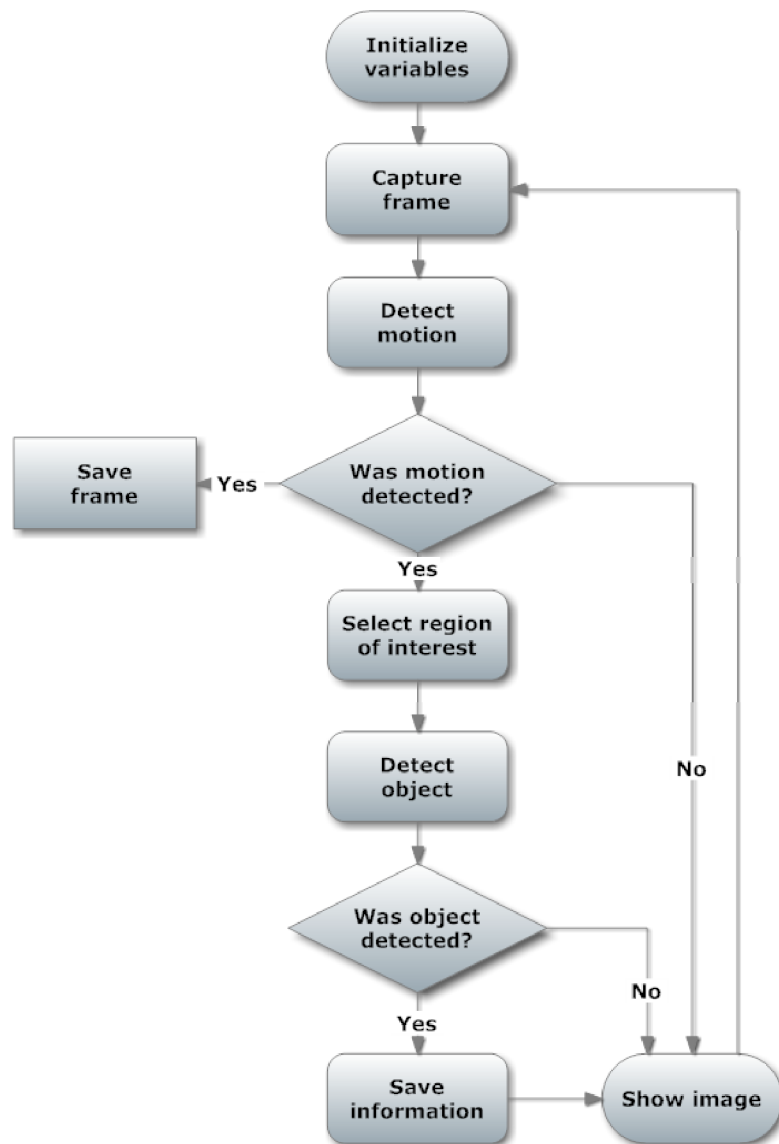

Fig. 3.1: Simplified flow chart of the surveillance firmware

3.1 Capturing frames

To allow the Raspberry Pi to "see" the surveyed area, an external web-camera is connected to the device via USB port. The access to the camera is realized through OpenCV functions, allowing to capture images into a video stream or as individuals. Since the following algorithms are working with static scenes, the actual images about the area are saved as separate frames. The source code parts included in this section are forming the basis of the whole algorithm.

First, an initialization is required for the camera module. The *VideoCapture* function opens the default camera, then the correct resolution is set beside the wanted frame per second ratio. The used web camera supports a maximum resolution of 640 x 480 pixels and a capture speed of 30 frames per seconds.

```
#define frame_width 640;
#define frame_width 480;
#define fps 30;

cv::VideoCapture cap(0);
cap.set(CV_CAP_PROP_FRAME_WIDTH, frame_width);
cap.set(CV_CAP_PROP_FRAME_HEIGHT, frame_height);
cap.set(CV_CAP_PROP_FPS, fps);
```

The OpenCV among other features provides new data types like *Mat*rices and *Scalar*s, serving as containers for the captured images. A matrix consists of a 2 dimensional field of scalar variables, containing the intensity values of the corresponding pixels. The length of these scalar variables are depending on the video file format, although, that how many channels are the format consists of. For example an RGB image has 4 channels (Red, Green, Blue and Brightness), so the corresponding Scalar has 4 assigned numbers. For the visualization of captured and processed images, a new window has to be initialized. As a default setting, the *VideoCapture* function returns the frames in RGB format. The information about the data types were drawn from the literature [13].

```
cv::namedWindow("Captured frames", CV_WINDOW_AUTOSIZE);
```

The core of this embedded application is the infinite loop. In each loop, the captured image is saved to the concerned matrix and the visualization window is refreshed according to it. An additional function is called for adding the actual date and time to the upper left corner of the image. The loop persists until a required key is pressed.

```
for (;;)
{
    Mat cameraFeed;
    cap >> cameraFeed;

    //body of the program

    addDateTime;
    imshow("Captured␣frames", cameraFeed);
    if (cv::waitKey(10) >= 0)
    {
        cout << "...exiting" << endl;
        break;
    }
}
```

The *addDateTime* function does no more, than gathers the date and time from the system clock. This information is then added to the image with the *putText* function from the OpenCV libraries. A point have to be determined, giving the initial x and y coordinates for the *putText* function.

```
void addDateTime( Mat &frame)
{
    time_t rawtime;
    struct tm *timeinfo;
    char buffer[80];

    time(&rawtime);
    timeinfo = localtime(&rawtime);

    strftime(buffer, 80, "%d-%m-%y␣%I:%M:%S", timeinfo);
    std::string str(buffer);

    Point org;
    org.x = 30;
    org.y = 30;
    putText(frame, str, org, 1, 1, Scalar(0,0,255), 1, 7);
    //putText( frame, string, Point, Font Type, Scale,
    //         Color Red, Thickness, lineType )
}
```

3.2   Detecting motion

The most fundamental function of this surveillance system is the detection of basic motions and determining their location in the video footage. The functions applied to the image sequence are drawn from the OpenCV library, their detailed description can be found on the official website [14].

```cpp
cv::Mat motionDetect(cv::Mat &currImg, cv::Mat &prevImg)
{
  cv::Mat thresImg;
  absdiff(currImg, prevImg, threshImg);
  threshold(thresImg, thresImg, 20, 255, 3);
  erode(thresImg, thresImg, Mat(), Point(-1,-1), 1);
  dilate(thresImg, thresImg, Mat(), Point(-1,-1), 1);

  return thresImg;
}
```

The described pattern works only with the intensity values of the pixels and discards the colors. Due to this, the input image is transformed into gray-scale format to remove the color channels and to reduce the file size of the processed images. The `cvtColor` serves the desired purpose, performing the image transformation based on the stated conversion code. To transform RGB to gray-scale format, the `CV_BGR2GRAY` code is selected, merging the color channels into one Y channel according to the equation below:

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \tag{3.1}$$

Before the algorithm could determine motion, a reference image has to be saved to the `refImg` variable with the specific function. The image is copied only in the first loop and further will be influenced by the actual image with the `addWeighted` function, what simply blends the two images with weighted values. Considering subjective experiences, the alpha parameter is set to 0.15.

```cpp
cvtColor(cameraFeed, grayImg, CV_BGR2GRAY);
if(refImg.empty() == true)
{
        grayImg.copyTo(refImg);
}
addWeighted(grayImg, alpha, refImg, beta, 0.0, prevImg)
```

3.2.1 Absolute difference

The next step is the background subtraction from the image, removing everything what is not connected to the moving object. The `absdiff` function calculates the absolute difference between two arrays, in this case between the pixel values of the given images.

$$dst = saturate(|src1 - src2|) \qquad (3.2)$$

The output array of pixels will contain near to zero values where the two images are identical and non-zero valued pixels in the area of motion.

```
absdiff(currImg, prevImg, thresImg );
```

3.2.2 Thresholding

The values of these pixels may vary in a range from 0 to 255 according to the difference between the two arrays. For separating the region of interest from the rest of the image, a basic segmentation method, the `threshold` function is used. This function performs a comparison between each pixel's intensity value and a predetermined threshold limit, setting the intensity value of the corresponding pixel to maximum or minimum depending on the threshold type. The OpenCV library supports 5 type of threshold operations including binary thresholding what is used in this paper. The function can be expressed by the equation 3.3, which says that if the pixel intensity value is larger than the threshold limit, then it is set to maximum (white color), otherwise it is going to be set to black with a pixel intensity value of 0.

$$dst(x,y) = \begin{cases} maxval & if\ src(x,y) > thresh) \\ 0\ otherwise \end{cases} \qquad (3.3)$$

The `threshold` function has the following input arguments:

- `src` - source image
- `dst` - destination image
- `threshold_value` - threshold limit value set to 20 by subjective experience
- `max_Binary_value` - the value set by the positive result of binary threshold
- `threshold_type` - for binary threshold the value is 0

```
threshold(thresImg, thresImg, 20, 255, 0);
```

3.2.3 Erode and dilate

The acquired image contains many false non-zero pixel values caused by pixel noise. To remove these errors, additional morphological operations are applied to the processed image. These functions are the `erode` and `dilate` used for noise removal and for the isolation of individual elements or to join disparate ones in the image.

