TAMPEREEN TEKNILLINEN YLIOPISTO

ANTTI EIVOLA

HOME ENTERTAINMENT DEVICE DETECTION USING MOBILE PHONE

Master of Science Thesis

Examiner: professor Irek Defee
Topic and examiner approved in the Computing and Electrical Engineering Faculty Council meeting on 3 November 2010.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Signaalinkäsittelyn ja tietoliikennetekniikan koulutusohjelma
EIVOLA, ANTTI: Kodin viihdelaitteiden tunnistaminen matkapuhelimella
Diplomityö, 52 sivua
Syyskuu 2012
Pääaine: Signaalinkäsittely
Tarkastaja: professori Irek Defee
Avainsanat: Konenäkö, hahmontunnistus, mobiilisovellus, avoin lähdekoodi

Mobiiliteknologian ja konenäön kehitys avaa uusia käyttökohteita niiden hyödyntämiselle. Avoimen lähdekoodin mobiililaitteiden sovelluskehitysalustat sekä konenäkökirjastot tarjoavat mahdollisuuden kehittää sovelluksia suhteellisen pienellä vaivalla. Tässä diplomityössä esitellään muokattu versio tunnetusta hahmontunnistusprosessista sekä kaksi sitä hyödyntävää mobiilisovellusta. Tämän työn tarkoituksena on selvittää, voidaanko avoimen lähdekoodin työkaluilla kehittää kiinnostavia mobiilisovelluksia ja esittää menetelmä, jolla mahdollisia valmiiden kirjastojen puutteita voidaan korvata.

Työ jakaantuu neljään osaan. Ensimmäinen osa käsittelee alan kirjallisuudesta tunnettua hahmontunnistusmenetelmää ja toinen esittää parannuksia tähän menetelmään. Kolmannessa osassa esitellään kaksi tätä työtä varten toteutettua mobiilisovellusta, jotka hyödyntävät aiemmin kuvatun menetelmän muokattua versiota. Näiden sovellusten avulla käyttäjä voi tunnistaa sekä ohjata kodin viihdelaitteita käyttäen kameralla varustettua matkapuhelinta. Viimeisenä arvioidaan esitetyn menetelmän tarkkuutta ja työn muita saavutuksia.

Työssä esitetyt muutokset käsiteltävään hahmotunnistusmenetelmään tuovat siihen lisää tarkastuksia ja muokattu menetelmä suoriutuu testatuissa olosuhteissa valmiina saatavilla olevaa toteutusta paremmin. Työssä esiteltävät mobiilisovellukset tarjoavat esimerkin siitä, mitä mobiililaitteen ja konenäön avulla voidaan saada aikaan. Molemmat sovellukset täyttävät niille asetetut tavoitteet ja niiden avulla voidaan tunnistaa laitteita puhelimen kameran tuottamasta reaaliaikaisesta videosyötteestä. Toinen sovelluksista tarjoaa myös rajapinnan, jolla tunnistettuja laitteita voidaan ohjata.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Signal Processing and Communications Engineering
EIVOLA, ANTTI: Home Entertainment Device Detection Using Mobile Phone
Master of Science Thesis, 52 pages
September 2012
Major: Signal Processing
Examiner: Professor Irek Defee
Keywords: Computer vision, object recognition, mobile software, open source software

Development in mobile technology and computer vision enables new usage scenarios related to these technologies. Open source mobile application frameworks together with open source software libraries for computer vision provide the means for developing applications with relatively low effort. This thesis presents a modified version of a popular object detection method and utilizes this in two separate mobile software implementations. The main goals are to assess if interesting end-user applications can be developed with these open source tools and to propose a possible solution for the parts that are not adequately covered by the readily available software components.

The thesis is divided into four parts. The first part deals with a popular object detection method from the literature and second part proposes improvements for this method. Next the two implementations, using the proposed method, are presented that enable the user to detect and control home entertainment devices using a mobile phone with a camera. The thesis is concluded with an evaluation of the performance of the proposed method and discussion.

The proposed method adds additional restrictive checks to the popular object detection method and, for the concerned scenarios, performs better than the readily available implementation. The two applications implemented for this thesis show an example of what can be achieved with computer vision and a mobile phone. Both applications fulfil the goals set for them and can be used to detect objects seen in the live video feed from the mobile phones camera. One of the implementations also provides an interface to interact with the devices once they are detected.

PREFACE

The process which led to this thesis started from a Demola Innosummer project. In the project we were set up to build a technical demo for a mobile application to detect and control devices with. Even without much prior knowledge we managed to build one and the summer was most educational and, above all, interesting. As I was offered premises to work in and support from the same team which supported us during the summer, I decided to continue the work in form of this thesis. I had just finished the second implementation when I took another job from another city. After this, the thesis became more of a hobby during random evenings after actual work and after our kids fell asleep. Keeping a tight schedule for the thesis project was not something I did, but I would strongly recommend others to do.

I wish to thank Mike Kapitonov for building all but the detection, device control and network parts of the first implementation and also for his valuable expertise on Linux and software development. Also thanks to professor Irek Defee for providing premises where I could continue working with the topic after the Innosummer and to Adrian Hornsby for his fine tips during the Innosummer concerning working methods and available open source tools. Especially I wish to thank Rod Walsh, who provided the initial idea to build a mobile application for device detection and was of big help to get the process running smoothly. Rod´s contagious enthusiasm for the field and the always so happy attitude was contagious. Cheers Rod.

Espoo 14.10.2012

Antti Eivola

TABLE OF CONTENTS

LIST OF FIGURES

1 INTRODUCTION

Modern mobile technology together with computer vision enables the creation of interesting augmented reality solutions. Freely available open source alternatives are available for both mobile application development in general and computer vision software libraries. Together these make it possible for anyone with little programming skills to utilize computer vision in mobile solutions.

The focus of this thesis is on developing a mobile application that enables the user to detect devices from the surrounding environment using a mobile phone with a camera. Then the user would be able to interact with the real world using an augmented reality user interface. For this purpose it was necessary to find a suitable computer vision method for the device detection and to find necessary frameworks and tools to build the actual application for a mobile phone.

The first approach for object detection was to gather data from images showing individual devices and then use this data to detect and recognize the devices from real world images. This approach did not work as well as expected so image data from individual devices was abandoned and the whole image, including the background, was taken into use.

For the actual means for object detection local image features were the starting point for the search for a suitable computer vision method because of the popularity of such methods in the literature. Soon the decision came down to selecting either SIFT [1] or SURF [2]. SIFT seemed to have gained more popularity but SURF was claimed to be faster [2; 3]. As the thesis deals with a mobile application, the computational efficiency was one of the most important aspects to consider. Others were sufficient accuracy and tolerance to changing conditions and also availability of an open source implementation. The open source implementation which was selected did not offer good enough performance without any enhancements. Certain additions to the nearest neighbour matching technique [4] were examined and implemented to improve the performance.

This thesis is structured in such a way that each chapter can be read independently. Each chapter gives a short introduction on the topics covered. Chapter two gives an overview on one method for object recognition from a theoretical point of view. Chapter three presents the proposed object detection method and chapter four describes the implementations from design and implementation perspective. More detailed results along with discussion are presented in chapter five and finally conclusions in chapter six.

2 OBJECT RECOGNITION

This chapter gives an overview on object recognition and presents the theory related to this work. First object recognition based on local image features is discussed briefly in general. Next a similar approach as the one proposed in chapter 3 is presented in more detail. The presentation is started by describing interest point detection, followed by interest point description and finally matching of interest points.

2.1 Object Recognition Process Overview

Object recognition by computer vision can be considered generally as a process where: 1) information is extracted from a given image and; 2) the extracted information is compared against prior knowledge, to; 3) produce meaningful information. This section gives an overview on some approaches how this can be done.
This thesis focuses only on object recognition approaches based on local invariant descriptors. This approach, adopted by Lowe [1], Matas et al. [5] and Bay et al. [2], relies on local image features, where the aim is to first extract interesting points, edges or regions from the image, describe them numerically and match the descriptions to those of known objects, object classes or scenes.
The overall process is shown in the Figure 2.1.

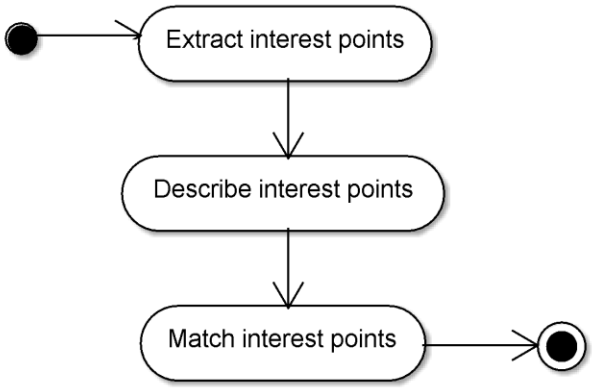

Figure 2.1 Overall object recognition process based on local image features. The input for the process is an image and the usage of matched points is application specific.

The process begins by extracting interest points from a given greyscale image, then those point are described numerically and finally the produced descriptors are matched

against descriptors extracted beforehand from another image or multiple images. How the matching point pairs are utilized, depends on the application. They can be used, for example, to determine the image that best matches the given image [6; 7], robot localization [8] or automatically stitch panorama images [9].
The proposed method in chapter 3 is based on Speeded Up Robust Features (SURF) presented Bay et al. in [2], thus SURF will be described in the following sections in more detail than other alternatives. The following sections deal with interest point extraction, description and matching separately.

2.2 Interest Point Extraction

The purpose of this phase is to produce interest points from images that are repeatable, accurately localizable and in sufficient quantity. Here the terms interest point and keypoint both refer to a point and its local neighbourhood, since a single point in an image, a pixel, has very little use for object detection in practice [10].
Repeatability means that the same points should be found under different viewing conditions. Unless the interest points are repeatable, they are useless even if they would fulfil the other two conditions, which makes this the most important property to evaluate [2]. Interest points also need to be accurately localizable in an image to make them semantically usable in later steps of the detection process. Even though quality goes before quantity for extracted interest points, the number of points should be sufficiently large to enable extracting them also from small objects and to provide enough data for subsequent steps of the process. Too high amount of points, on the other hand, can pose high computational requirements for processing all of them. [1; 10]
In real world images, the objects are seen differently in different images because the position of the camera or the illumination changes. In an optimal case, the used method should be invariant, or at least robust, to these changes to make it usable in real world conditions. Different viewing conditions include image translation, scaling, rotation, illumination changes and affine or perspective projection [1]. In the following expressions $\mathbf{P}$ presents the original image matrix and $\mathbf{P}'$ the resulting image matrix after transformation or projection. $\mathbf{P}$ (for picture) is used instead of $\mathbf{I}$ (for image) in order to avoid confusion with the identity matrix. The matrices are presented in augmented form to simplify notation for some of the operations. This is only a mathematical presentation and the bottom rows have no significance in the real world.
Translation means moving each point in image by constant amount in a given direction. Translation is a non-linear transformation and in matrix form the translation for images can be expressed as

$$\boldsymbol{P}' = \boldsymbol{T}_v\boldsymbol{P} = \begin{bmatrix} 1 & 0 & v_x \\ 0 & 1 & v_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ 1 \end{bmatrix} = \boldsymbol{P} + \boldsymbol{v} \tag{1}$$

where $\mathbf{T}_v$ is the translation matrix, P_x and P_y the x and y components of a point P(x,y) from the matrix **P**, and **v** the fixed vector containing the x and y values for the amount of translation.

Scaling means enlarging or shrinking the image by constant factor. This linear transformation can be presented in matrix form as

$$\boldsymbol{P'} = \boldsymbol{S_v P} = \begin{bmatrix} v_x & 0 & 0 \\ 0 & v_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ 1 \end{bmatrix} \tag{2}$$

where S_v is the scaling matrix and v contains the x and y factors for scaling in the corresponding directions.

Rotation is a linear transformation and means rotating the image by constant angle about the origin. In matrix format rotation of image clockwise about the origin can be expressed as

$$\boldsymbol{P'} = \boldsymbol{R_\theta P} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ 1 \end{bmatrix} \tag{3}$$

where R_θ is the transformation matrix and θ is the rotation angle.

Affine projection is a transformation that preserves straight lines and relative distances. It is equivalent to translation followed by linear transformation. These linear transformations can be, but are not limited to, scaling or rotation. Affine projection for images in matrix form is

$$\boldsymbol{P'} = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ 1 \end{bmatrix} = \boldsymbol{AP} \tag{4}$$

where **A** is the affine transformation matrix.

Perspective projection, as the analogy from viewing perspective indicates, points to image plane along lines that emanate from a single point. Thus it is a mapping from three dimensions into a two dimensional plane. [11; 12]

The interest point extraction process for SURF uses a method called Fast-Hessian detector to detect interest points from a given image. The interest points are here called keypoints. This process is shown in the figure 2.2.

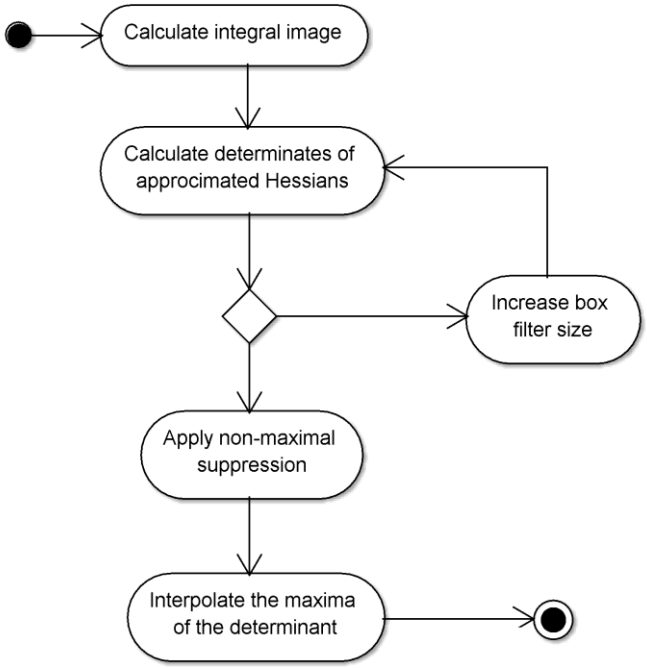

Figure 2.2 Fast-Hessian detector used in SURF. The detector receives an input image and outputs the keypoint locations.