The *erode* function is applied first to the corresponding image. This function performs a convolution of a kernel image and the input image. The kernel image is scanned over the input image computing the minimal pixel value overlapped by the kernel. The intensity value of the pixel located in the kernel image center is then replaced with the minimal value. This causes the white regions of the image to shrink and get thinner, removing the unwanted noise from the image.

```
erode (thresImg, thresImg, Mat(), Point(−1,−1) ,1);
dilate(thresImg, thresImg, Mat(), Point(−1,−1), 1);
```

Dilating is the twin process of eroding, while it does exactly the opposite thing. The intensity values of the center pixels are replaced with the maximum value in the stated region, making the white region of the image larger and thicker thus compensating the area lost during the erosion and smoothing the contours.

The two functions have the same input argument:

- `src` - source image
- `dst` - destination image
- `kernel` - structuring element for the process, in the case of Mat(), a 3 x 3 rectangular structure is used
- `anchor_point` - the element center is set as default or by the (-1, -1) value
- `iteration` - the number of iteration is 1

The computational time of these operations is relatively long, therefor it can be shortened by reducing their iteration number.

The resulting image now contains the information about the occurred motion. The background image is usually permanent and one can assume that it does not contain any objects of interest.

3.3   Selecting region of interest

The computational time required for the object detecting algorithm is directly proportional to the area of the scanned image. On this basis the reduced amount of scanned pixels decrease the required time for the detection process, increasing the frame per second ratio of the whole system. Due to this, only those image sections are scanned by object detection, where a motion has occurred.

The aim of the `extractROI` function described with the pseudo-code Alg. 3.1, where ROI stands for region of interest, is to select and crop the interested part from the original image by evaluating the threshold image received from the `motionDetect` function. In the first place, `findContours` determines the contours of the present blobs, saving their coordinates in the `contours` vector. As the next step, a rectangle is calculated around each blob and their area is determined with the help of contour moments. The rectangles are then sorted to filter out too small ones probably caused by pixel errors and too big rectangles which may be caused by changing light conditions or an unexpected motion of the camera. If a rectangle meets the specified requirements, it's attributes are saved to a predefined structure named `rectan`.

```
struct Rectangles {
        Point  pointPos[100];
        Point  tl;
        Point  br;
        int    pNumber;
        bool   show;
        Mat    img;
};
```

A single moving object may consist of more blobs which are not connected. This phenomena causes the algorithm to draw rectangles around each blob belonging to the same object. To eliminate this problem, the rectangles located really close to each other or overlap, are merged together. To accomplish such thing, each area embraced by a rectangle is sampled and is further compared with the samples of other rectangles. The pseudo-code of the algorithm can be found in Alg. 3.2.

The sampling process works the following way. If a pixel is located within the rectangle and their x and y coordinate is dividable with `sampl_freq` without reminder value, the coordinates are saved as samples. These acquired points located in the given rectangles are then compared with other rectangle samples. If a sampled coordinate belongs to more than one rectangle, it means the area embraced by the rectangles overlap, and are further merged together.

Algorithm 3.1: Extracting region of interest

```
1    initialize rect_counter for 0;
2    initialize MIN_BLOB_AREA for 50*50;
3    initialize MAX_BLOB_AREA for frame_width*frame_height/1.5;
4    initialize vector for contours;
5
6    find contours in matrix MD_img
7    initialize vector of rectangles boundRect for the size of contours
8
9    for i 0 to size of contours
10    compute rectangle around contours[i] and save to boundRect[i]
11    compute moments of contours[i]
12    compute area of rectangle
13    if area > MIN_BLOB_AREA and area < MAX_BLOB_AREA
14     object is found
15     save top−left and bottom−right point of the rectan[rect_counter]
16     set rectan[rect_counter].show to true value
17
18     sample the area of the rectangle
19        for r rectan top−left point y to
20        bottom−right point y coordinate
21         increment sample_freq
22         for c rectan top−left point x to
23         bottom−right point x coordinate
24             increment sample_freq
25          rectan[rect_counter].pointPos[p_counter].y is r
26          rectan[rect_counter].pointPos[p_counter].x is c
27          increment p_counter
28         endfor
29        endfor
30     increment rect_counter
31    else
32       object is not found
33    endif
```

The next part of this function merges together the rectangles which share a common area, or those closer to each other than the half of `sampl_freq` value. Sampled rectangles also have an assigned attribute called `show` with true default value. The function gradually compares the sample values and in the case of a match the corresponding rectangles are merged, modifying the top-left and bottom-right point of the first one to cover both areas. Further, the second rectangle is disabled by setting their `show` attribute to false value thus telling the algorithm to discard it.

Algorithm 3.2: Algorithm for rectangle merging

```
1    initialize overlap to true
2    while over is true or overlap counter < 10
3     for a 0 to rect_counter
4      for b a+1 to rect_counte
5       if a != b and rectan[a] and recta[b] show is true
6        for c 0 to number of samples of rectan[a]
7         for d 0 to number of samples of rectan[b]
8          if rectan[a] c sample coordinates are equal to
9           rectan[b] d sample coordinates
```

33

```
10
11          overlap is true
12          find the lesser x coordinate for top-left point
13          find the lesser y coordinate for top-left point
14          find the bigger x coordinate for bottom-right point
15          find the bigger y coordinate for bottom-right point
16
17          resample the area of the rectangle
18          set rectan[b] show parameter to false
19        endif
20       endfor
21      endfor
22     endif
23    endfor
24   endfor
25  endwhile
26
27  for a 0 to rect_counter
28   if rectan[a] show parameter is true
29    draw rectangle around area
30    save the image of rectangle
31   endif
32  endfor
```

The effect of the rectangle merge process is shown in Fig. 3.2, where the left picture represent the evaluated threshold image, the center picture represents the display of detected motions without merging the rectangles and the right picture shows the image with merged rectangles.
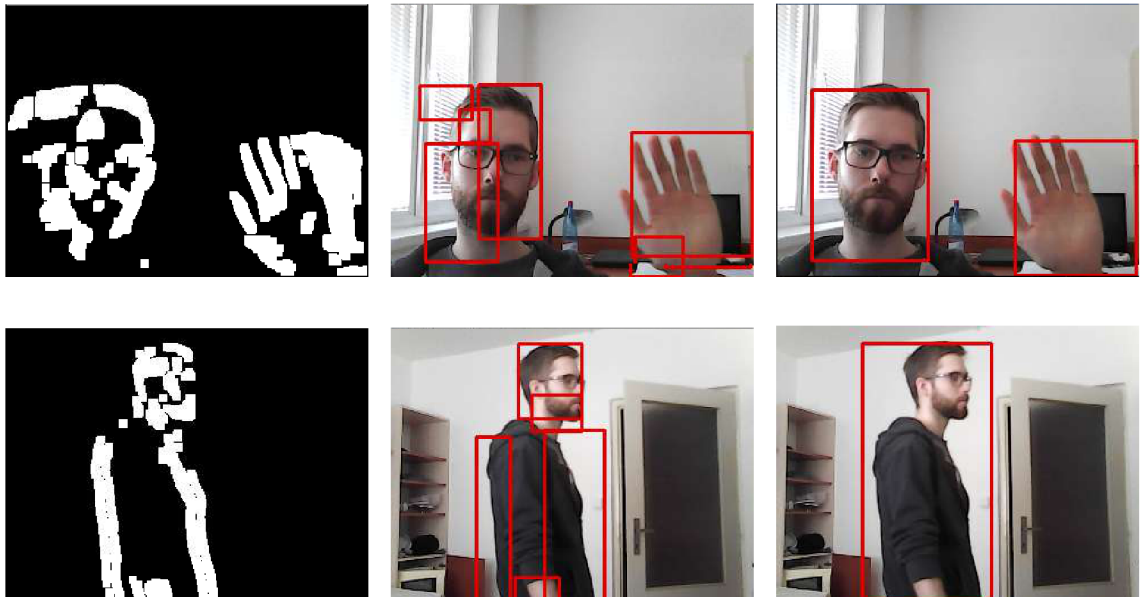

Fig. 3.2: The effect of rectangle merge

The result of this function is a set of images representing individual moving objects in the video sequence. These images are also used as input data for the

object detecting algorithm instead the whole image from the camera feed, thus decreasing the region of interest. To highlight these motions for the users, also a red rectangle is drawn around each of them. Further test results of the algorithm are shown in the `Experimental results` chapter.

3.4   Object detection

In this paper, LBP based Cascade Classifiers are utilized for the detection of frontal face and hands. The OpenCV library already contains ready to work cascade classifiers, inter alia, for the detection of frontal faces, eyes and mouth. Additionally a custom classifier is trained and included to the project for the detection of open hands and palms. The behavior of this classifier is described further in this paper.