The first step is to calculate the integral image for the given image. The integral image at location **x**=(x, y) contains the sum of the pixels above and to the left of **x** [13]. The integral image at location **x** can be presented as

$$\boldsymbol{I_{\Sigma}}(\boldsymbol{x}) = \sum_{i=0}^{i<x} \sum_{j=0}^{j<y} \boldsymbol{I}(i,j) \qquad (5)$$

where $\boldsymbol{I_{\Sigma}}(\boldsymbol{x})$ is the integral image at location **x** and **I**(i,j) is the input image pixel intensity value at location (i,j). This is shown in the figure 2.3:

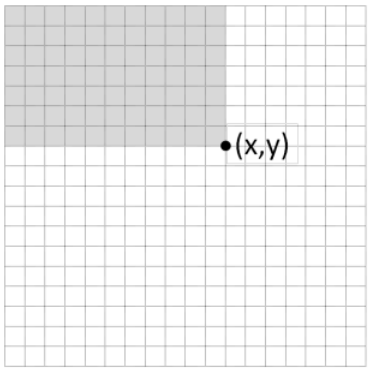

Figure 2.3. Integral image at location (x, y) is the sum of all pixels above and to the left of it.

Note that the calculation of integral image is performed on a single colour channel of an image, not for all channels in case of a colour image. Usually a greyscale conversion is performed before calculating the integral image.
The benefit of the integral image representation is that the sum of pixel intensities of any rectangular area can be calculated by four additions, as shown in the figure Figure 2.4.

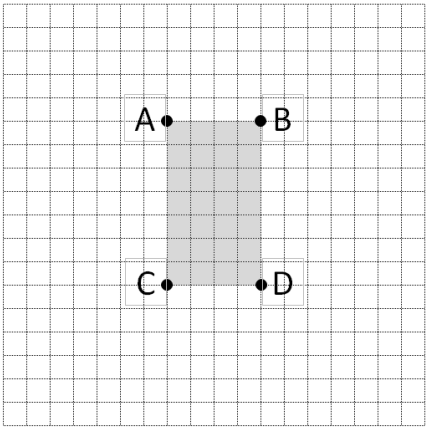

Figure 2.4 Example of integral image usage for calculating the sum of pixel intensities of a rectangular image area.

Here the sum of the rectangular area ABCD is calculated using the integral image with the formula

$$\sum \boldsymbol{ABDC} = \boldsymbol{I_{\Sigma}}(\boldsymbol{D}) + \boldsymbol{I_{\Sigma}}(\boldsymbol{A}) - \boldsymbol{I_{\Sigma}}(\boldsymbol{B}) - \boldsymbol{I_{\Sigma}}(\boldsymbol{C}) \quad (6)$$

where $\sum \boldsymbol{ABDC}$ is the sum all pixel intensities within the rectangle $\boldsymbol{ABCD}$. This enables efficient calculation of the filter values in the next stage and the integral image itself can be formed in one pass of the original image. [2; 13]
Once the integral image has been formed, Hessian matrices of second order Gaussian derivatives are approximated at each point and on different scales. The Hessian matrix describes the local curvature of a function of many variables and here it is used to detect potential keypoint locations. In the Fast-Hessian detector the function to be described is the second order derivative of the Gaussian function

$$g(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\left(\frac{x^2+y^2}{2\sigma^2}\right)} \quad (7)$$

where x and y are the coordinates of the image, that is the distance from the origin in the horizontal and vertical axis, and σ^2 is the variance [14]. Thus the Hessian matrix to be approximated is

$$H(\boldsymbol{x},\sigma) = \begin{bmatrix} \frac{\partial^2}{\partial x^2} g(\boldsymbol{x},\sigma) & \frac{\partial^2}{\partial xy} g(\boldsymbol{x},\sigma) \\ \frac{\partial^2}{\partial xy} g(\boldsymbol{x},\sigma) & \frac{\partial^2}{\partial y^2} g(\boldsymbol{x},\sigma) \end{bmatrix} \quad (8)$$

where H(**x**,σ) is the Hessian matrix at location **x**=(x, y) and at scale σ. The approximations used for the Gaussian second order derivatives are done using box filters shown in the figure 2.5. The approximations for $\frac{\partial^2}{\partial x^2} g(\boldsymbol{x},\sigma)$, $\frac{\partial^2}{\partial y^2} g(\boldsymbol{x},\sigma)$ and $\frac{\partial^2}{\partial xy} g(\boldsymbol{x},\sigma)$ are denoted by D_{xx}, D_{yy} and D_{xy}.

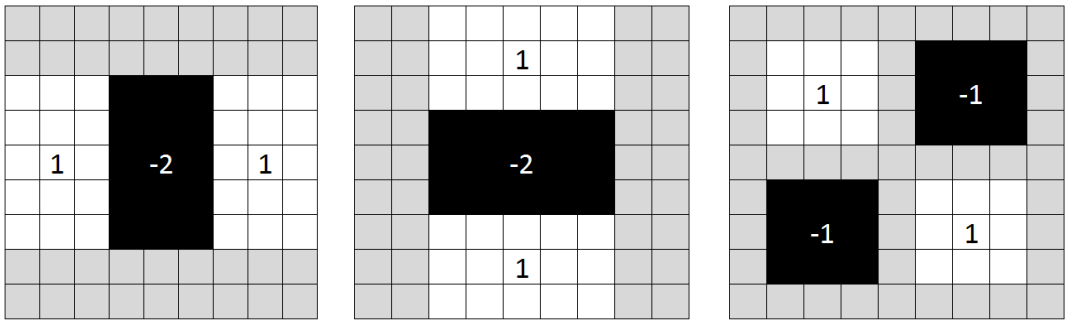

Figure 2.5. 9x9 box filters D_{xx}, D_{yy} and D_{xy} approximating Gaussian second order derivatives.

These box filters can be efficiently calculated using the integral image constructed in the previous step. The determinants of the Hessian matrix are used to determine and localize the keypoints. The determinant of the approximated Hessian matrix is calculated as

$$det(H_{approx}) = D_{xx}D_{yy} - (0.9D_{xy})^2 \quad (9)$$

where H_{approx} is the approximation of the Hessian matrix Here the value 0.9 is used to balance the weights and is calculated based on filter size and scale. Once this determinant has been calculated at each point on the initial scale, where the sizes the box filters are 9x9, the filter sizes are increased and the determinants are calculated again. The used filter sizes are grouped into octaves, each containing four filter sizes, and the filter size increase is dependent on the octave. For the first octave the filter size increase is 6 and is doubled for the next octaves. The table 2.1 demonstrates the filter sizes.

Table 2.1. Box filter sizes used in calculating the determinates of the approximated Hessian matrix in different scales in case of 4 scales per octave

Octave	***Scale 1***	***Scale 2***	***Scale 3***	***Scale 4***
1	9x9	15x15	21x21	27x27
2	15x15	27x27	39x39	51x51
3	27x27	51x51	75x75	99x99

...	...	...	...	...

Once the determinant values have been calculated, the local maxima of the determinant is searched by applying a non-maximum suppression, where a local maximum has greater value than all its neighbours [15]. This produces the locations of the keypoints. The suppression is applied to each point´s immediate neighbourhood and also to neighbouring scales, that is in a 3x3x3 neighbourhood of the points. [2]
After the non-maximum suppression, 3-D quadratic interpolation is performed to provide keypoint locations with sub-pixel and sub-scale accuracy [16]. The method used is the one proposed by Brown et al. in [17]. In this method, a 3D quadratic is fitted to the approximation of the Gaussian. This is shown in the equation (10).

$$\boldsymbol{D}(\boldsymbol{x}) = \boldsymbol{D} + \frac{\boldsymbol{\partial D^T}}{\boldsymbol{\partial x^2}} + \frac{\mathbf{1}}{\mathbf{2}}\boldsymbol{x^T}\frac{\boldsymbol{\partial^2 D}}{\boldsymbol{\partial x^2}}\boldsymbol{x} \tag{10}$$

Here $D(\boldsymbol{x}) = D(x, y, \sigma)$ is the approximation of the Gaussian and $\mathbf{x}=(x,y,\sigma)^T$ is the scale-space coordinate. The location of the keypoint, $\hat{\boldsymbol{x}}$, is taken as the extremum of equation (x), which is calculated using the derivative

$$\hat{\boldsymbol{x}} = -\frac{\boldsymbol{\partial^2 D}}{\boldsymbol{\partial x^2}}^{-\mathbf{1}}\frac{\boldsymbol{\partial D}}{\boldsymbol{\partial x}} \tag{11}$$

and setting the derivative to zero. [12; 17]

2.3 Interest Point Description

The detected keypoints need to be described in a way that produces features which can be used in the matching phase. The features should be distinctive, local and efficient to calculate. Distinctiveness means that the intensity patterns of the keypoints should contain a lot of variation so that features can be distinguished from one another and matched. Local features can be used even in case of partial occlusion and allow modelling of geometric and photometric deformations between two images, which is important in the method proposed in this thesis. The distinctiveness requirement however poses restrictions on the locality of the features, since they need to contain enough variation in order not to lose their distinctiveness. Efficient calculation is especially important when using the method in a mobile application. [3]
First the orientation of the keypoint is assigned. This is done because the same keypoints might be rotated differently in different images because either the camera or a real world object has been rotated between images. This process is shown in the figure 2.6.

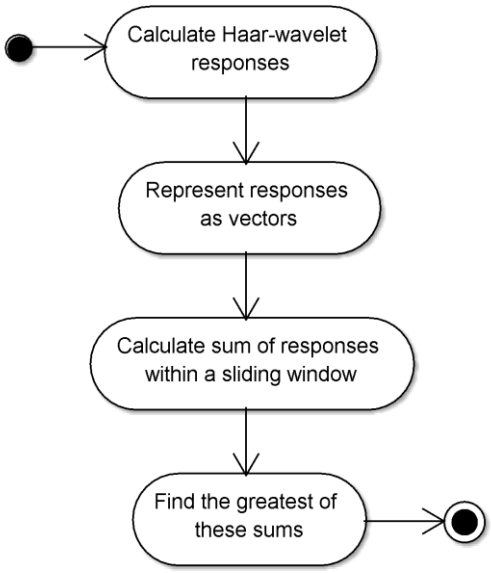

Figure 2.6. Keypoint orientation assignment process in SURF. For each keypoint, an angle representing keypoint orientation is assigned.

The orientation is assigned separately for each keypoint produced by the extraction phase. The Haar-wavelet responses are calculated in x and y directions. The size of the wavelet, sampling step and the neighbourhood of the keypoint to be searched are all dependent on the scale the keypoint was found at. The wavelets used are shown in the figure 2.7.

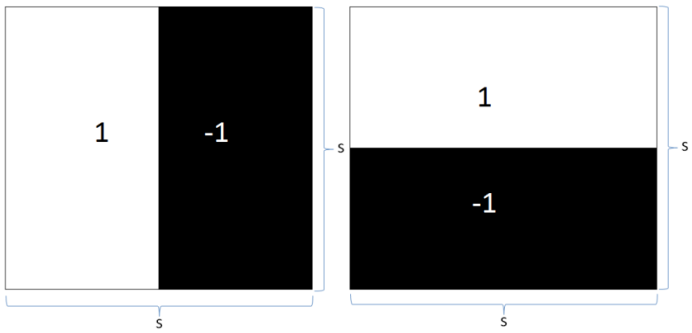

Figure 2.7. Haar wavelets used in orientation assignment. The response is a weighted sum of pixels within an area.

The wavelet sizes as well as the sampling steps are s and the neighbourhood to be evaluated is a circular region of size 6s around the keypoint. Here s is the scale where the keypoint was extracted from. Once all the responses in the keypoint neighbourhood are calculated, they are weighted with a Gaussian centered at the keypoint using value 2.5s for σ. Next step is to use a sliding orientation window covering an angle of 60° at a time and sum up all the responses within the window. The window producing the largest sum of responses determines the orientation of the keypoint. There is also an upright

version of SURF called U-SURF, which skips the orientation calculation and always assumes on upright orientation for all the keypoints. This saves computational resources but is only suited for scenarios where camera rotates around vertical axis. U-SURF is not used in this thesis. [2]

After the orientation has been determined for keypoints, they are given numerical descriptions in form of SURF descriptors. The process of calculating the descriptor for a single keypoint is presented in figure 2.8.

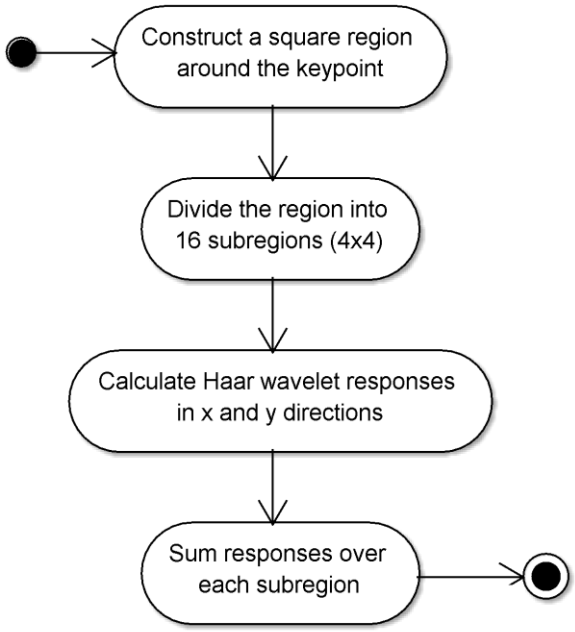

Figure 2.8. Calculation of a SURF descriptor for a single keypoint. The input is a keypoint having a location, scale an orientation determined in previous phases and the output is a numerical feature vector.

First a square region of size 20s is constructed around the keypoint rotated according to the orientation calculated in the previous step. This region is then divided into 4x4 subregions. For each of the 16 sub-regions, Haar wavelet responses are calculated in x and y directions and weighted with a Gaussian centered at the keypoint location using value 3.3 for σ. The responses are denoted dx and dy. Note that here x and y directions are relative to the keypoint orientation, not the original image orientation. The sums of the responses dx and dy and summed over the sub-region, along with the sums of the absolute responses |dx| and |dy|. This produces four numerical values for each sub-region ($\sum dx, \sum dy, \sum|dx|, \sum|dy|$). The values from each sub-region are combined to form a feature vector consisting of total of 16x4=64 numerical values. This vector is then scaled into a unit vector to produce the final SURF descriptor for a keypoint. [2; 6].

2.4 Matching Interest Points

The keypoint descriptions alone are of little use unless they can be used to match keypoints from different images or views. One approach is the nearest neighbour matching method. The idea is to find the best match from a database of reference features for each feature extracted from a given image while at the same time discard those features for which a proper match is not found.
Figure 2.9 shows an overview of the nearest neighbour matching process.

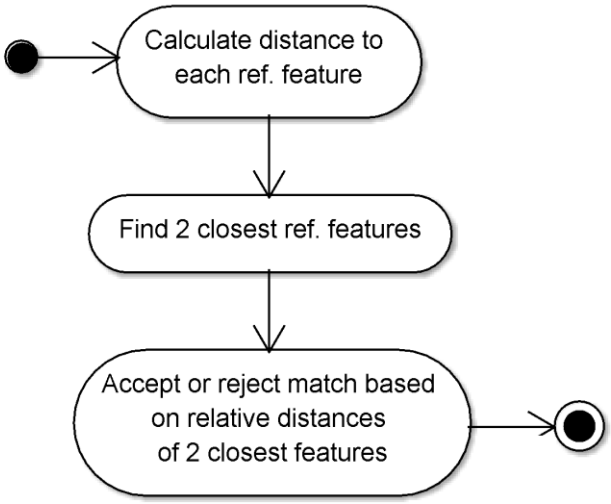

Figure 2.9. Nearest neighbour matching method. Keypoint is matched with the closest reference keypoint is the distance to the second closest big enough compared to the distance to the closest one.

The process begins by calculating the distance from the keypoint under consideration to all the keypoints in the reference database in the 64-dimensional feature space. Each keypoint represents a single point in the features space through its 64-dimensional SURF descriptor vector. In this thesis the Euclidean distance

$$d(\boldsymbol{p}, \boldsymbol{q}) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \quad (12)$$

is used as the distance metric, as done in [6; 12]. In the equation (12), p and q are the 64-dimensional SURF descriptors of the keypoint under consideration and one of the keypoints from the reference database, respectively. Other metrics have also been used with this method, like the Mahalanobis distance used by Baumberg in [4].
Once all the distances have been calculated, the two smallest ones are selected. The decision on whether to accept or reject the closest reference keypoint as a match is based on the relative distances of the two closest keypoints to the keypoint under consideration. This is written as

$$d_1 < \alpha d_2 \quad (13)$$

where d_1 is the distance to the closest reference keypoint, d_2 the distance to the second closest reference keypoint and α is a numerical constant. Value 0.8 is used for α in [6; 12]. However, a different value is used in this thesis. This is discussed in the next chapter in more detail. Once all the keypoints from the given image are processed, the matching phase results in a set of corresponding keypoints from another image or from multiple other images. How this information is then utilized, depends on the application [4; 6; 12].

3 PROPOSED METHOD

In this chapter, a customized mixed-reality object selection method is presented. The original intended purpose for the method was to enable a user to point at objects with a mobile phone, select one of the objects and control it using the phone. For this, suitable existing methods were examined, and after selecting one, it was modified to better fulfil the requirements for the specific scenarios.
Here first the use-case scenarios are discussed in order to present the requirements from object detection point of view and to explain the steps taken before selecting the used approach. After that, the proposed method is described in more detail by presenting the overall process and then focusing more on the details of the parts of the process that were modified from the common process. The use-cases are further described from more practical point of view in chapter 4.

3.1 Use-Case Scenarios and Background

The initial idea was to make a mobile phone recognize devices present in a home environment, like in a living room. The devices considered were mainly TVs, DVD-players, loudspeakers and such. After a successful recognition of devices, the devices were to be marked with an icon to enable the user to select and control them. This is shown in figure 3.1.

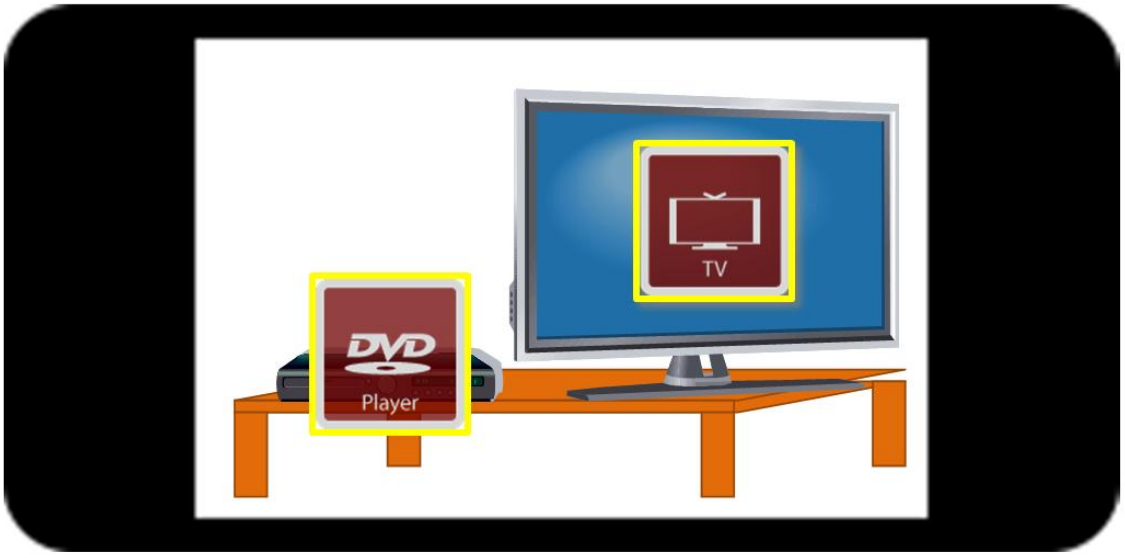

Figure 3.1. Example scenario where the detected devices have been marked with icons on top of them. These icons can be used to initiate interaction with the devices.

This example image shows a case where the application has detected two devices for the user and provides a means to interact with the devices. In a real scenario, a camera image often contains much information on the background too and the devices themselves are not so easily distinguished from the background.

When devices like these are viewed using a mobile phone´s camera from several metres away, in most cases they can be described as small dark-coloured rectangular objects. A TV and a DVD-player, for example, are still easily distinguished from one another. This is however not the case between two DVD-players having the same brand but a different model, or even the same model but only a different version number. This can be difficult even for a human being. Very similar looking devices naturally pose a challenge for a method that aims to distinguish them from one another.

Controlling the devices at home can be done in various ways depending on the device at hand and not all devices present at home can be controlled using one universal interface. The way of transferring the commands to devices can include infra red (IR) or wireless local area network (WLAN) among others. Even if commands to every device could be transferred in an identical way, each device or device model can react differently to a given control command and each device can have its own individual control interface. For example pressing an “arrow right” button on a remote control might cause the TV to increase volume, but the Blu-ray player might switch to the next scene of a movie. Though devices from one manufacturer may have partly identical interface, the capabilities of the devices can vary and thus the interfaces for controlling them will vary too. It was assumed, that knowing the device´s properties as well as possible, it is most probable that the device can also be controlled in a way the user wants to. Because of this, recognizing the devices to a unique device identification code level became the goal for the recognition.

The scenarios the method proposed in this chapter was designed for are described in more detail in the implementations chapter. In this chapter, only the requirements from the object detection and recognition points of view are discussed.

The specific conditions the approach was designed for resemble conditions in a normal living room at home. The objects are assumed to be rather permanently stationed. This limits the types of objects the approach is suited for. TV and such devices do not tend to move frequently, but remote controls, magazines or toys do. While the detectable objects themselves were assumed to be stationary, a fair amount of occlusion and other changes in the environment was assumed. This was the main reason why a method based on local image features was selected as the basis of the approach, and another reason was that the approach utilizes background information in addition to the information extractable from the devices themselves. This way a person or an object blocking direct view to the object to be detected or controlled does not necessarily cause considerable issues. Lightning was not given much consideration during the development. Typical mobile phone cameras do not work very well in dark conditions and thus the environments were assumed to be well lit to enable the use of the approach with off the shelf camera phones.

3.2 Object Detection Process

The proposed method is a modified combination of SURF feature detection and nearest neighbour matching approach. Both of these were described in detail in the previous chapter. The basic idea is to first extract features from a given image and then find matching features from reference image or images. These matching features provide corresponding points between images which can then be used to determine object locations based on the given image. For most parts, the proposed method follows the commonly used feature detection and matching process that is presented in chapter 2. The modifications proposed in this thesis aim to eliminate incorrect matches both faster and with smaller error rate than in the standard approach for the considered scenarios.
The proposed object detection process is shown in figure 3.2.

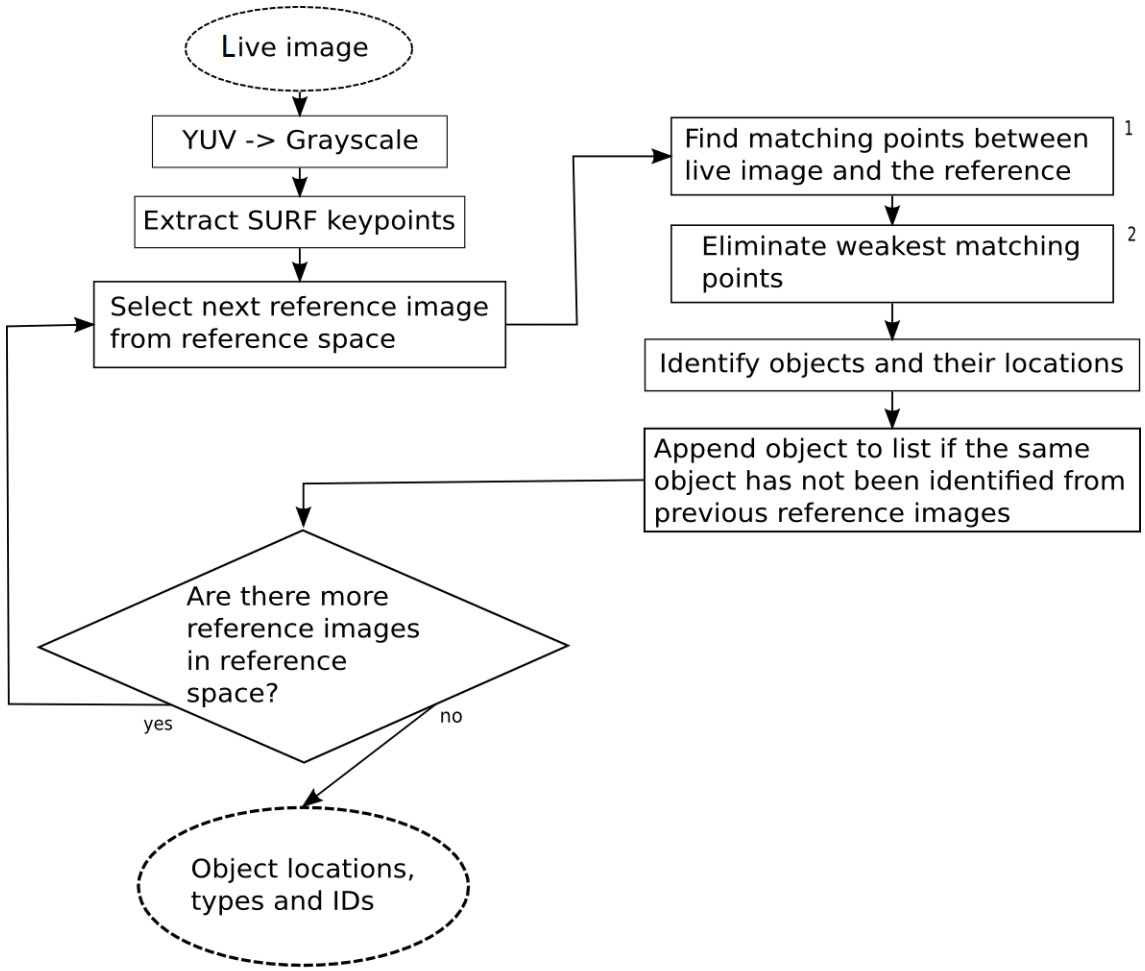

Figure 3.2. Proposed object detection process. The input for the process is captured image data and it results in object locations, types and identification codes. The steps marked with numbers 1 and 2 are presented in more detail in figures 3.4 and 3.6.

The process starts when a colour image is obtained from camera sensor. The image data is first transformed into grayscale in order to reduce the amount of information needed to process. No subsampling is used here. The actual transformation used depends on the

exact format of the colour data. Two formats are considered in this work: RGB and YUV. RGB data is converted into grayscale using the formula

$$Y = 0.299 * R + 0.587 * G + 0.114 * B \quad (14)$$

where Y is the intensity value of a pixel in the resulting grayscale image and R, G and B are the red, green and blue component values of the corresponding pixel in the colour image. The numeric values here are the ones that are used in the software implementation which was utilized in the thesis. The exact YUV format considered here is the YUYV [18], where two horizontally adjacent pixels of the colour image are represented by one macropixel, as shown in the figure 3.3.

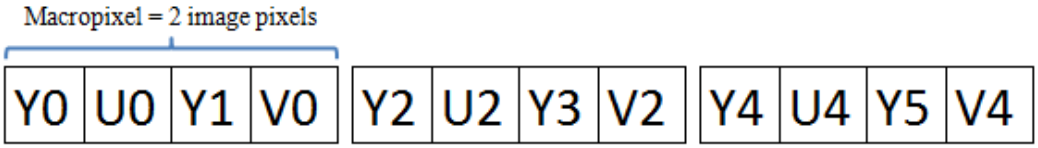

Figure 3.3. YUVY image format [18]. Each macropixel represents two horizontally adjacent image pixels. Y is the luminance component and U, V are the chrominance components.

The conversion from this colour format to grayscale is done by using each luminance value Y from the colour image as the intensity values of the grayscale image. The chrominance components U and V are discarded.

After the colour transformation, SURF extraction is performed on the grayscale data. This extraction process is described in the previous chapter and it is not modified in any way for the proposed method. The approach here uses the normal 64 digit long SURF descriptors.

The SURF keypoints from the live image are then matched against reference keypoints obtained from a reference image with identical extraction process. Building a reference database and an efficient way for retrieving data from one was not part of the scope of this work. Instead, each reference image is processed one at a time in a loop, as shown in the figure 3.2. After the matching keypoints are found, the weakest matches are eliminated before proceeding further. The matching and elimination steps are described in section 3.3.