The core source code for the object detecting function was drawn from an OpenCV tutorial available on the official OpenCV website [15]. Further in this section, the source code of face detection is thoroughly described.

At first, the location of the XML files have to be saved as strings to the corresponding global variables. These files are then loaded with the `loadCascades` function, returning an error message in the case of missing or wrong files.

```cpp
String face_cascade_name =
"Cascades/lbpcascade_frontalface.xml";
CascadeClassifier face_cascade;
int loadCascades(void)
{
    if(!face_cascade.load(face_cascade_name) )
    {
        cout << "Error loading face cascade" << endl;
        return -1;
    }
}
```

Since the execution time of the cascade classifier based detection process is highly dependent on the size of the scanned image and the minimal detection window size of the algorithm, the input image is scaled down with an integer value. The command performing the object detection based on the loaded classifier, is the `detectMultiScale` function. It scans the input image for the desired object, which in the case of successful detection are then returned as a vector of rectangles defined as `faces`.

```cpp
bool faceDetect( cv::Mat inp_img, cv::Point st_pt,
                 cv::Point &p_tl, cv::Point &p_br)
{
    const int scale = 3;
    cv::Mat gray_img;
    // transform input image to gray-scale format
    cvtColor(inp_img, gray_img, CV_BGR2GRAY);
    // resize input image
    cv::Mat resized_gray_img(
        cvRound( gray_img.rows / scale),
        cvRound(gray_img.cols / scale),
        CV_8UC1);
    cv::resize( gray_img, resized_gray_img,
                    resized_gray_img.size());
    std::vector<Rect> faces;
    //scan input image for faces
    face_cascade.detectMultiScale(resized_gray_img,
                                    faces, 1.1, 2,
                                    0, Size(20,20));
    for(size_t i = 0; i < faces.size(); i++)
    {
    //obtain top-left and bottom-right point of face
        p_tl.x = st_pt.x + faces[i].x*scale;
        p_tl.y = st_pt.y + faces[i].y*scale;
        p_br.x = st_pt.x + (faces[i].x + faces[i].width)*scale;
        p_br.y = st_pt.y + (faces[i].y + faces[i].height)*scale;
    }
    if(faces.size() > 0)
    {
        cout << "Face␣detected..." << endl;
        return true;
    }
    return false;
}
```

Once the detection is complete, the algorithm checks the number of `faces`. If it's size is larger than 0, that means the detection was successful and at least one face was detected in the image. With the help of the `st_pt`, what determines the top-left point of the ROI window in the base image, the function also returns the position of the detected faces. The coordinates are then used in the main function to draw rectangle around the face, showing their actual position. Fig. 3.3 shows an example image for a detected face.

If the number of detected faces reach at least one, the `detectFace` function returns a true value and leaves a message in the terminal. One can gain more information about the described function on the website of the stated tutorial.
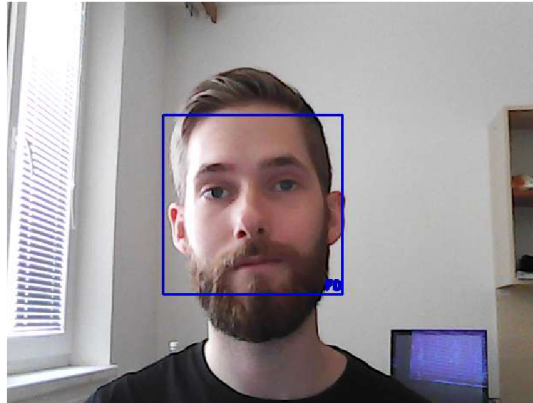
Fig. 3.3: Example for detected frontal face

3.5 Face detection features

As it was mentioned, the function for object detection requires a relatively long time for scanning the input image, keeping the whole algorithm busy. The `extractROI` function's purpose is to limit the attention of the object detecting algorithm to regions of interest based on the ongoing motions, reducing the overall area size what needs to be scanned. As a result, the algorithm is not searching for faces until something moves before the camera. This kind of solution is effective in terms of computational time, however it may not be suitable for every situation. For example, a man is approaching towards the camera and stops a few meters before it. The system could detect his face in motion, but since the person is not moving, the algorithm stops and misses the potential to capture the closest images about him.

The problem is solved by the following way. The regions of interest selected by the `extractROI` function are one by one scanned for faces. An additional structure is defined, where the information about the detected faces are saved to, including their position in the actual image. If a new face is found, the algorithm searches for a free slot in the structure and saves the required information about it. This process is thoroughly described in Alg. 3.3

Algorithm 3.3: Detect faces in extracted ROI

```
1    for q to number of rectangles
2    detect face in actual rectan[q]
3    if face is detected
4     initialize overlap to false
5     while slot of dFaces[o] is not free
6      increment o
7     endwhile
8     sample the area of rectan[q]
9
10    for t 0 to size of face_memory
11     if slot of dFaces[t] is occupied
12      for c to number of samples of dFaces[q]
```

37

```
13        for  d  to  number  of  samples  of  dFaces[q]
14          if  dFaces[t]  c  sample  coordinates  are  equal  to
15        dFaces[t]  d  sample  coordinates
16          dFaces[o]  slot  is  not  occupied
17            set  overlap  to  true
18          endif
19        endfor
20      endfor
21      endif
22    endfor
23
24    if  overlap  is  true
25      set  dFaces[o]  occupied  parameter  to  true
26      set  dFaces[o]  top−left  point  to  rectan[q]  top−left  point
27      set  dFaces[o]  bottom−right  to  rectan[q]  bottom−right  point
28      copy  image  of  rectangle  to  dFaces[o]  img  matrix
29    endif
30    endif
31  endfor
```

An additional part of the code Alg. 3.4 enables to track an appeared face continuously after it is detected. The position of this face is loaded and the area is further scanned in the next iteration of the infinite loop. This allows to observe the given face regardless the person is moving or not. In the case if the face is found in the next iteration too, the position of the corresponding face is refreshed and the loop goes on until the face is present. When the mentioned face disappears, their slot in the structure is released, allowing new face to occupy it.

Algorithm 3.4: Further detect face in the area of previously detected faces

```
1   for  q  to  size  of  face_memory
2     if  dFaces[q]  is  occupied
3       detect  face  in  the  dFaces[q]  img
4       if  face  is  detected
5         set  dFaces[q]  occupied  parameter  to  true;
6         refresh  the  top−left  point  of  dFaces[q]  by  the  detected  face
7         refresh  the  bottom−right  point  of  dFaces[q]  by  the  detected  face
8         resample  the  area  of  dFaces[q]
9         copy  image  of  the  detected  face  to  dFaces[q]  img  matrix
10      else
11        set  dFaces[q]  occupied  parameter  to  false
12        resample  the  area  of  dFaces[q]  with  zeros
13      endif
14    endif
15  endfor
```

After a new face is detected, their position is compared with other faces listed in the structure. If the position of a new face is identical or overlap with an already enlisted face, that means they are identical and the found one is already observed by the algorithm. In every other case the newly detected face belongs to a new person, creating an individual profile in the structure and saving the information about them.

```
1   for q to size face_memory
2     if dFaces[q] is occupied
3       draw rectangle around face
4     endif
5   endfor
```

Alg. 3.5 includes a pseudo-code part, which is responsible to show the detected faces on the screen, drawing rectangles around each of them.

3.6   Saving video sequence

To ignore the uneventful parts of the processed video, the Raspberry focuses only on video sequences where a motion was detected. A detected motion triggers the saving process introduced in Alg. 3.6, making a copy of the actual camera feed and saving them as a video file. The video saving feature is active while an object is in motion and persists 5 more seconds after the last motion was detected. The main function includes a constructor initialized for the video writing with the attributes of the output video file including its size, fps and video codec. The `MJPG` code stands for a motion-jpeg codec.

```
video_name = "Videos/Motion0.avi";
VideoWriter oVideo(video_name,CV_FOURCC('M','J','P','G'),
                   fps, Size(frame_width,frame_height),true);
```

To separate the video footage based on the time and date of detected motions, new files are created for different events. However, to do not distinguish really close events, a 5 seconds control period is added after each one. After the detected motion ceases, the algorithm continues to save the captured images as a video file for 5 more seconds. If a new motion is detected under this time period, the video saving continues.

The control period is realized through the incrementation of `save_counter` variable after the video saving begun. While the motion is still active, the variable is reset in each loop. If the motion stops and the control period runs down, the actual video is released and the next detected motion entails the creation of a new video file.

```
1   if motion was detected
2     if openNewFile is true
3       generate new name for video file
4       constructor for new video file
5       set openNewFile to false
6     endif
7     set saveVid to true
```

```
 8    set save_counter to 0
 9    endif
10    if saveVid is true
11    save_counter ++
12    add captured frame to video file
13    if save_counter > 5 times fps
14      release video file
15      set saveVid to false
16      set openNewFile to true
17      set save_counter to zero
18    endif
19    endif
```

The `videoNameGenerator()` function's only purpose is to create an individual name for each video returning them to the main function.

```
void videoNameGenerator( &videoName)
{
    char name_buffer[50];
    static char video_counter = 0;
    string format = ".avi";
    video_counter++;
    sprintf(name_buffer, "Videos/Motion%d",video_counter);
    videoName = name_buffer + format;
}
```

3.7 Remote access

This section deals with the design and realization of remote access to the embedded surveillance system, which ensures a permanent insight to the actual events. This kind of access allows the user to reach the system from any place where an internet connection is available.

The remote access to the Raspberry Pi is realized by a web server installed on the device. This server hosts a simple web page displaying the actual information, allowing the users to browse and watch saved videos or take a look to the actual events through a live stream.

The *Apache 2 web server* had been chosen to host the described web page. The official Raspberry homepage [16] contains a detailed installation manual of the web server and others tools like *PHP* and *MySQL*, which are also required for building the web page. After the installation is complete, a default HTML file appears in the following location:

```
/var/www/html/index.html
```

3.7.1   Opening video files

The first feature of the web page is to play videos saved by the surveillance system, which are captured in the case of detected motions. Since the OpenCV library can save the files only in `.avi` format, the videos have to be converted into .mp4, what is one of the formats supported by HTML. To accomplish such conversion, FFmpeg [17] is installed on the device, which is a free software project specialized for handling multimedia data. With the help of this program the conversion process can be executed directly with a single terminal command.

The Raspbian Jessie operational system is based on Debian Linux version 8, which not supports the installation of FFmpeg through a single terminal command. Because of this, the program was installed on the Raspberry Pi following an installation tutorial [18]. The only change was made in the configuration part, where the *–enable-shared* prefix has to be removed, because it caused the program to crash on.

```
./configure --enable-gpl --enable-postproc --enable-swscale
--enable-avfilter --enable-libmp3lame --enable-libvorbis
--enable-libtheora --enable-libx264 --enable-libspeex
--enable-pthreads --enable-libopenjpeg --enable-libfaac
--enable-nonfree
```

The conversion tool is now installed and it has to be run each time a new video file is saved by the surveillance application. The C++ source code part responsible for saving the video files is completed with the following rows.

```
static char captCounter=0;

sprintf(command, "ffmpeg -y -i Videos/Motion%d.avi -preset
ultrafast ~/../../var/www/html/videos/Capture%d.mp4",
captCounter, captCounter);
captCounter++;

system(command);
```

This part runs the corresponding command in the terminal each time a new video file is saved. The converted video file is at the same time moved to a new folder, where the HTML file has access to it.

3.7.2 Listing video files

Separate events captured by the surveillance system are saved as individual video sequences. The converted videos are moved to the `var/www/html/videos` directory what has to be inspected to list it's content on the web page. A short php script developed with the help of PHP 5 tutorial available on [19] does all the work. The function examines all existing files in the given directory, saving their name, file size and the date of their last modification.

```php
<?php
function getFileList($dir)
{
  // array to hold return value
  $retval = array();
  // open pointer to directory and read list of files
  $d = @dir($dir) or die("getFileList: Process failed!");
  while(false !== ($entry = $d->read())) {
    // skip hidden files
    if($entry[0] == ".") continue;
    if(is_readable("$dir$entry"))
    {
      $retval[] = array
      (
        "name" => "$dir$entry",
        "size" => filesize("$dir$entry"),
        "lastmod" => filemtime("$dir$entry")
      );
    }
  }
  $d->close();

  return $retval;
}
?>
```

The information gathered about the video files are listed in a constructed table among two images in the row. Clicking on these images calls a specific javascript function depending on the clicked image. The event is either playing the selected video file or to delete it from the directory and from the table.

```php
<?php
$dirlist = getFileList("./videos/");
foreach($dirlist as $file)
{
   if(!preg_match("/\.mp4$/", $file['name'])) continue;
      echo "<tr>\n";
      echo "<td><img src=\"img/play.png\" width=\"64\"
     alt=\"Image\" onclick=\"playFile(this)\"></td>\n";
      echo "<td>{$file['name']}</td>\n";
      echo "<td>{$file['size']}</td>\n";
      echo "<td>",date('r', $file['lastmod']),"</td>\n";
      echo "<td><img src=\"img/delete.png\" width=\"64\"
     alt=\"Image\" onclick=\"deleteFile(this)\"></td>\n";
         echo "</tr>\n";
   }
?>
```

The structure of the table can be seen on Fig. 3.4.

Play	Name	Size [B]	Last Modified	Delete
▶	./videos/Capture2.mp4	1135188	Tue, 18 Apr 2017 18:31:15 +0200	🗑
▶	./videos/Capture1.mp4	849370	Tue, 18 Apr 2017 18:31:01 +0200	🗑
▶	./videos/Capture0.mp4	470884	Tue, 18 Apr 2017 18:30:45 +0200	🗑

Fig. 3.4: Table structure showing directory content

3.7.3 Video player window

To play the selected video file on the web page, a window has to be constructed with specific parameters. The dimensions of the window are the same as the video files and the source of the video is set to zero by default. The figure showing the video player window can be seen on Fig. 3.5

```
1  <video width="640" height="480" controls>
2    <source id="videoSource" src="" type="video/mp4">
3    Your browser does not support the video tag.
4  </video>
```

Clicking on the correct image showing a play button located in the table calls an event, changing the source of the video player. The javascript function simply copies the video file name located in the table next to the image, and uses it as a link to the video.

```
1  function test(element)
2  {
3    videoTitle = $(element).parent().next()[0].innerText;
4    $(element).next()
5    $("#videoSource").attr("src", videoTitle);
6    $("#divVideo video")[0].load();
7  }
```
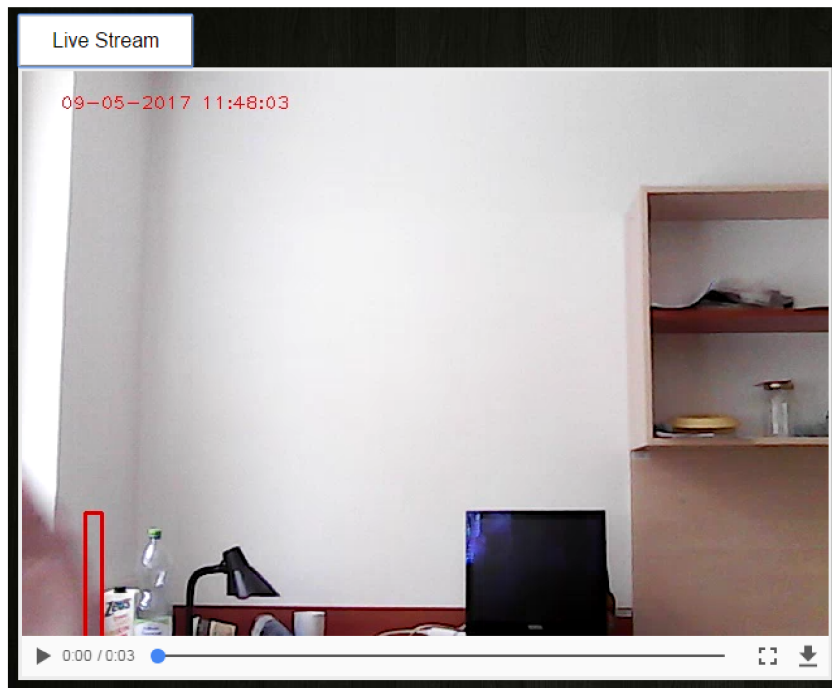

Fig. 3.5: Window structure of video player

44

3.7.4 Live stream

Adding a live stream window to the web page would require additional space. A window is created with the same dimensions as the video player, but with a different identification name. Also a button is added above the window, what simply calls a javascript function responsible to change the display attribute of the video player window and the live stream window. This means pressing this button would hide the element for video playing and show the element for live streaming, and another press of the same button would exchange the effect. The window structure is shown in Fig. 3.5

```
1  function ObjToggle() {
2    var x = document.getElementById('videoDiv');
3    var y = document.getElementById('streamDIV');
4    if (x.style.display === 'none')
5    {
6      x.style.display = 'block';
7      y.style.display = 'none';
8    }
9    else
10   {
11     x.style.display = 'none';
12     y.style.display = 'block';
13   }
14 }
```

The video sequences saved by the Raspberry Pi do not contain any audio channels. Since these video files are just arrays of captured frames, the live streaming feature could be realized by saving the actual image to the HTML directory each time a new frame is captured. The image is then displayed in the window, refreshing it's content when the image is overwritten. The javascript function responsible for refreshing the image can be seen below:

```
1  function init()
2  {
3    var canvas = document.getElementById("canvas");
4    var context = canvas.getContext("2d");
5    img = new Image();
6    img.onload = function()
7    {
8      canvas.setAttribute("width", img.width)
9      canvas.setAttribute("height",img.height)
10     context.drawImage(this, 0, 0);
11   };
12   refresh();
13 }
```

```
14
15  function refresh()
16  {
17    img.src = url + "?t=" + new Date().getTime();
18    setTimeout("refresh()", refreshInterval);
19  }
```