Once the matching phase is concluded, the locations of the remaining matching keypoints are used to determine the objects that exist in the given image and their locations. This is done by calculating a perspective transformation between the corresponding locations from the reference image and the given live image. For this purpose the transformation available in the OpenCV [23] was used. The points used in the transformation are limited only by the steps described in section 3.3. The software implementation would also enable the use of RANSAC [19], but this is not utilized here. This is because

the tested RANSAC implementation on OpenCV [23] resulted in poor transformations and decreased the speed of the matching.
Using the resulting transformation matrix the object locations from the reference image are transformed into coordinates of the given image. The object is considered to be detected from the given image, if the coordinates of it, obtained through the transformation, lie within the image boundaries. Before the object is accepted among the found ones, a check is made whether the same object has been found from previous images or not.
Due to the SURF limitations, which can be seen in the Results chapter, multiple reference images from one scene are often used to improve accuracy. This can lead to the object being detected using more than one reference and thus a decision must be made about which of these detections to use or should they be combined in some way. During this work, several methods were tested for this purpose. One way was to use the detection that placed the object closest to the centre of the given image. Another was to use the one that was produced by the smallest transformation; that is the one that was found from the reference that was considered to be geometrically most similar with the given image. Several methods that tried to calculate a goodness value for each detection based on various properties of the corresponding keypoints were also tested. These properties included the absolute and relative differences in keypoint strengths and average distances of keypoints in the feature space. These methods were compared against selecting a random detection and none of them proved to be usable in all situations. As the added computational costs did not offer considerable improvements, they were abandoned and a more simple solution was adopted. If the same object has been detected from a previous reference the new detection is discarded and the existing one kept.
After the current reference image has been processed, a check is made whether there are more reference images available that should still be processed. If there are any, the same steps are performed for each reference image. Once there are no more reference images to process, we have obtained a list objects detected from the given live image and their locations in the coordinates of the given live image.

3.3 Modified Nearest Neighbour Matching Method

The overall object detection process in this work is a commonly used one. The actual modifications are related to the matching phase of the process. The reason for this was originally that this was the area that lacked usable implementations. During the creation of such implementations for this work, several issues were faced that required additional algorithmic development. As the result of this development, the matching phase was modified to be more suitable for the given scenarios.
The matching phase is divided into two separate tasks, which are marked by numbers one and two in the figure 3.2. The first task, marked with number one in the figure, is to find matching points between two images. The second one, marked with number two in

the figure, is to refine the matching by eliminating some of the most probable incorrect matches.

The first part of the matching phase is shown in more detail in the figure 3.4.

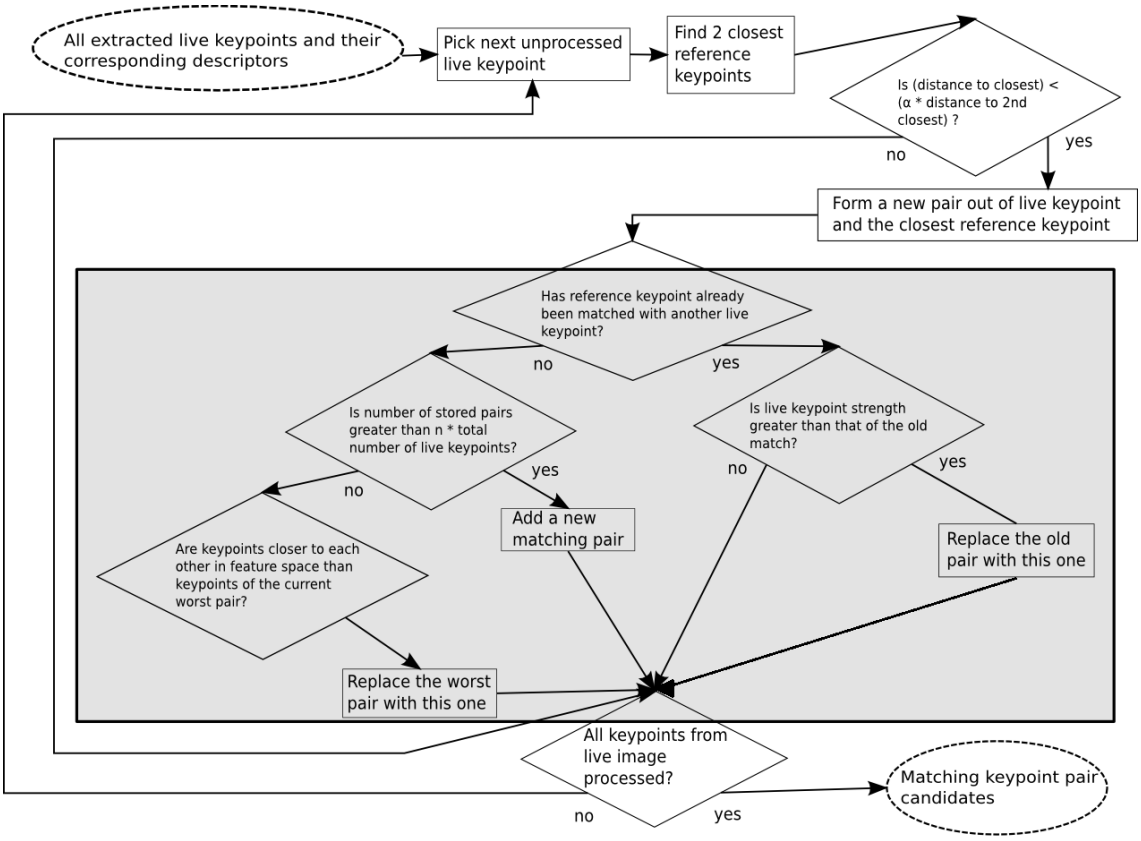

Figure 3.4. Proposed keypoint matching process. The input is the list of keypoints and descriptors from a given live image and the output is a list matching keypoint pair candidates. The grey box shows the part that is added to the nearest neighbour matching process [4].

The matching process starts after the keypoints have been extracted from the given live image. The extracted information includes the keypoints and their corresponding descriptors. More detailed explanation of these can be found from chapter 2. Each of these keypoints are processed one at the time by comparing them against the keypoints extracted from a reference image. The first step is to find the two closest reference keypoints in the feature space of the keypoint descriptors. As the descriptors used in this work were the 64 digit long descriptors, the two closest keypoints are the reference keypoints for which the descriptors are closest to the currently processed live keypoint descriptor in the 64-dimensional feature space. Euclidean distance between the descriptors in the feature space is used as the metric for determining the closest ones.

Once the two closest keypoints have been found, the distance from the currently processed live keypoint to the closest reference keypoint and the distance from the currently processed live keypoint to the second closest reference keypoint are examined. The idea here is to check if the closest reference keypoint is the correct match. At this stage it is

considered to be one, if the second best candidate is far enough compared to the best candidate. Equation (13) is used as the condition for determining if this is the case. If the condition is true, then a new keypoint pair is formed out of the currently processed live keypoint and the closest reference keypoint. The values used for α in this thesis are 0.2 and 0.25. These values are more strict than the value 0.7 used by Bay et. al. in [2] or 0.8 used by Fasel and Van Gool in [6] as the matched keypoint pairs have a higher probability of being discarded. This is demonstrated in figure Figure 3.5

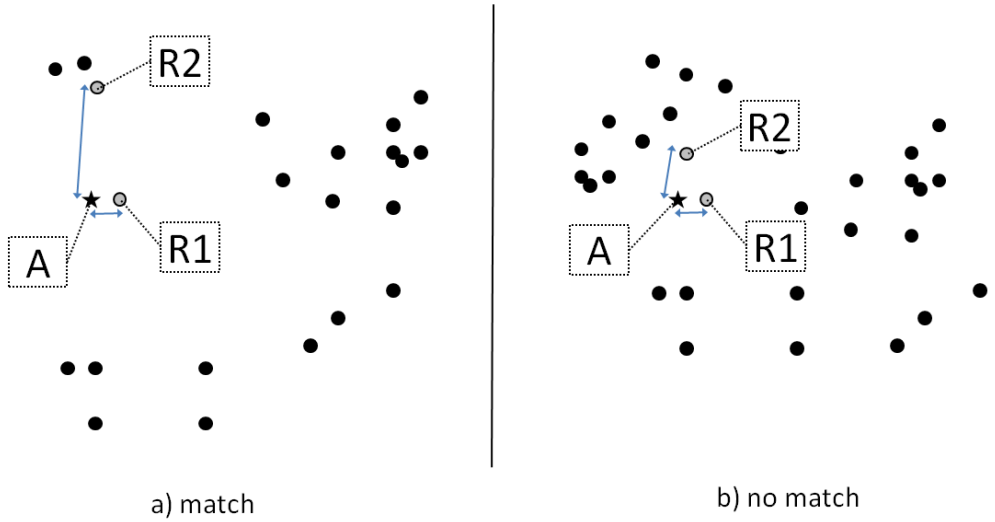

Figure 3.5. The nearest neighbour ratio matching criteria. The closest reference point (R1) to the given keypoint (A) in feature space is considered as a match, if it is close enough compared to the second closest reference keypoint (R2).

Figure 3.5 shows two cases for nearest neighbour ratio matching criteria. In the figure a 2-dimensional feature space is shown. The actual feature space is 64-dimensional. In Figure 3.5 a) the distance from the given keypoint (A) to the closest reference keypoint (R1) is 0.1 and the distance to the second closest (R2) 1.0. Thus the condition from equation (13) is fulfilled using any value greater than 0.1 for α. R1 is considered a match to A since it´s clearly the only close reference keypoint to A. In Figure 3.5 b) the distance between A and R1 is again 0.1 but the distance from A to R2 is now 0.2. Now any value for α which is greater than 0.5 would fulfil the equation (13). The values 0.2 and 0.25 used in this thesis would not produce a match in this case, since there is no reference keypoint that would be clearly the correct match for A.

The commonly used nearest neighbour ratio matching method does not process the newly formed pair any further but instead considers that a match was found [6]. The modifications to the common method are additional checks that are performed on the newly formed keypoint pair in order to determine if the match was actually a correct one.

The first addition is to check if the reference keypoint of the new pair has already been matched with another keypoint from the given live image. The assumption here is that each point in the given live image has at most one matching point in any one reference

image. More than one match can be correct from the algorithm point of view, for example if the image contains two identical objects, like speakers. Even if the local areas of the image would be identical, or almost identical, the possibly resulting in multiple matches for one live image keypoint are not desired. This is demonstrated in the figure 3.6, where both matches are locally correct from algorithm point of view, but the match with the left speaker from the reference image is not desirable.

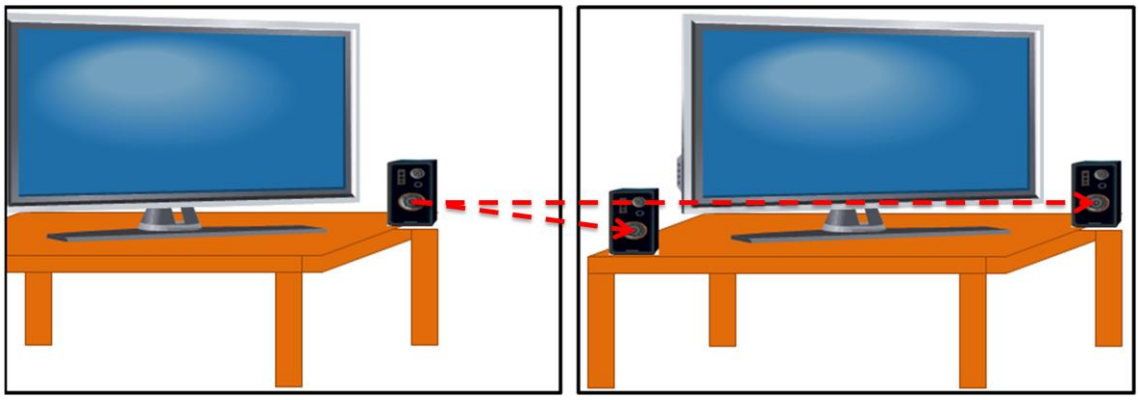

Figure 3.6. Multiple matches for one point from a given live image. The left image represents a live image and the right image a reference image. The match with the left speaker is semantically incorrect.

In most cases this issue should be taken care by different scales of the extracted SURF features, but this is not always the case in practice. Thus the check is made to prevent multiple matches, which would be disadvantageous for the geometrical transform later on. If the reference keypoint of the new keypoint pair has been matched already with another keypoint from the live image, then only one of the pairs is kept. The pair for which the live keypoint strength is greater is kept and the other pair is discarded.

If the currently processed live keypoint has not been matched previously with any reference keypoint, the match might still be an incorrect one. In practice it was observed that all matches that fulfilled equation (13) were rarely the correct ones, so additional conditions were examined. The properties of the incorrectly matched pairs were compared against those of the correctly matched pairs in order to create a new condition to eliminate all the incorrect matches. Such a universal condition was not found in this work, but there were few properties that did distinguish correct matches from incorrect ones in most cases. The most promising out of these was the Euclidean distance of the live keypoint descriptor and the reference keypoint descriptor in the feature space. Since this distance was already calculated for the first step of the matching phase, it did not require much additional computational effort either. No fixed absolute value for this distance was found that alone could be used as a threshold to distinguish correct matches from incorrect ones, so selecting a certain amount of the best ones was the approach that was used.

Here it is first checked if there are already enough matched keypoint pairs against the currently processed reference image. The amount is defined as percentage of the total

amount of keypoints extracted from the given live image. If the maximum amount has not yet been reached, then the current pair is added to the list of matched keypoint pairs. Otherwise another check is made if the current pair has smaller Euclidean distance of the live keypoint descriptor and the reference keypoint descriptor in the feature space than the pair that is currently ranked worst out of the accepted matching pairs. If the distance is smaller than the distance for the worst pair, the worst pair is replaced by the current pair.
Replacing the worst pairs in a loop can be considered as eliminating keypoint pairs using a dynamically set threshold which is dependent on the given image data through the extracted keypoints and their descriptors.
If there are more live keypoints that have not been processed yet, the next one is selected and the same matching process is performed. This is repeated until all the keypoints extracted from the live image have been processed. Once there are no more keypoints to process, the first part of the matching phase is concluded, resulting into a list of matching keypoint pair candidates.
The second part of the matching phase is shown in the figure 3.7.