3.7.5   System authentication

The Apache 2 web server can be used to restrict the access to sections of the site,
setting up a basic authentication process to prevent unauthorized users reach the
web page. This can be done as described in the following tutorial [20]. To authenti-
cate users, the web server requires a password file containing user names and their
assigned passwords. The file can be created with the next command in the terminal:

```
sudo htpasswd −c /etc/apache2/.htpasswd admin
```

After this command, the web server automatically asks to supply and confirm a
password for the specific user. As the next step, the apache configuration file has
to be modified and the password protection has to be added to the virtual host file.
The default virtual host file is then edited to ask for a password if a user tries to
reach the restricted directory.

```
sudo nano /etc/apache2/sites−enabled/000−default.conf
```

The virtual host file has the following content after editing:

```
<VirtualHost *:80>
  ServerAdmin webmaster@localhost
  DocumentRoot /var/www/html
  ErrorLog ${APACHE_LOG_DIR}/error.log
  CustomLog ${APACHE_LOG_DIR}/access.log combined

  <Directory "/var/www/html">
    AuthType Basic
    AuthName "Restricted Content"
    AuthUserFile /etc/apache2/.htpasswd
    Require valid−user
  </Directory>
</VirtualHost>
```

Once the file is saved and closed, the apache server has to be restarted. With the
next start-up the directory is password protected.

```
sudo service apache2 restart
```

4 EXPERIMENTAL RESULTS

4.1 Motion detection

4.1.1 Testing of rectangle merge

The following section gives an insight to the behaviour of the used motion detection algorithm and it's features. The functions are applied to several video surveillance recording, highlighting the indicated motions in them and showing the difference between the algorithm with and without the rectangle merge feature. The video examples include moving people in a metro station, passing cars on a street and customers at market checkout [21] [22].

(a)	(b)	(c)
(d)	(e)	(f)
(g)	(h)	(i)
(j)	(k)	(l)
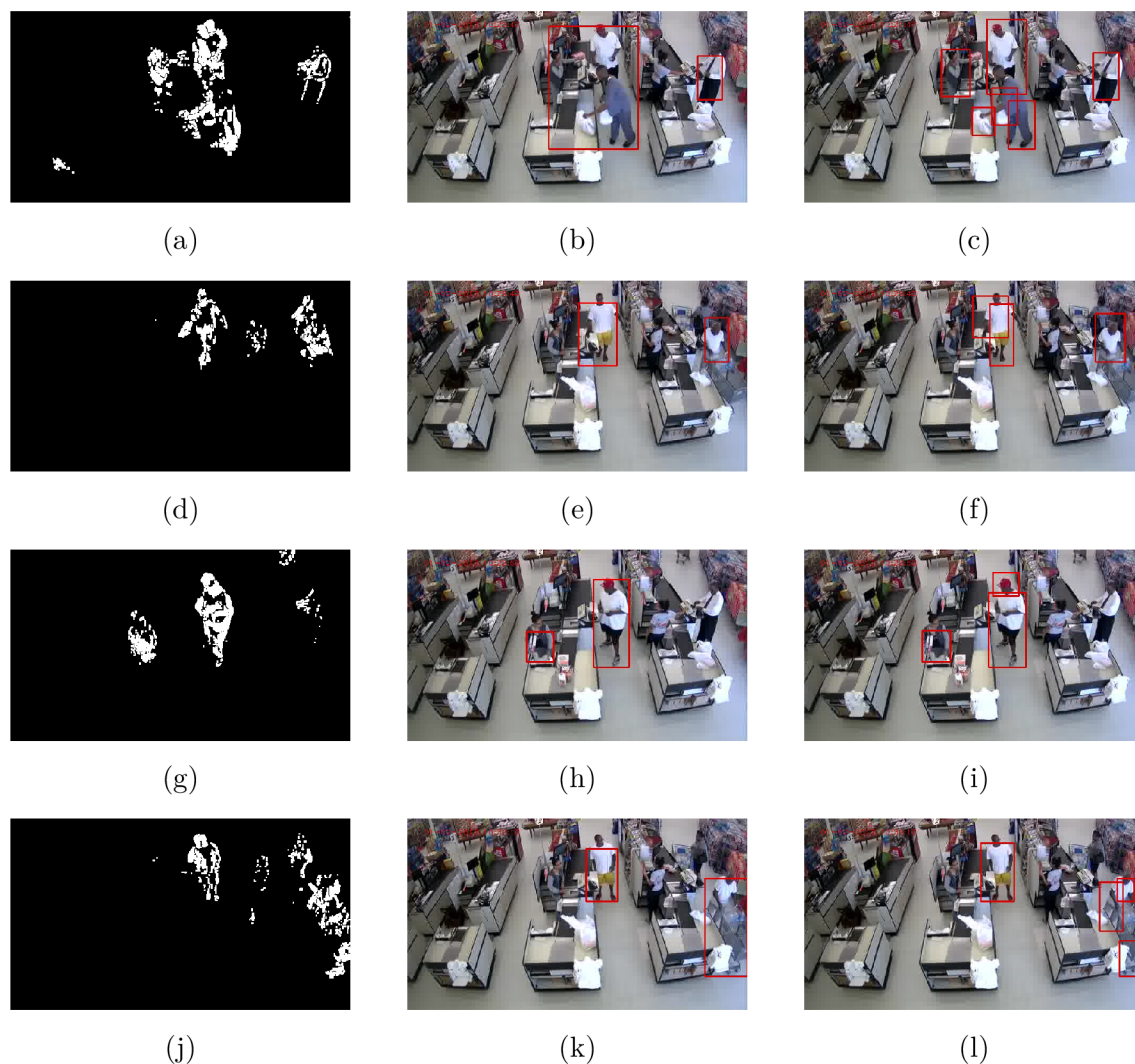
Fig. 4.1: Motion detection: market checkout

Since the thresholded image of a moving object can consists of more blobs, the ROI selecting algorithm will sense them as individual objects, splitting them into more parts. This effect is significant on close objects, because the camera can perceive more details about them. Complex objects with stronger contours cause the threshold function to return more blobs.

(a)   (b)   (c)

(d)   (e)   (f)

(g)   (h)   (i)

(j)   (k)   (l)
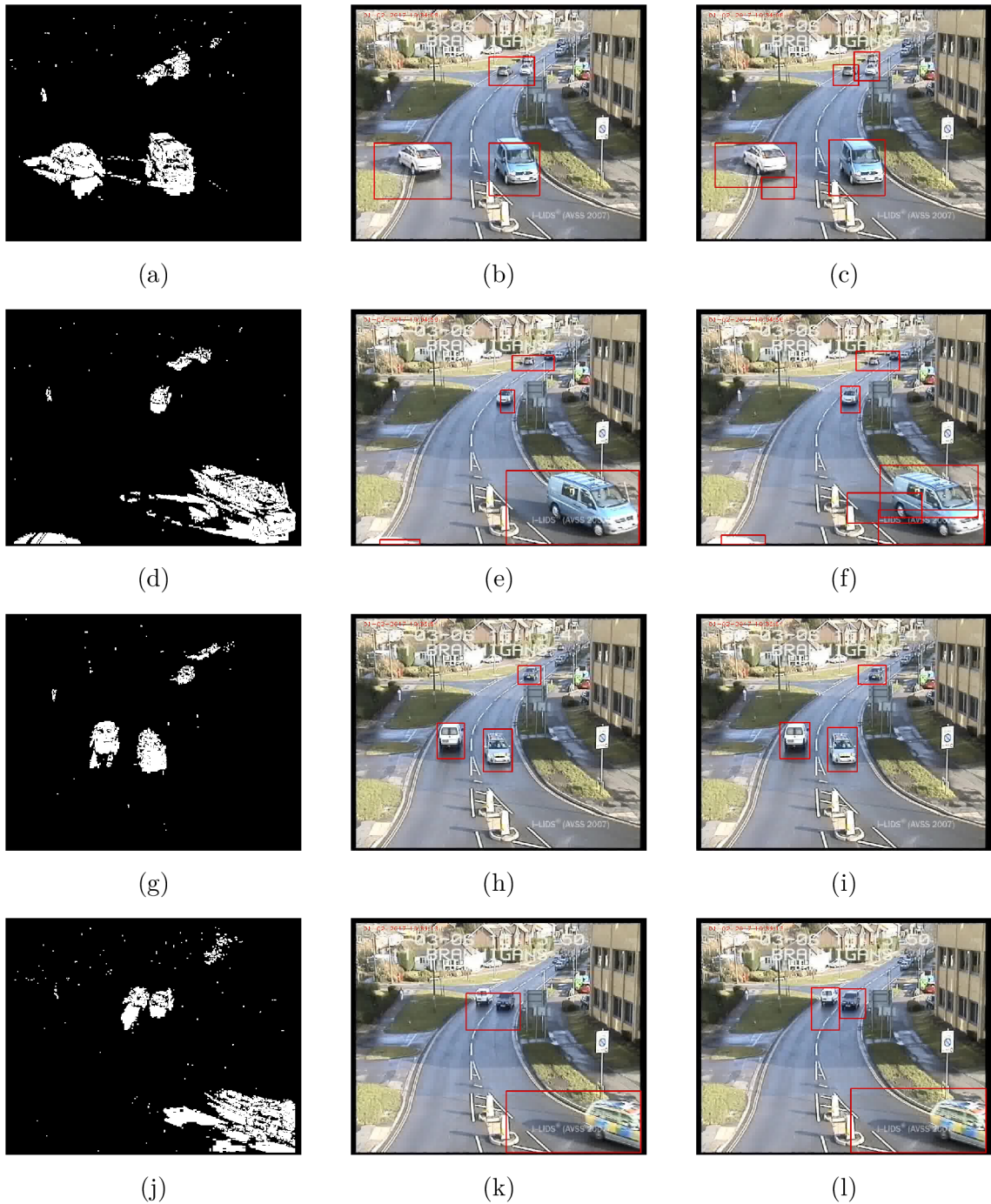
Fig. 4.2: Motion detection: passing cars on a street

This feature is corrected with an additional function, what merges together rectangles with common areas. The results are shown in the following figures. The left image in Fig. 4.1, Fig. 4.2 and Fig. 4.3 always show the threshold image from the motion detection algorithm, the center image shows the detected motions with rectangle merge and the right image shows the detected motions without the feature.