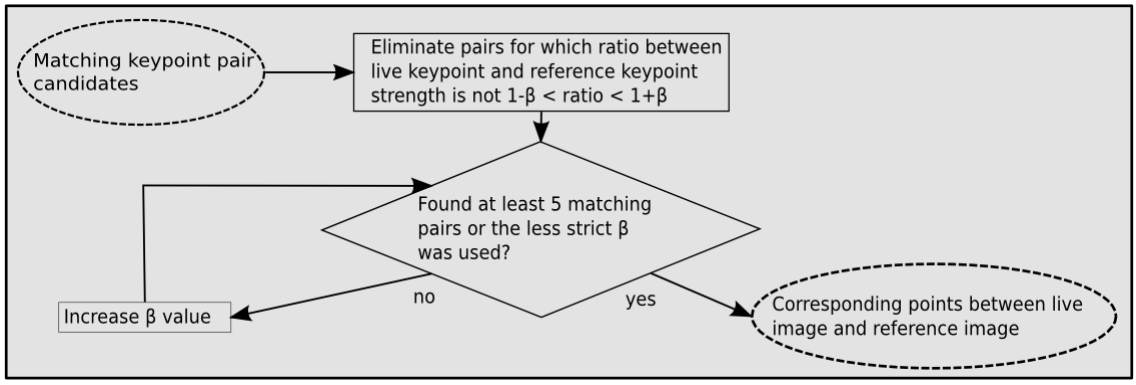

Figure 3.7. Proposed keypoint elimination process. The input is the list of matching keypoint pair candidates from the first part of matching phase and the result is the final list of matching keypoint pairs. The grey box shows the part that is added to the common nearest neighbour matching process [4].

This part of the matching phase is performed after the first part has produced the matching keypoint pair candidates. The last steps of the first part of the matching phase discarded a portion of the matched pairs by eliminating the weakest matches. While the first part eliminated pairs by using a threshold value that was based on the given image data, the second part of the matching phase eliminates pairs based on a predefined threshold value. The threshold in the last steps of the first part was for Euclidian distance between the live keypoint descriptor and the reference keypoint descriptor in the feature space; here a threshold is used for the ratio between live keypoint strength and reference keypoint strength.
The assumption that led to this condition was that two keypoints for which the keypoint strengths deviate from one another by a certain amount are unlikely matches. This as-

sumption was tested by comparing correctly matched pairs to incorrectly matched ones after the first part of the matching phase and the strength difference was in average smaller for correct matches than for incorrect matches. The absolute values for keypoint strengths depended on the images in question so the absolute difference did not offer a good solution. Instead the relative difference of the keypoint strengths was used to discard most probable incorrect matches.

Two different values for the maximum allowable deviation as a percentage are used in the second part of the matching process. First the more strict β value, that is the smaller percentage, is used to eliminate pairs that do not fulfil the condition

$$1-\beta < R < 1+\beta \tag{15}$$

where β is the maximum allowable deviation as fraction and R is the ratio of the keypoint strengths. The ratio is calculated using the formula

$$R = \frac{S_{live}}{S_{reference}} \tag{16}$$

where R is the ratio, S_{live} is the strength of the live keypoint and $S_{reference}$ is the strength of the reference keypoint.

If less than 5 keypoint pairs fulfil the condition (15), the less strict value β is used and the elimination is performed again on all of the keypoint pair candidates. If again less than five keypoint candidate pairs fulfil the condition (15), the whole detection process is aborted prematurely and no objects are considered to be found. If at least 5 keypoint candidates remain after the elimination process, the detection process is continued as described in the section 3.2. The more strict and the less strict values for β used in this thesis are 0.1 and 0.5 respectively.

4 IMPLEMENTATIONS

In this chapter two different implementations are presented, which make use of the object detection method described in the previous chapter. While both implementations have several common goals and design principles, both also have their own distinct objectives. This chapter only deals with the implementations from design and implementation points of view. Testing results are presented in the chapter Results.
First the objectives and use-cases for both implementations are presented including the common ones. Here the implementations are described from a software design point of view, more detailed object detection perspectives are described in the previous chapter. Next each implementation is described in more detail including programming tools, languages and physical devices used and any issues faced during the design and implementation phases.

4.1 Objectives and Use-Case Scenarios

The two implementations are both called the One-Eyed Wizard (OEW). The name originates from the analogy of the mobile phone having a one eye, the camera, and practicing magic by enabling the user to control the environment through the phone. To distinguish between the two, we use the names ‘client-server OEW’ and ‘standalone mobile OEW’ here. From the user´s point of view both implementations aim to provide the user means to detect, select and control devices that exist in home-like environments.
The client-server OEW has a kind of client-server architecture and it consists of two applications. There is a mobile application running on the mobile phone and a server application running on an external computer. The mobile application offers the user interface (UI) and some light-weight functionality. Computationally more expensive functions are performed on the server application, which is also responsible for storing image related data. The original purpose of the client-server OEW was to act as a technical demo for the object detection method described in the previous chapter and to demonstrate some of the possibilities where this method could be used.
The standalone mobile OEW is a standalone mobile application. All functions described in this section are performed on the mobile phone and all the data is stored on the mobile phone. This application was created in order to investigate if some of the limitations of the client-server OEW could be avoided with a different approach.
Both versions share some common design principles. The user was to be able to select and control objects in a real world using a mobile phone utilizing computer vision. The user was supposed to be able to control the devices that he or she sees or, more generally, interact with the surrounding environment. From the technical point of view, the

applications were to be developed using open source software tools and components freely available to anyone. Both versions were to be modular so that any component could be replaced with a new one without a need to change the other parts of the software. The applications were to use readily available building blocks as much as possible.
The client-server OEW was designed to enable the use-cases shown in the figure 4.1.

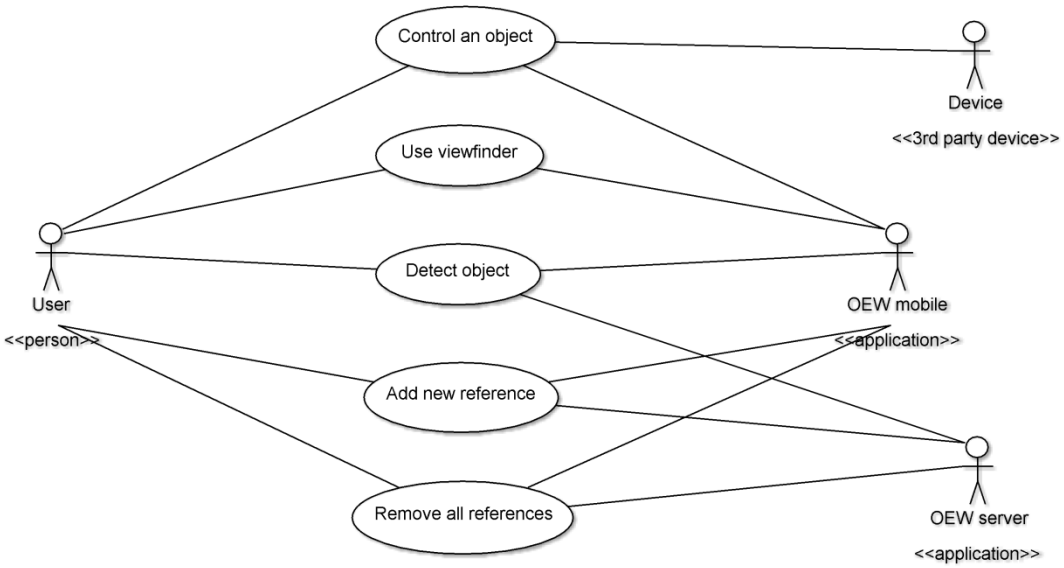

Figure 4.1. Use-cases for the client-server OEW. The user can view the scene through the viewfinder, detect and control objects or add and remove reference scenes. The user and the mobile application are involved in all the cases.

The user can view the scene through the camera viewfinder, which is displayed as a real time video feed on the mobile phone´s display. When the user is pointing at the desired location, the objects can be detected by clicking on the button at the centre of the screen. The UI for the viewfinder and detection is shown on the figure 4.2.

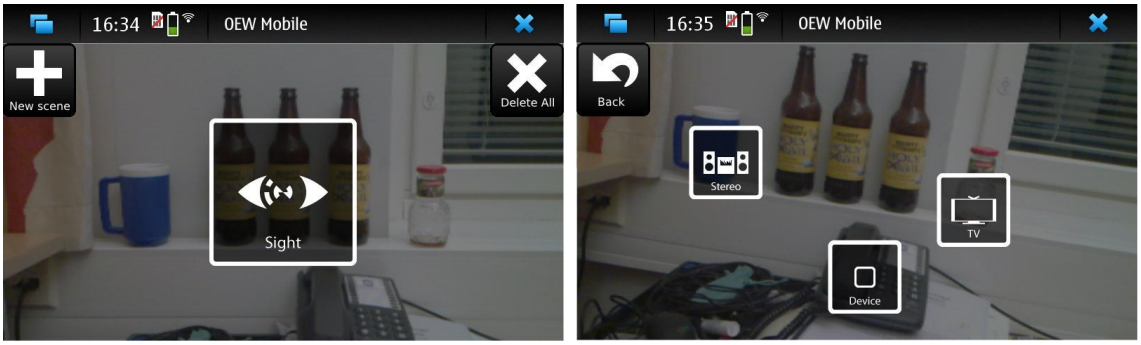

Figure 4.2. UI example from the client-server OEW. The left side shows the viewfinder with the detection button at the centre. The right side shows the detection results with icons on top of the detected objects.

Clicking on an icon on the detection results screen will show the available control options. These options can be directly related to a single device, if the icon represents a real world device. An icon can represent a more abstract object also. Clicking on an icon placed on top of a loud speaker for example could show options for controlling the volume of a single device or a group of devices. The actual control interfaces were not part of the scope of this work. For demoing purposes two interfaces were created. The first allowed the user to control a media centre. The user was able to view a list of video files on an external server and play them on the screen connected to the external server. On the second interface the user was able to browse audio files on the mobile phone and play them either from the mobile phone´s speakers or stream the audio to play from speakers of an external device.

The detection requires reference scenes to be stored in advance. A reference scene composes an image and the object locations information. Both versions of the OEW offer a functionality to add new reference scenes and to remove all existing reference scenes. For client-server OEW, these functions can be used with the buttons on the viewfinder UI, as shown in the figure 4.2. Here the button with plus sign on the upper left corner of the screen is used to create new reference scenes and the button on the upper right corner is used to delete all existing reference scenes.

The standalone mobile OEW includes most of the use-cases of the client-server OEW, as shown in the figure 4.3.

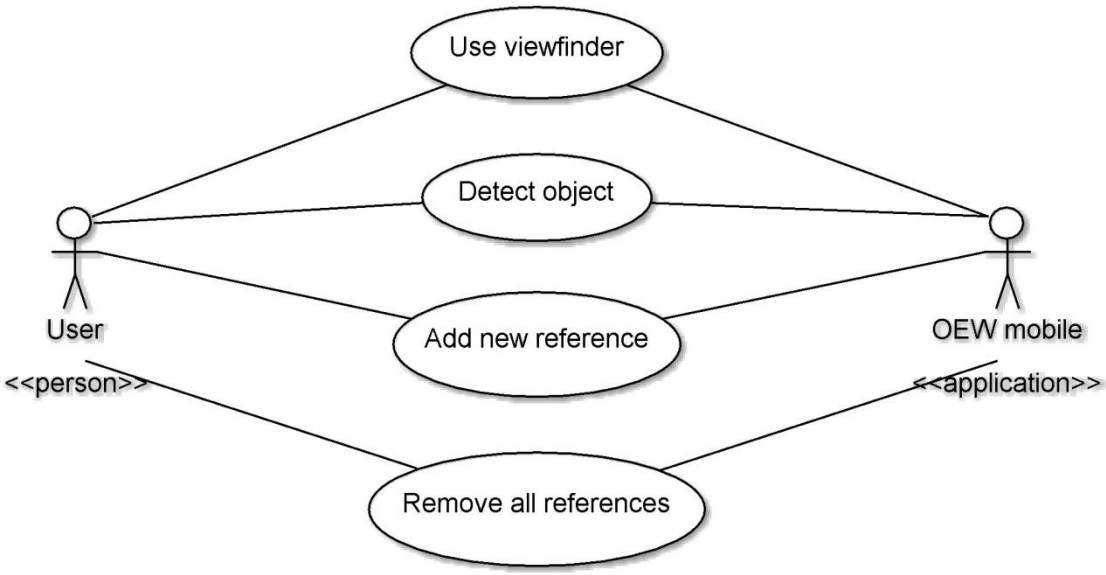

Figure 4.3. Use-cases for the standalone mobile OEW. The user can view the scene through viewfinder, detect objects or add and remove reference scenes. The standalone mobile OEW does not involve an OEW server application placed on an external computer.

One difference is that no control interfaces were implemented on the standalone mobile OEW. The devices can be selected in a same way as in the client-server OEW, but no control interface screen appears. This is because the standalone mobile OEW was fo-

cused on improving the user experience related to the object selection process compared to the client-server OEW. Another difference to notice is that the client-server OEW uses an OEW server application for most of the functions. This server application is run on an external computer. All of the functions of the standalone mobile OEW are performed on the mobile phone itself.

4.2 Client-Server Implementation

The client-server OEW uses the mobile application for UI and communications with the external devices to be controlled. The computationally heavy detection process is performed on the server application running at an external server to speed up the detection.
Both the mobile and the server application are implemented using Python. Python is an interpretable object-oriented programming language. It was selected because it was easy to learn, is available for all major operating systems, has a wide variety of libraries available and was free to use because of its open source license. It also provided a faster way to develop small demo applications compared to the alternative, which was C++. [20]
A mobile application required a suitable application framework for mobile application development. Qt was selected for this purpose because it was open source, cross platform, had good documentation and an active developer community. Since Qt did not have an official application programming interface (API) in Python, PyQt was used to counter this issue. PyQt is a set of Python bindings for the Qt framework. [21; 22]
The object selection using computer vision is a complex process involving various algorithms. It was not practical to implement all the required algorithms from scratch, so a suitable computer vision library was needed. For this purpose the OpenCV was selected. It is an open source computer vision programming library for real time computer vision. OpenCV has a wide range of computer vision related functions available, is constantly being developed and also has a large active developer community. [23]
As OpenCV's Python API did not offer all the necessary functions, which were available on the OpenCV's C++ API though, another API was needed for Python. PyOpenCv was selected for this purpose. It had the necessary functions that the official OpenCV Python API lacked and was, like OpenCV itself, a freely available open source project. [24]
For communication between the mobile application and the server application, a remote procedure call (RPC) enabling library RPyC was chosen. It was simple to use and offered all the necessary functionalities needed, like calling remote procedures and transferring data between the systems. [25]
The application required one more component for demoing the device controlling. VLC multimedia framework was used to communicate with and to control an external media server. It is free, supports many platforms and codecs and also has a Python API that satisfied the needs of the client-server OEW for demo purposes. [26]
The overall structure of the application is shown in figure 4.4.

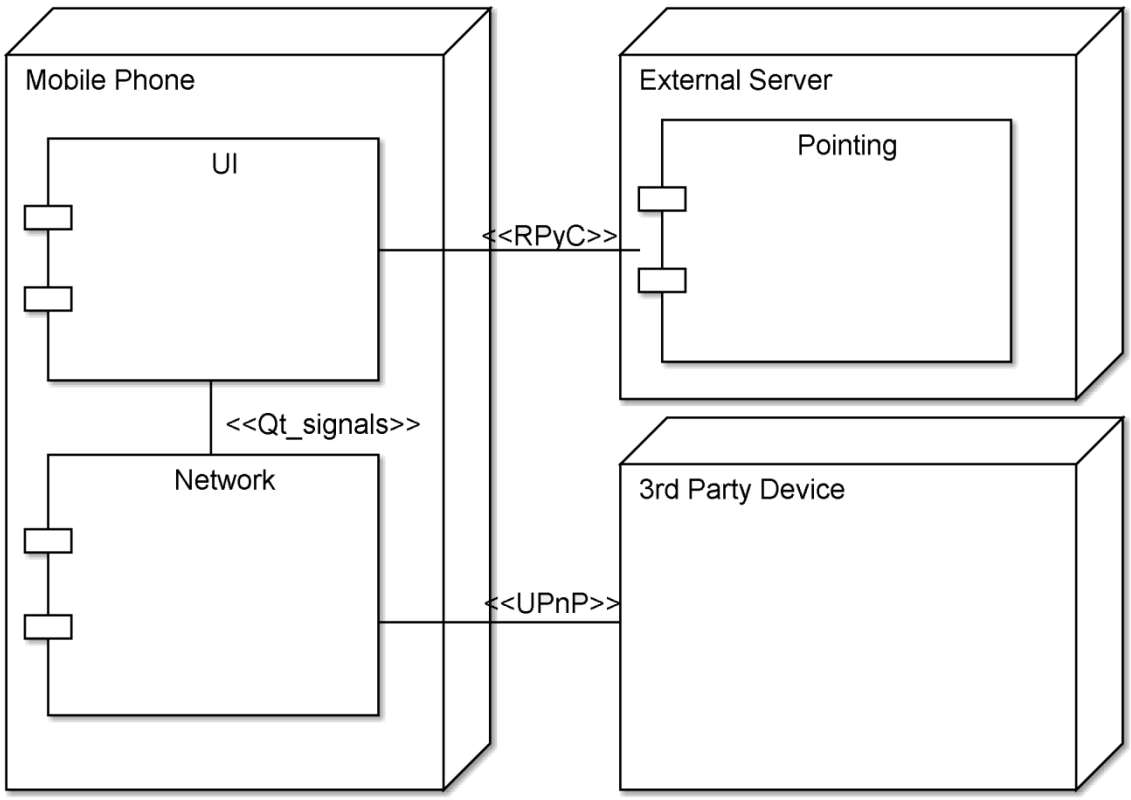

Figure 4.4. UML deployment diagram for the client-server OEW. Mobile phone communicates with the external server using remote procedure calls and with 3rd party devices using UPnP.

The application consists of three main parts: the pointing module, the UI module and the network module. The pointing module is in charge of transforming images into semantically usable information. While this implementation only utilizes computer vision for the pointing, the application could use other means too with a different pointing module implementation. The UI module handles the user interaction and the network module provides connections 3rd party devices and the pointing module.
Figure 4.5 shows the structure of the pointing part of the application.

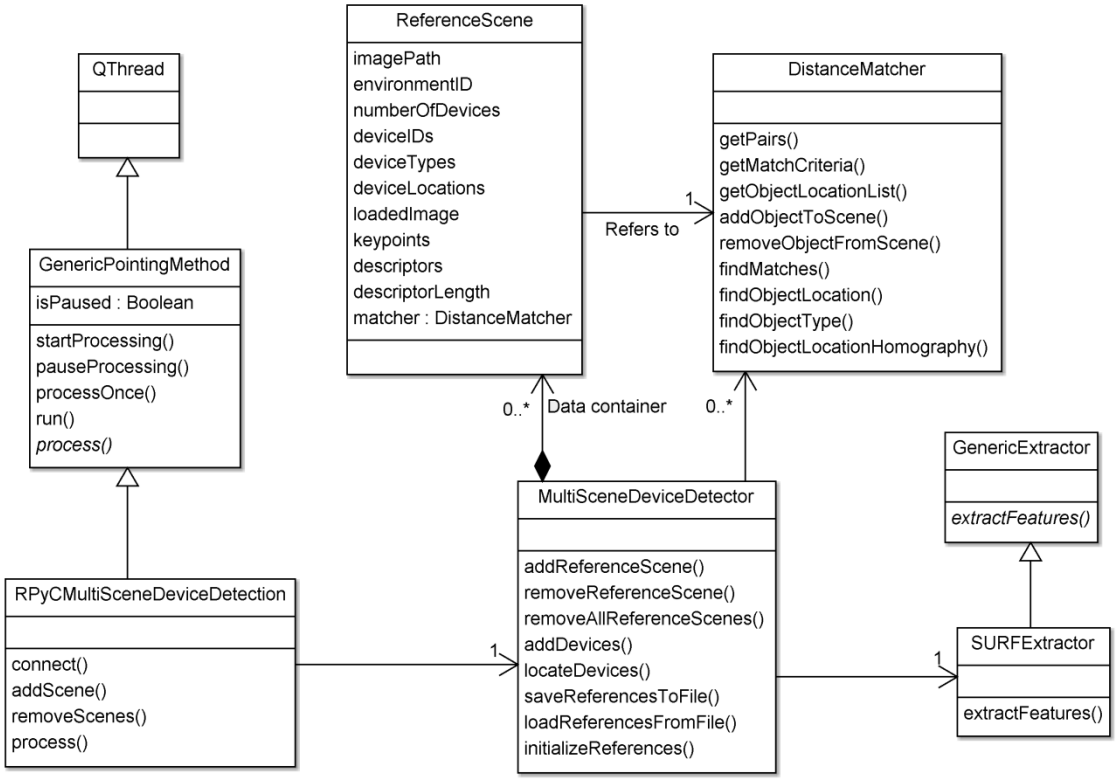

Figure 4.5. UML class diagram for the pointing part of client-server OEW. RPyCMultiSceneDeviceDetection uses the MultiSceneDeviceDetector remotely, which performs the actual detection by using the SURFExtractor and DistanceMatcher.

Here the MultiSceneDeviceDetector class is the main component of the pointing module. Together with the class RPyCMultiSceneDeviceDetection it provides an interface to program components outside the module. The data about the reference scenes is stored in instances of the class ReferenceScene. This data is persisted in a text file when the application is closed and reloaded once the application is started. SURF features are extracted from images using the class SURFExtractor and feature matching is performed using the class DistanceMatcher.
The test results for device detection using the above components are presented in the chapter Results.

4.3 Standalone Mobile Implementation

The standalone mobile OEW was designed to work on the mobile phone without the need for an external server application. For this implementation, Python was abandoned as the implementation language and C++ was used instead. The reasons for this change were potential improvement in processing speed, better support for the necessary API´s and the possibility to use QML as the language for implementing the UI´s.
QML provided an easy way to create UI´s having good usability and customizability. It also offered means to decouple the UI from the actual application logic, which was desirable from software design point of view. It is a JavaScript and CSS like declarative

UI language and is integrated into the Qt framework using Qt´s Meta-Object System. [27]
The overall structure differs from that of the client-server OEW (see figure 4.4). The structure of the standalone mobile OEW is shown in the figure 4.6.

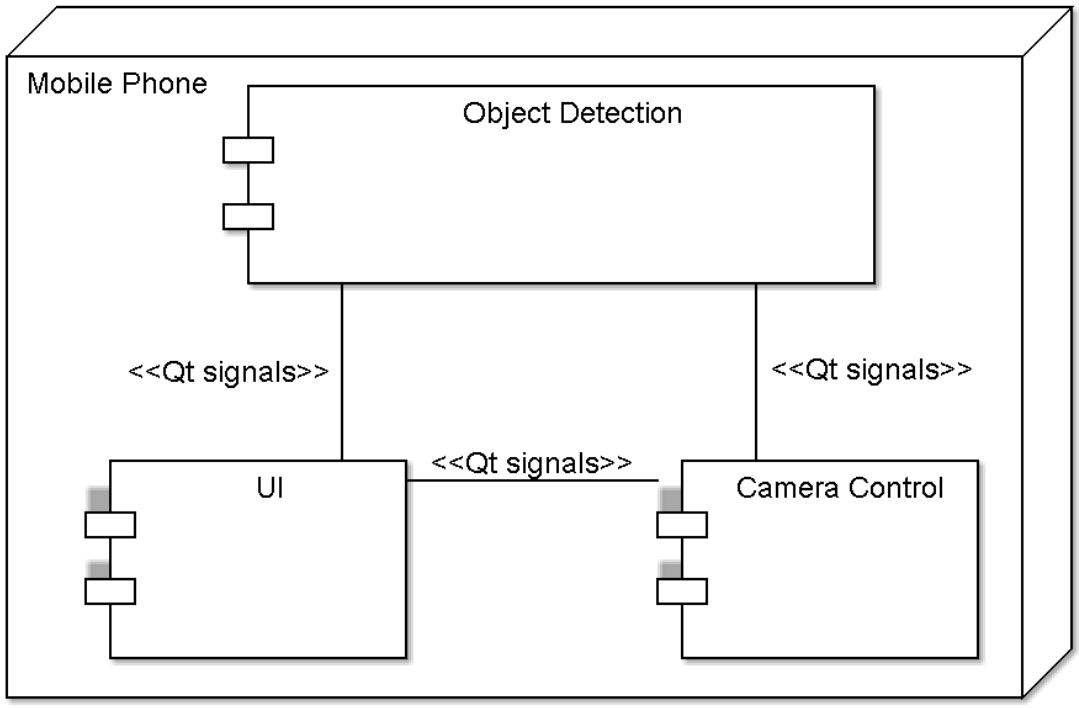

Figure 4.6. Deployment diagram for the standalone mobile OEW. The application consists of 3 components: user interface, camera control logic and the object detection logic. All three parts are located in a mobile phone.

The above structure is simplified compared to the structure of the client-server OEW. No external devices are required anymore and all communication between components is performed using Qt signals. The three components are loosely coupled through the use of signals instead of direct method calls. The user interface component is implemented using QML. It provides the UI and notifies other parts of the application about user actions using signals. It also receives certain signals in order to react to changes in the application state. The other two components are implemented using C++ and use signals to communicate with other parts of the program. Within the components themselves, normal method calls are used for communication between the classes.
The class structure of the standalone mobile OEW is presented in the figure 4.7.

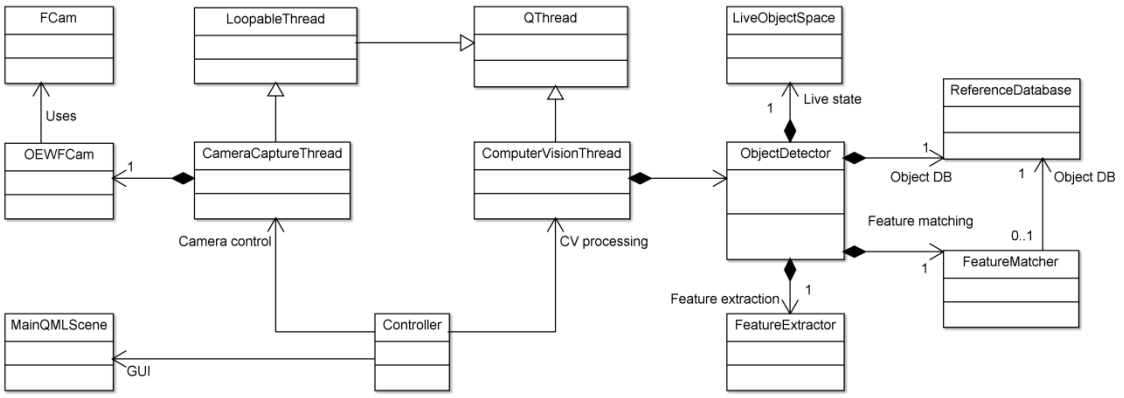

Figure 4.7. UML class diagram for the standalone mobile OEW. The controller class initializes the camera control, computer vision and user interface parts of the application.

The controller class acts as the main program of the application. It is in charge of creating and initializing the three parts of the application, that is the camera control, computer vision and the UI. The controller itself has no role after the application is successfully initialized, since the application components do not require the controller class to communicate with each other. The components are loosely coupled and communicate through events. This mechanism in Qt is called the signals and slots mechanism. Signals correspond to events that can be triggered explicitly and slots and methods of classes that the signals can be bound to e.g. event handlers. The controller thus initializes the classes and makes the necessary signal/slot bindings, after which the program flow is controlled by signals.

The camera control and the computer vision based processing parts are places in their separate threads to enable them to operate concurrently. This makes it possible to show the camera feed on the UI without interruptions from the computationally heavy computer vision processing.

The object detection part is similar to that of the client-server OEW in most aspects. The biggest difference is due to aim for real-time detection from the camera feed. This was not possible with the client-server OEW since the image acquisition, transfer to external computer, actual detection process, getting the results back and showing them took more than a second, sometimes several. For a real-time detection, this time needed to be reduced by at least an order of magnitude. For this reason, a slightly different approach was used.

Instead of detection object from every frame in an identical way, frames are processed differently based on the results of the previous frames. The process is described in figure 4.8.