(a)	(b)	(c)
(d)	(e)	(f)
(g)	(h)	(i)
(j)	(k)	(l)

Fig. 4.3: Motion detection: metro station

4.2 Face detection

4.2.1 Comparison of two type of cascades

To acquire more experience in the field of object detection, the section also contains the comparison of two cascade classifiers: one with Haar features and one with local binary patterns. The OpenCV libraries include both classifiers for the detection of faces, which are compared in terms of computational time required for scanning an input image. It also contains an experiment testing the effectiveness of face detection based on the position and complexity of one face. Table 4.1 contains the computational times used by the `detectMultiScale` function for scanning images with given sizes.

Tab. 4.1: Computational time of used cascades

Resolution	640 x 480			Based on ROI		
Comp. Time	min	max	avg	min	max	avg
	[milliseconds]					
Haar classifier	91	140	115.5	1.5	33	17.3
LBP clasifier	20	48	34	0.3	9	4.7

Based on the measured results, the object detection algorithm using LBP cascade classifiers is 3 times faster than the Haar features based one.

4.2.2   Effectiveness of frontal face detection

Considering the results from the previous measurement, the LBP classifier is selected and further observed. In the next step, the effectiveness of the classifier is tested. A given face is tilted to several position while the result from the classifier is observed. The result images are shown in Fig. 4.4.
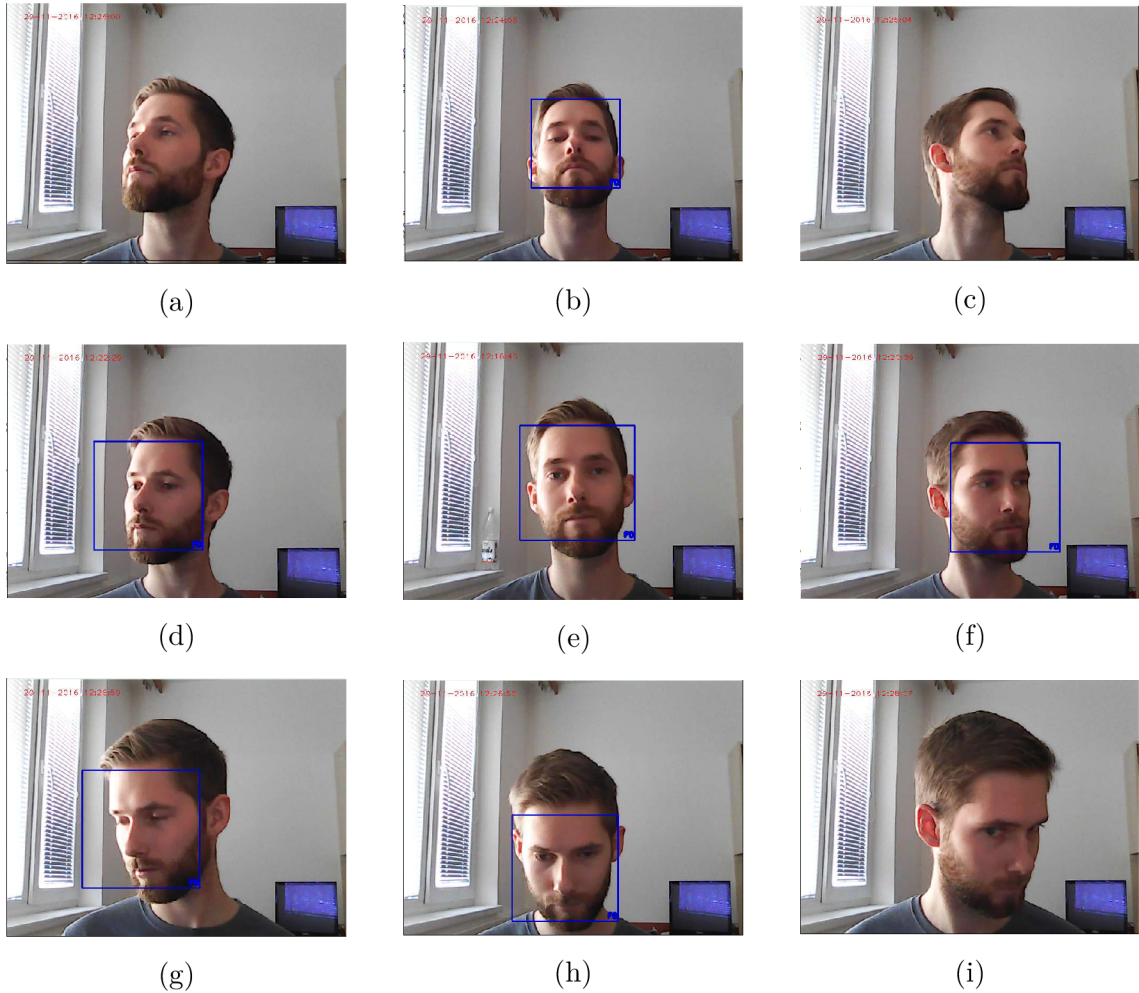

(a)　　　　　　　　(b)　　　　　　　　(c)

(d)　　　　　　　　(e)　　　　　　　　(f)

(g)　　　　　　　　(h)　　　　　　　　(i)

Fig. 4.4: Results of LBP face detection on average face

The same procedure is repeated with adding new components to the same face, making it more complex and difficult to detect. These modifications are including facial hair, glasses (Fig. 4.5), a cap or all of them (Fig. 4.6).

(a)

(b)

(c)

(d)

(e)

(f)

(g)

(h)

(i)
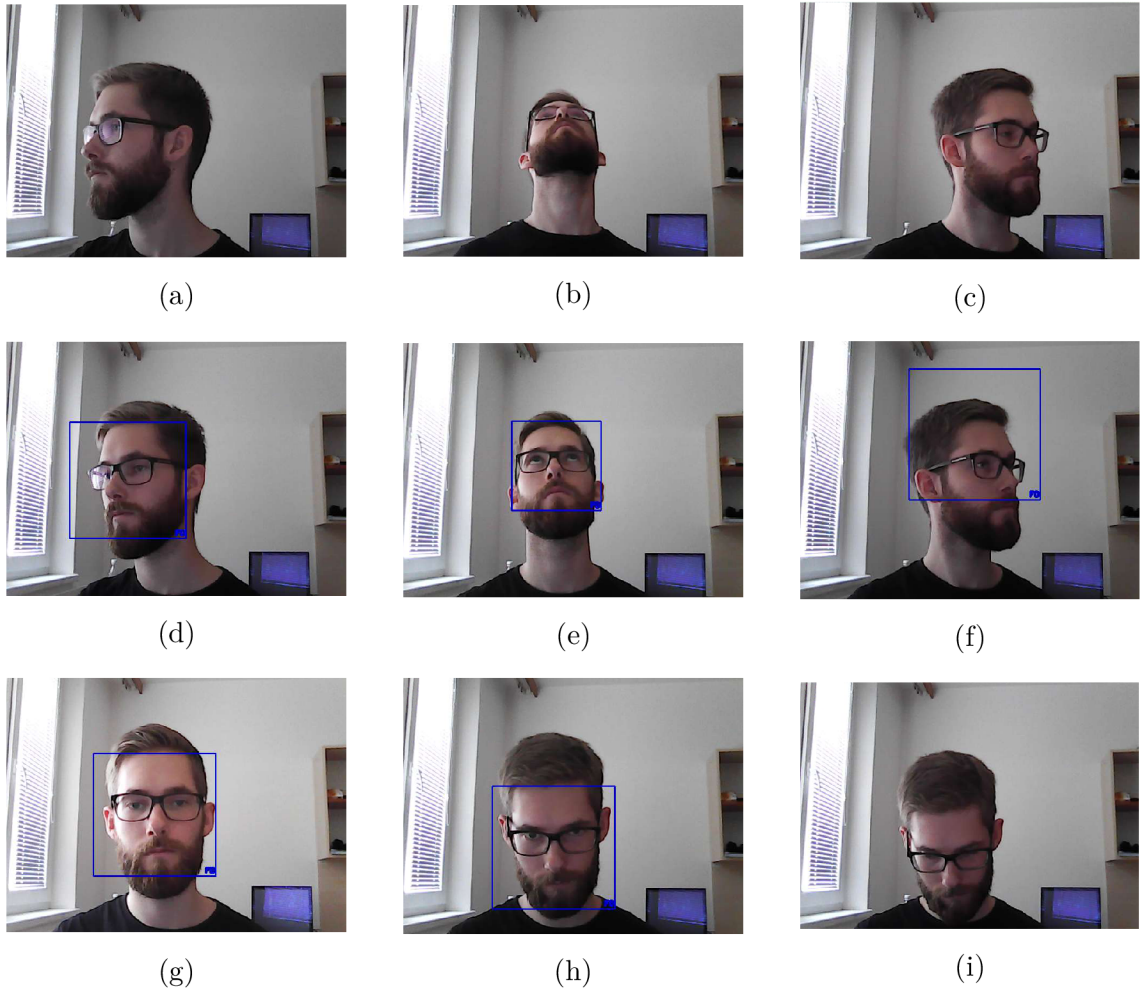
Fig. 4.5: Results of LBP face detection on face with glasses

In the case a person is wearing glasses, the results show that the detection returns a negative value, if the angle between the face and the camera is larger than 45 °. The same rule applies if a person wears a cap, however, the detector shows better results if the person's head is tilted up or down. If one is wearing both glasses and a cap, the algorithm detects the person's face only if he looks straight to the camera.
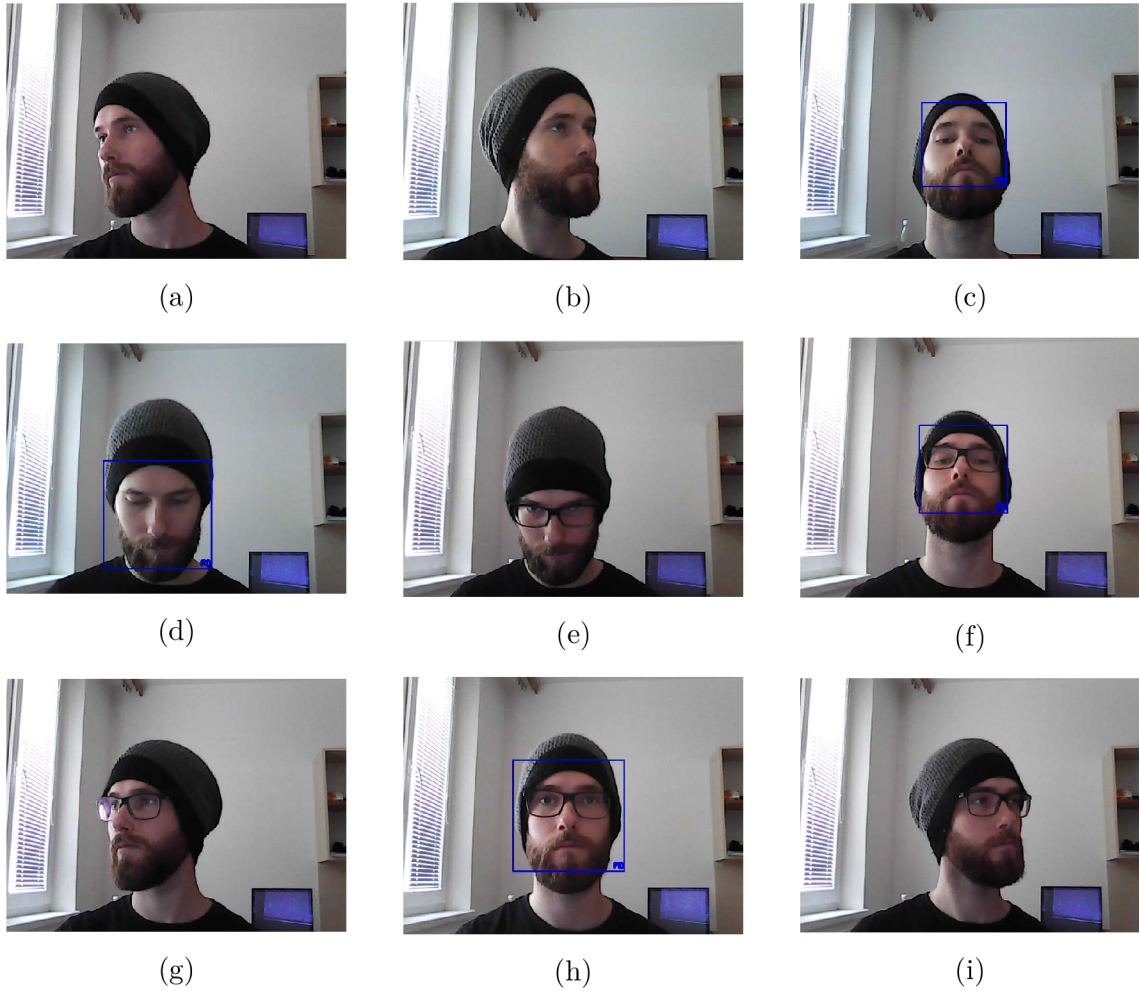
Fig. 4.6: Results of LBP face detection on face with cap(a-d) and glasses(e-i)

4.3 Hand detection

This section deals with the testing of custom trained classifier for open hand and palm detection. The information about the training process can be found in the section `Cascade classifier training`. The amount of original positive samples used for the training process was relatively small, including only 100 images of open hands in different position. The number of positive samples was artificially increased, however, using these kind of positive samples resulted in a classifier with lesser quality. This means it returns false positive results in the area of complex background, or return negative result if the color of the detected hand and the background is similar. The testing database includes a set of images about a given hand tilted to several angles in various distances.

| (a) | (b) | (c) |

| (d) | (e) | (f) |
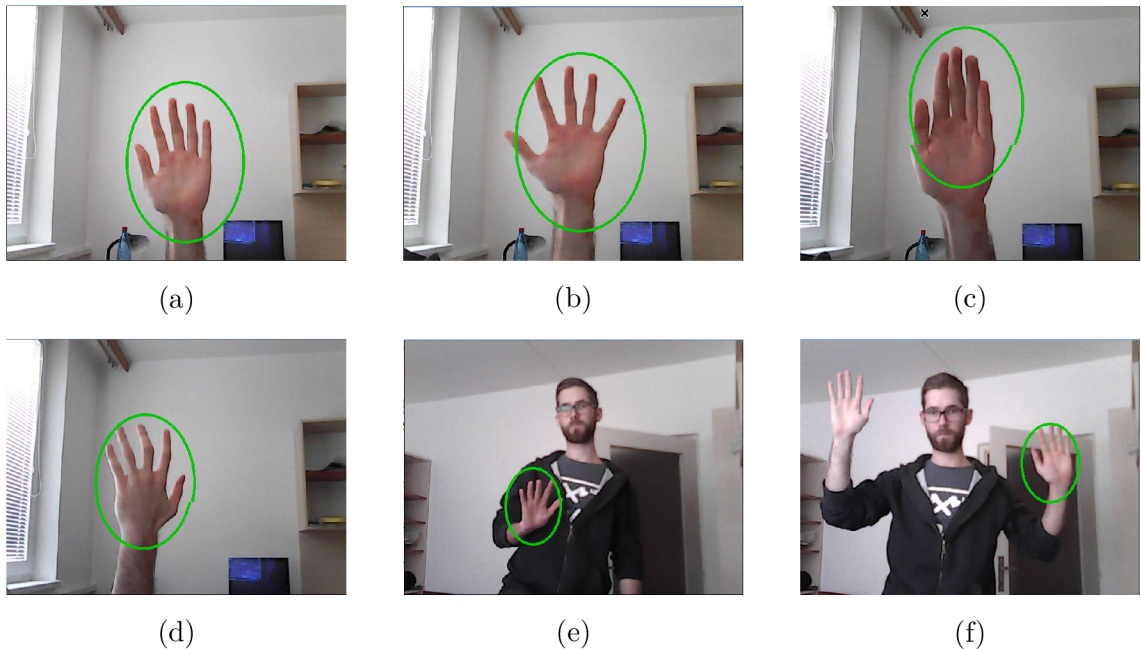
Fig. 4.7: Results of LBP hand detection

Obtaining at least 2 000 - 3 000 real positive samples about open hands would significantly increase the quality of the classifier. Such hand detector could be used for example in a system which could recognize the hand gestures of the sign language. An open hand shown to the camera would act as an initialization gesture, locating the position of the observed hand.