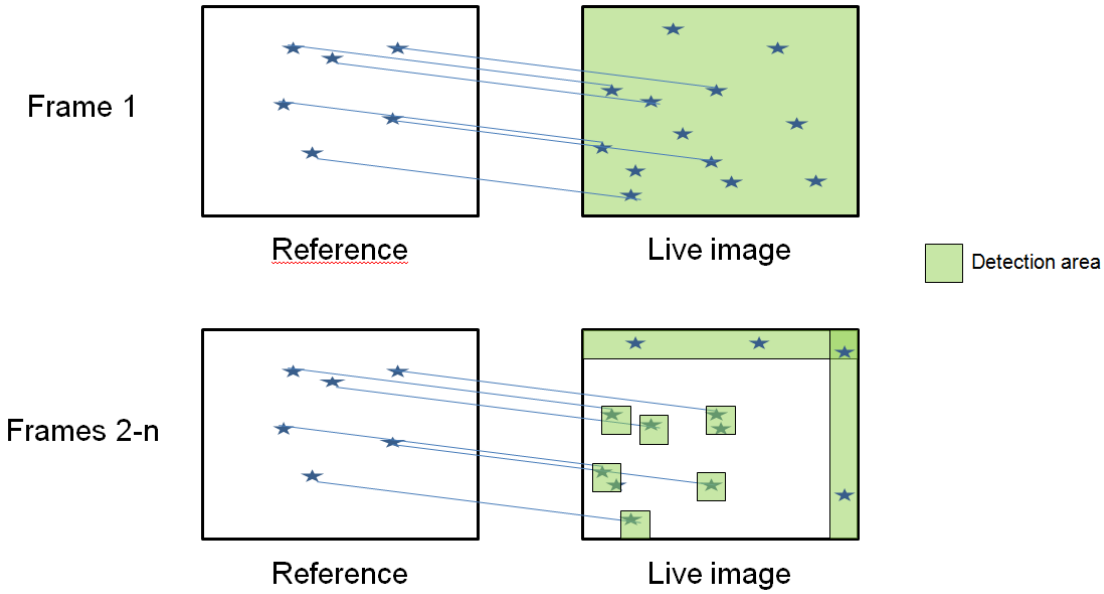

Figure 4.8. Detection approach of the standalone mobile OEW. For most frames, the detection is only performed for areas near previous matches and the areas that were not present in the previous frame.

For the first frame and also for any frame after an unsuccessful detection, the whole frame is used for the detection. After a successful detection, only parts of the successive frames are used for detection. The areas used are the areas surrounding the keypoints that were successfully matched in the previous frame and the areas that were not present in the previous frame.

For determining the parts that were not present in the previous frame, a rough estimate for the motion is calculated for each frame. This estimate is based on the location of a single object found from two frames and thus requires that the same object is found from two frames, frame *n* and frame *n+1*. If such an object exists, the estimate for the motion is the difference of the calculated object location in x and y directions. The estimate does not take into account any image transformations except for the simple translation.

Based on the estimate, certain areas next to the edges of the frame are included in the detection. If the estimate cannot be calculated, then only the surrounding areas around 10 keypoint locations from frame *n* are used as the areas where keypoints are searched in frame *n+1*.

5 RESULTS

This chapter describes the results of tests for measuring the accuracy of the proposed method. The results were obtained using the object detection component of the client-server OEW implementation. The goal was to assess the accuracy of the detection algorithm implementation in different environments and viewing conditions.
First the testing and evaluation arrangements are presented. This includes the measurement arrangements and process. Then the actual results are presented and analyzed. The discussion section includes an analysis of the results presented and also discussion on the achievements and shortcomings of the work from method and software design points of views.

5.1 Test Arrangements

The implementation was evaluated using image data gathered from four different living room-like environments. Each of the environments contained a TV and varying other devices and furniture. Example images from the used environments are shown in the figure 5.1.

Figure 5.1. Reference scenes used in the evaluation. The locations are numbered 1-4, from left to right, top to bottom.

In each environment four artificial object positions were determined. The positions were selected in such a way that the positions were accurately identifiable from each image and that the positions were as far away from each other as possible but still visible in all of the test images. In practice the positions were selected to be at the corners of real objects in the images, since these positions could be most accurately identified from different images. The positions were in no way correlated with the actual keypoint positions that the algorithm used.

The displacement between the ground truth and the object locations found by the implementation was used as the indicator of accuracy. Euclidian distance between the actual object position and the position produced by the implementation in pixels was used as the measure. Thus the smaller the distance, the better the accuracy.

The purpose was to evaluate the general accuracy of the implementation and the effect of environment and viewpoint changes on the accuracy. These viewpoint changes include viewing angle, camera rotation, panning angle and viewing distance.

Lightning was kept as constant as possible during the tests. This was because subjective tests did not show noticeable decrease in accuracy due to lightning changes as long as the lightning was comparable to normal indoor lightning or daylight. Dark environments were not in the scope of this work. Also it proved to be difficult to adjust the lightning of the tested environments to more than one level in a way that would have been mearurable or even repeatable.

The effect of environment was assessed by comparing the results from the above four environments to each other. The viewpoint changes are shown in figure 5.2.

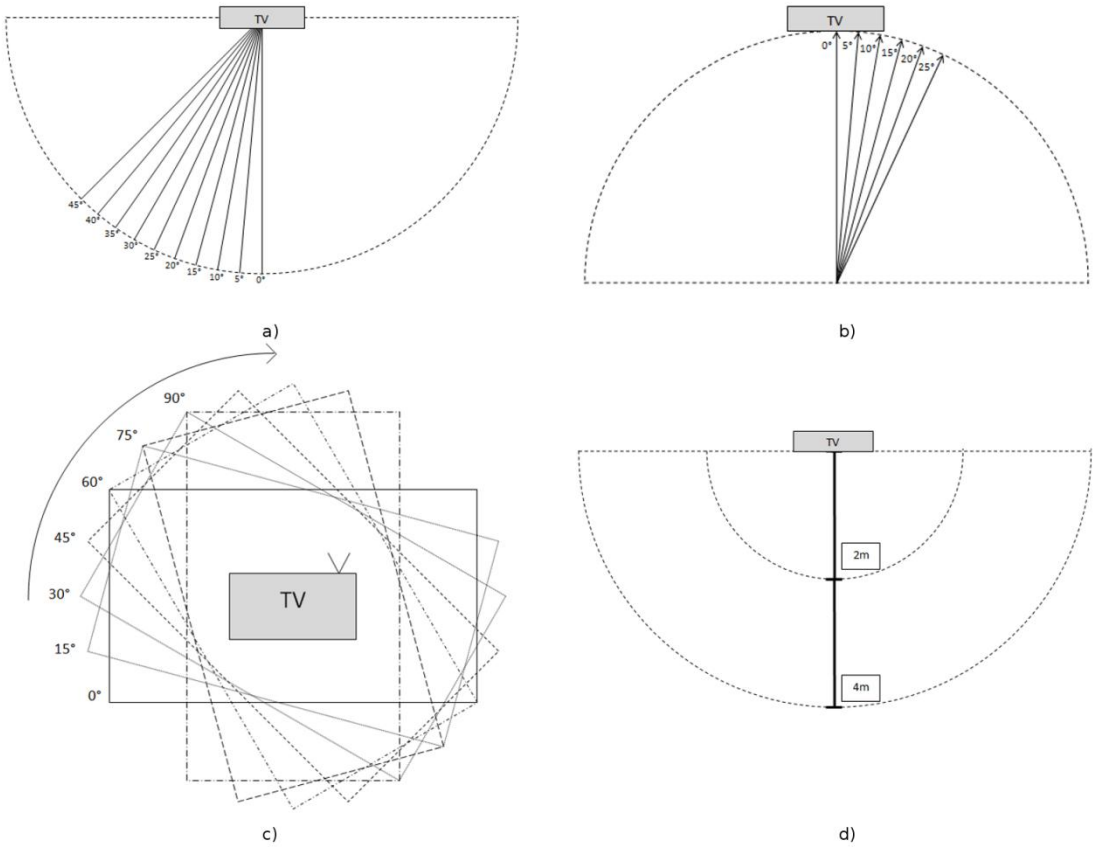

Figure 5.2. Viewpoint changes for accuracy evaluation. a) shows the setup for viewing angle changes, b) for panning angle, c) for camera rotation and d) for viewing distance changes.

Ten different viewing angles were used in the measurements ranging from zero to 45 degrees in steps of 5 degrees. The angle was measured from the line perpendicular to the TV screen. The viewing angle setup is shown in the figure 5.2 a).
Six values for camera panning angle were used between zero and 25 degrees in steps of 5 degrees. The angles were again measured from the line perpendicular to the TV screen. Bigger angles were not used since for those the camera view did not contain all of the defined object positions. The panning angle setup is shown in the figure 5.2 b).
Seven angles were used for camera rotation around the camera lens axis. The values used were between zero and 90 degrees in steps of 15 degrees. The zero degree position was set to be the normal horizontal camera position. The setup for rotation is shown figure 5.2 c).
Each of the above three viewpoint changes was tested from two different viewing distances measured from the TV screen. The distances used were two and four meters, as shown in the figure 5.2 d).
Each viewpoint change was applied one at a time. For each environment this resulted in 46 test images plus one additional image that was used as the reference image for the implementation. This reference image was taken from distance of three meters from the

TV with no rotation, viewing angle and panning angle. In addition to these images, six other images were taken from environment number two to evaluate the maximum distance where the implementation would still be usable. Thus the total number of test images was 190 plus 4 reference images.

Figure 5.3. Test images for viewing angle change. Images are taken from environment number two from distance of 4 meters.

Figure 5.4. Test images for panning angle change. Images are taken from environment number two from distance of 4 meters.

Figure 5.5. Test images for camera rotation change. Images are taken from environment number two from distance of 4 meters.

Figure 5.3 shows the ten different viewing angles used in testing for one distance from one environment. The angle values are shown in the figure 5.2 a). Figure 5.4 shows the

6 panning angles used in testing for one distance from one environment. The angle values are shown in the figure 5.2 b). Figure 5.5 shows the seven camera rotation angles for one distance from one location. The values for the angles are shown in the figure 5.2 c).

Each image was taken using the same mobile phone that was used to develop and run both of the implementations. The resolution of the images was 2560x1440 pixels. From the original size, the images were reduced to quarter size in both horizontal and vertical directions to produce images of size 640x360 pixels. This size was similar to the one the actual implementation uses (640x480).

For every image, each object position was determined manually to obtain the ground truth that was used in comparison with the results that the implementation produced. As the positions were selected to be in clearly identifiable locations, like corners, it was possible to determine the positions accurately by zooming the image into level of several individual pixels. The manual position identification process was repeated twice to minimize the effects of this manual process on the accuracy results. Thus the x and y coordinates each of the three objects in each of the 190 test images were identified to create the data to compare the implementation results to.

5.2 Test Results

Results from the test arrangements from the previous section are described by dealing with each type of viewing condition change from Figure 5.2 separately. The figures here present the average displacement between the ground truth object locations and the locations produced by the used algorithm.

The results in case of changes in the viewing angle, described in the Figure 5.2 a), are shown in the Figure 5.6.

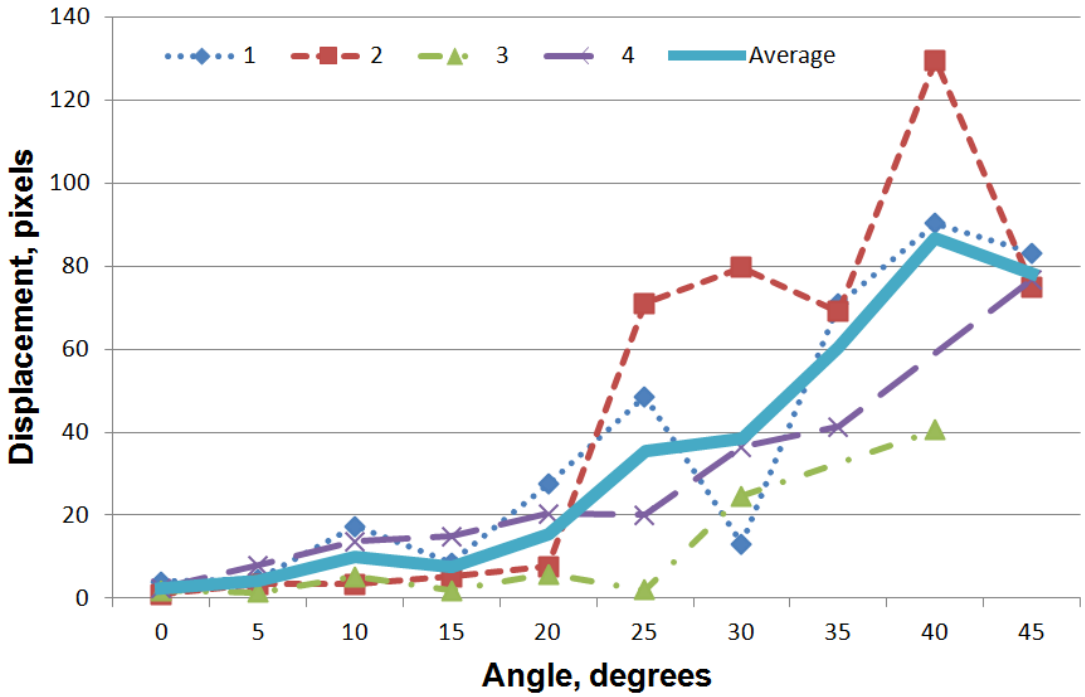

Figure 5.6. Average object displacement versus viewing angle.

Here each point represents the average displacement of eight objects: each image contained four objects and each viewing angle was measured from two distances for each location. It can be seen that the accuracy of the algorithm starts to decline rapidly after 20°-25°. When the camera is rotated less than twenty degrees, the average displacement remains below 10 pixels. With bigger angles however, the error starts to show to the user, which hinders the user experience.

As SURF is not affine invariant and changing the viewing angle by rotating the camera around the objects causes affine deformations to the image, the above results are expected. As a workaround to this SURF limitation, the used method provided the option to use more than one reference image. Figure 5.7 shows the results for changing viewpoint angle when using two reference images instead of one as in Figure 5.6.