4.4 Execution time of used algorithms

This section includes a measurement of the required computational time of used algorithms and functions on the Raspberry Pi 3 B. The used algorithms for motion and object detection were originally designed to work effectively on personal computers with higher computational power than mini-computers like Raspberry Pi. Because of this, the project was also dealing with the optimization of these algorithms and functions. Table 4.2 includes the measured execution time of the listed algorithms, functions and program parts, showing the slowest and fastest parts of the system, also the overall FPS.

Tab. 4.2: Computational time of program parts in ms

Computation time in ms	min	avg	max
Image capture + conversion to grayscale	5	8	13
Motion detection	20	25	30
ROI selection and extraction	3	6	10
Face detection depending on the area of ROI	3	15	30
Video saving	20	30	40
Hand detection	30	40	60
Sum without HD	51	84	123
Overall FPS	8	12	16

The Raspberry Pi 3 B processes an average of 10 frames per second when all functions are active including the hand detection. Deactivating additional features like hand or face detection further increases the speed of the system to a maximum of 15 FPS, enabling only features responsible for motion detection and video saving. Increasing the number of detected objects in parallel would further slow down the system, with 2-3 FPS for each classifier.

5 CONCLUSION

This master thesis was dealing with with the design and realization of an embedded surveillance system for in-door usage. The system was optimized to work effectively on the Raspberry Pi 3 on-board mini-computer, including features and functions commonly used in modern camera systems like motion and object detection, and also recording video files in the case of a detect events.

The project includes a self developed source code for a new approach to the frame difference based detection method. Since the basic method separates the detected objects in motion into more parts due to the problem described in section `Selecting region of interest`, the extracted regions of interest could not serve as input images for the object detecting algorithm. The newly added algorithm merged together these parts belonging to the same object, making possible to extract these image parts and feed them to object detecting algorithm. With this method, the area scanned by face detection could be shrinked into these regions of interest, instead of the whole picture from the camera feed, thus reducing the execution time and increasing the speed of the system.

On top of the assignment, the thesis also describes how to train a classifier for the detection of any desired object. In this case, the tracked objects were hands and palms. The purpose of this custom trained classifier was experimental, however it could be used as basis for a system recognizing hand gestures of the sign language.

Because the video procession requires relatively lot of computation power, one of the hardest task was to optimize the system to work with a proper frame per second ratio on the Raspberry Pi. The system in it's final state can process 12 frames per second when all functions are active except the hand detection, the execution time required for the individual features and function are listed in Tab. 4.2.

The remote access enables the user to edit and browse among saved video files and to take a look at the actual events seen by the Raspberry Pi. The web page which makes possible the above mentioned function can be reached only within local host.

The designed system could be used as a compact surveillance system for home usage. The camera installed above an entrance could warn the residents if someone was coming. With additional effort the system could be trained to recognize individual faces of the users allowing them access to the house automatically.

BIBLIOGRAPHY

[1] SOMHORST, Maarten. *Multi-camera video surveillance system* Master's thesis [online]. 2012 [cit. 8.10.2016] Available from URL: <`http://www.kbs.twi.tudelft.nl/docs/MSc/2012/Somhorst/thesis.pdf`>.

[2] SZELISKI, Richard. *Computer Vision: Algorithms and Applications* Electronic version, 2010 Springer Available from URL: <`https://szeliski.org/Book/`>.

[3] LAGANIERE, Robert. *OpenCV 2 Computer Vision Application Programming Cookbook.* Published by: Packt Publishing Ltd. May 2011, 287 p, ISBN 978-1-849513-24-1

[4] *Open Source Computer Vision.* [online]. 2006 [cit. 11.10.2016] Available from URL: <`https://www.opencv.org/`>.

[5] SINGALA, Nishu. *Motion Detection Based on Frame Difference Method* International Journal of Information and Computation Technology 2014 [online]. [cit. 20.11.2016] Available from URL: <`http://www.ripublication.com/irph/ijict_spl/ijictv4n15spl_10.pdf`>.

[6] *RASPBERRY PI FOUNDATION.* [online]. 2016 [cit. 10.10.2016] Available from URL: <`https://www.raspberry.org/`>.

[7] *Haar-like features* Wikipedia article [cit. 1.11.2016] Available from URL: <`https://en.wikipedia.org/wiki/Haar-like_features/`>.

[8] VIOLA, Paul and JONES, Michael. *Rapid Object Detection using a Boosted Cascade of Simple Features* Accepted Conference on Computer Vision and Pattern Recognition 2001 [online]. [cit. 1.11.2016] Available from URL: <`https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf`>.

[9] Shengcai Liao, Xiangxin Zhu, Zhen Lei, Lun Zhang, and Stan Z. Li. *Learning Multi-scale Block Local Binary Patters for Face Recognition* Center for Biometrics and Security Research and National Laboratory of Pattern Recognition 2007 [online]. [cit. 2.11.2016] Available from URL: <`http://www.cbsr.ia.ac.cn/users/lzhang/papers/ICB07/ICB07_Liao.pdf`>.

[10] Ball, Thorsten. *Train Your Own OpenCV Haar Classifier* Coding Robin 2013 [online]. [cit. 10.11.2016] Available from URL: <`http://coding-robin.de/2013/07/22/train-your-own-opencv-haar-classifier.html`>.

[11] Seo, Natoshi. *Tutorial: OpenCV haartraining (Rapid Object Detection With A Cascade of Boosted Classifier Based on Haar-like Features)* Tutorial

[online]. [cit. 10.11.2016] Available from URL: <http://note.sonots.com/SciSoftware/haartraining.html>.

[12] *Open Source Computer Vision.Cascade Classifier Training* [online]. [cit. 8.11.2016] Available from URL: <http://docs.opencv.org/trunk/dc/d88/tutorial_traincascade.html>.

[13] *Open Source Computer Vision.Core modules* [online]. 2006 [cit. 11.11.2016] Available from URL: <http://docs.opencv.org/2.4/modules/core/doc/core.html>.

[14] *Open Source Computer Vision.Image processing* [online]. 2006 [cit. 11.11.2016] Available from URL: <http://docs.opencv.org/2.4/modules/imgproc/doc/imgproc.html>.

[15] *Open Source Computer Vision.Cascade Classifier* [online]. 2014 [cit. 12.11.2016] Available from URL: <http://docs.opencv.org/2.4/doc/tutorials/objdetect/cascade_classifier/cascade_classifier.html>.

[16] Raspberry Pi Learning Resources *Build A Lamp Web Server With WordPress* [online]. [cit. 20.11.2016] Available from URL: <https://www.raspberrypi.org/learning/lamp-web-server-with-wordpress/>.

[17] FFmpeg *A complete, cross-platform solution to record, convert and stream audio and video.* [online]. [cit. 18.04.2017] Available from URL: <https://ffmpeg.org>.

[18] AssetBank *Installing Ffmpeg on Debian GNU/Linux Version 8.0 (Jessie)* [online]. [cit. 18.04.2017] Available from URL: <https://www.assetbank.co.uk/support/documentation/install/ffmpeg-debian-squeeze/ffmpeg-debian-jessie/>.

[19] W3Schools *PHP 5 Tutorial* [online]. [cit. 19.04.2017] Available from URL: <https://www.w3schools.com/php/default.asp>

[20] DigitalOcean *How To Set Up Password Authentication with Apache on Ubuntu* [online]. [cit. 19.04.2017] Available from URL: <https://www.digitalocean.com/community/tutorials/how-to-set-up-password-authentication-with-apache-on-ubuntu-14-04>.

[21] *Advanced Video and Signal based Surveillance* [online]. [cit. 30.03.2017] Available from URL: <http://www.eecs.qmul.ac.uk/~andrea/avss2007_d.html>.

[22] YouTube *HD-SDI 1080P Hi Definition Megapixel Security Camera and Super Market Checkout Sample 1* [online]. [cit. 30.03.2017] Available from URL: `<https://www.youtube.com/watch?v=-8zyEwAa50Q>`.
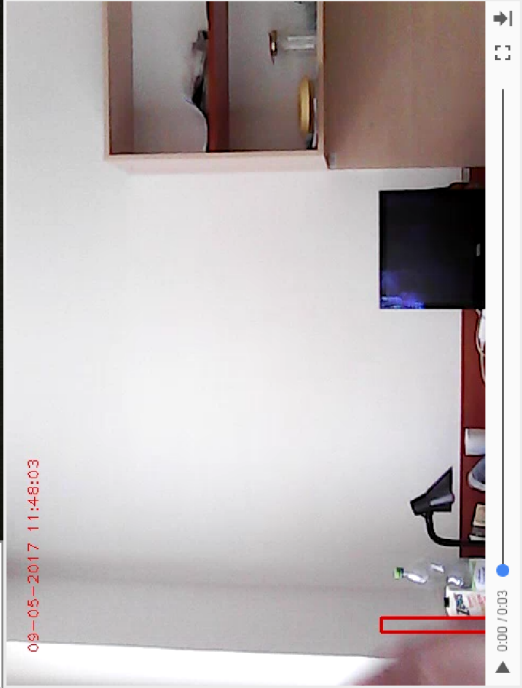
LIST OF APPENDICES

A   WEB PAGE