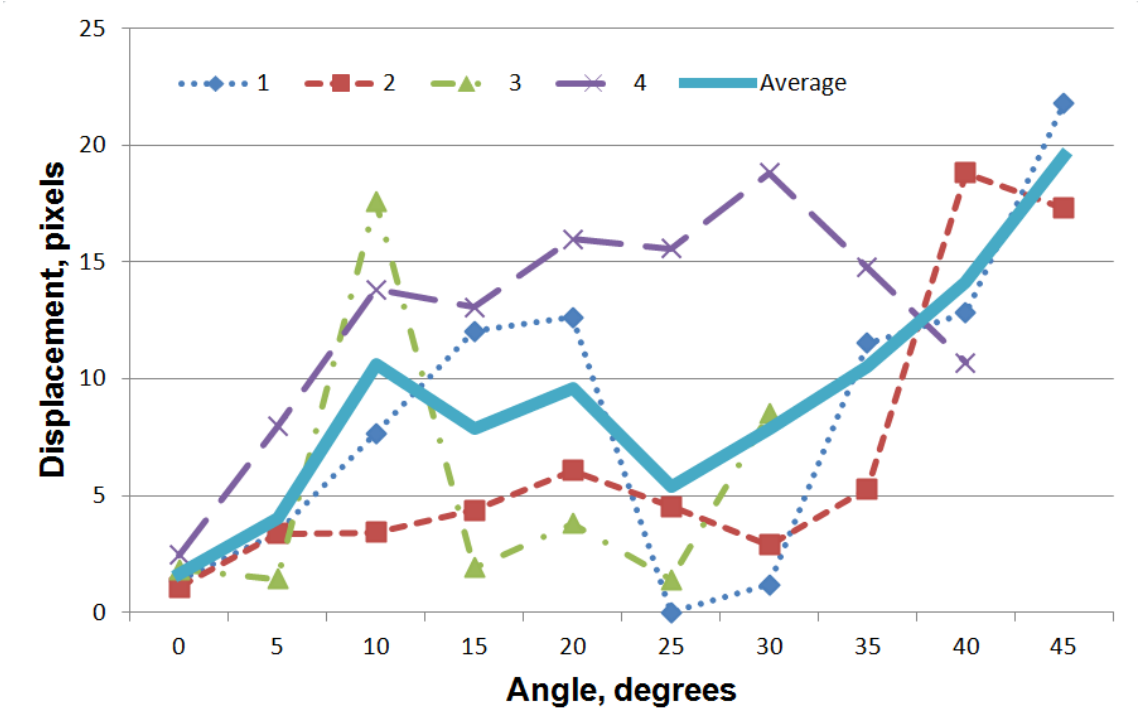

Figure 5.7. Average object displacement versus viewing angle when using two reference images.

Here the second reference image was one of the test images for each location, the one taken from distance of four meters and viewing angle of 25 degrees. It was up to the algorithm to decide which reference image to use in each case.

Compared to the results shown in Figure 5.6, the average displacement was decreased considerably, particularly for the largest angles. Note the change in the scale of the y-axis between the two figures. The improved accuracy comes with a cost however. The computational effort is doubled when the reference database is built from separate images and there is not efficient means for comparing except to compare to each reference image one at a time. This is the case with the proposed method in the thesis. Also in practice each additional reference image requires additional effort from the user since the reference locations for objects need to be set for each reference.

The method is more robust to camera panning angle described in the Figure 5.2 b). The results for panning angle are shown in the Figure 5.8.

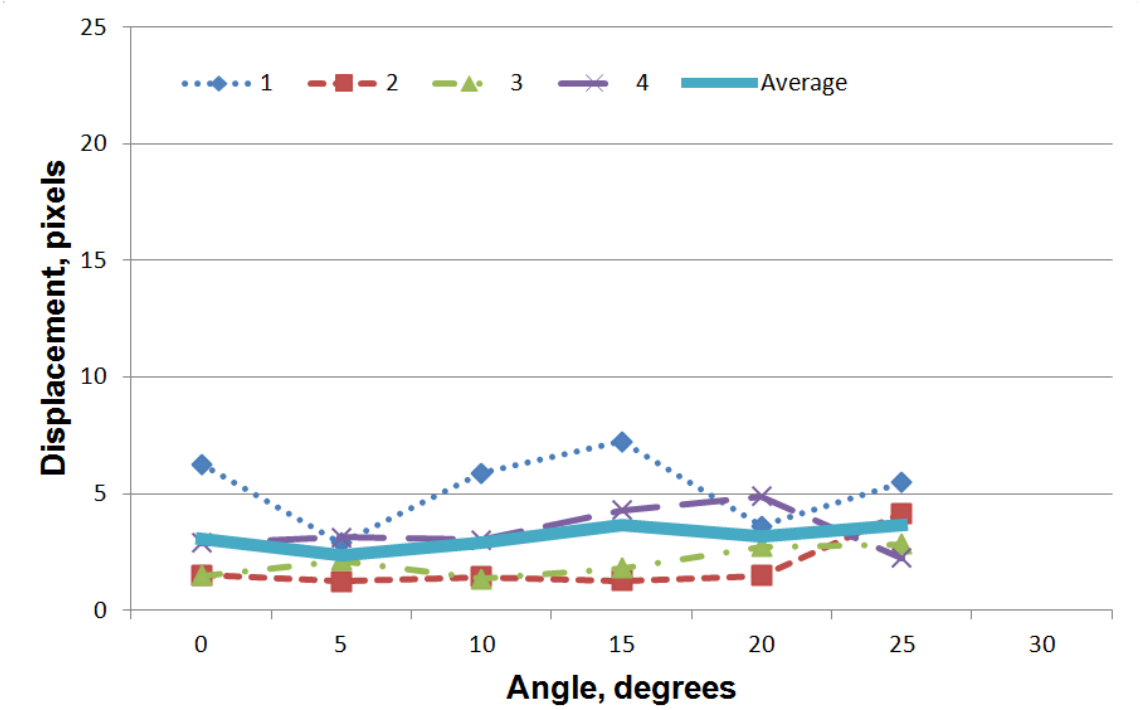

Figure 5.8. Average object displacement versus panning angle.

The average displacement in this case remains below five pixels. The differences between different locations are more significant than differences between panning angles. The effects of camera rotation, as shown in the Figure 5.2 c), are shown in the Figure 5.9.

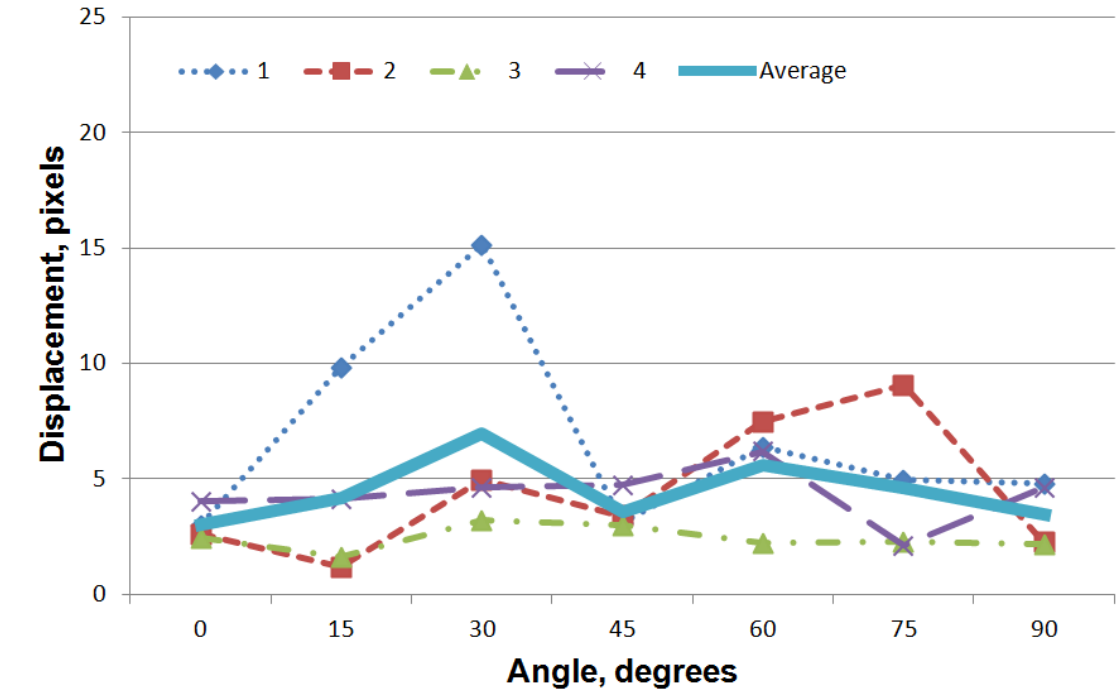

Figure 5.9. Average object displacement versus camera rotation.

Camera rotation has some effect on the accuracy, but again the environment seems to have a bigger effect. This is expected, as SURF is rotation invariant. The rotation does however somewhat change the area of the environment visible in an image. For example areas visible near horizontal edges are not visible anymore when the camera is rotated 90 degrees. This can also be seen by comparing the areas the image covers in the 0° and 90° cases in Figure 5.2 c). If the original image contains distinctive keypoints in these areas, the accuracy might decline when the camera is rotated.

Viewing distance has some effect on the accuracy. The average error for all the measured displacements from distance of two meters was 23 pixels while the corresponding value for four meters was 12.5 pixels. If the values for viewing angle above 20 degrees are discarded, then the average errors for two meter distance and four meter distance are 6.7 and 1,7 pixels, respectively. The median displacements were 3.4 and 1.1 pixels and standard deviations 9.5 and 1.9 pixels. All these results indicate that the method does not work well when used too close to the desired objects. Larger distance offers more background data which enhances the accuracy of the method. Particularly well this can be seen in the results from the environment 1, which contains a lot of plain white wall having little distinctive features. There many images from the two meter distance suffer from lack sufficiently distinctive keypoints.

The maximum usable distance was searched by measuring the accuracy by moving the camera as far from the TV as possible. Due to spatial constrains in the environments, only environment number 2 had enough room to make such measurements from more than 5 meters away and even there no other changes in the image could be used simultaneously. No increase in average displacement was observed up to 7 meters away, which was the maximum distance possible as the room was not long enough for longer distances.

5.3 Discussion

The results show that the method is accurate under restricted conditions, but suffers greatly from big changes in the viewing angle compared to the reference images. This is in line with the SURF´s abilities and limitations described in the literature. These can be partly overcome with certain workarounds like using more than one reference for a single environment. The accuracy of the method is good enough to enable it to be used in real world applications. However, very low lightning or big changes in viewing angle can produce errors which severely hinder the user experience.

The images used in the testing were very similar to the live images used by the actual application, but not identical. This is one possible source of error in the results. Also the test images were taken while the camera was firmly attached to a tripod, so the tests do not take into account possibly poorer image quality caused by the user’s movement with the camera. The subjective tests with live images in various environments do support the test results. This indicates that the difference between the test setup and real world usage scenarios is not significant.

The tedious reference addition process is a mandatory requirement for this approach. At the same time it makes the whole detection process simple, but also requires additional effort from the user before the application could be used. Also considerable changes, like redecoration, would require that the reference images are added anew.

The method is not sensitive to small occlusions since it utilizes local image features from the whole image instead just from the objects themselves. Obviously this does increase computational requirements as the amount of information to process is larger.

From the software design point of view, the results show that an implementation using freely available open source components can be made into a working object detection application. Aside from the modified keypoint matching process, the object detection parts of both implementations utilize OpenCV´s APIs without modifications to them. The keypoint matching process needed additional functionality that was not available from OpenCV or did not work as expected. Here the proposed additions to the commonly used method offer better accuracy and performance than the implementations from OpenCV. This might be due to errors in the OpenCV implementation at the time or because the implementation did not work as desired with the scenario used in this thesis.

The standalone mobile implementation shows that the whole detection process is possible to perform on the mobile phone. This does slow the process down however and the detection times for the first frames for the standalone mobile application are too slow for a commercially viable application. In subjective tests the time taken to detect objects from the first frame was several seconds, usually at least 4, and for the successive frames around 0.25 to 0.5 seconds. The client-server implementation fares better in this sense. The detection times in subjective tests were 1-3 seconds, but it requires a good network connection to a server with sufficient computational resources at all times and the network has a big effect on the overall detection time. This limits the possible usage scenarios for such an application.

There was no reliable way to compare the computational performance of the used PC and the mobile phone. The exact hardware was not known for either and comparing only processor clock rates was not sufficient. Also the software packages used were not built for a mobile device and this adds another unknown factor to the performance comparison considerations. Thus comparing the detection speeds of the two implementations was not sensible or even possible.

6 CONCLUSIONS

The open source tools can be used for building computer vision applications for mobile devices. What most of them lack in documentation and support they compensate with an active developer community, which also improves the chances that the tools are developed further. The developer community activeness proved to be a good criterion for selecting the tools to be used.

The readily available software components alone did not fulfil the requirements for the implemented applications. The proposed and implemented enhancements improved the performance of the selected object detection approach for the concerned scenarios. The results show that the used method is accurate enough for a commercial end-user application in most cases, but it has certain restrictions which limit the potential usage scenarios. Also the time taken to produce the detection results is too high for a pleasant user experience, particularly for the standalone mobile implementation.

As the thesis mainly focused on testing if the target scenario is achievable using the readily available tools, it left a lot of room for further investigation related to optimizing speed and accuracy as well as the user experience. The program code followed good programming practices in most cases but it was in no way optimized for performance, which leaves an open question whether a significant performance improvement could be gained by software optimization only.

The accuracy of the method depends on the image conditions, but the used parameters for the computer vision algorithm also play a big role. For the client-server implementation a good amount of time was spent on finding parameter values that perform well enough under various conditions. The parameter optimization was not done for the standalone mobile implementation which shows as poorer accuracy. One option would be not to try to find one set of universal parameters but instead try to find the connection between environment conditions, like lightning or image complexity, and working parameter values. This kind of a dynamic parameter setting would be an interesting topic to examine and could very well improve the performance of the approach. More optimized parameters would also provide better grounds for evaluating the potential of the proposed method for the used and other scenarios.

Even though the thesis focuses on computer vision and its usage in mobile software, considerable amount of time was also spent on designing a good user interface. The application to detect and interact with devices is not a simple one from the user interface point of view and improving the usability would be needed if a similar end user application would be built. The user interfaces in the implementations are based on the subjec-

tive views of the developers only, but for a good usability a much larger group of potential end users would need to be consulted.

The thesis shows that accurate computer vision based device detection can be implemented using readily available tools with certain additions. Open software components provided all the necessary tools for it. The accuracy of the proposed method is sufficient for end user applications and the accuracy increases when the physical area visible in the camera image increases. Computer vision and open source libraries provide the means for anyone with basic software development skills to build interesting and impressive new applications.

REFERENCES

[1] Lowe, D.G., Object recognition from local scale-invariant features. Proceedings of the Seventh IEEE International Conference on computer vision volume 2, Corfu, Greece, September 20-27, 1999. IEEE. pp. 1150-1157.

[2] Bay, H., Tuytelaars, T., Van Gool, L., SURF: Speeded Up Robust Features. Computer Vision–ECCV 3951(2006)2, pp. 404-417.

[3] Tuytelaars, T., Mikolajczyk, K., Local invariant feature detectors: a survey. Foundations and Trends in Computer Graphics and Vision 3(2008)3, pp. 177-280.

[4] Baumberg, A., Reliable Feature Matching Across Widely Separated Views. Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Hilton Head, USA, June 13-15, 2000, IEEE. pp. 774-781.

[5] Matas J., Chum, O., Urban, M., Pajdla. T. Robust Wide Baseline Stereo from Maximally Stable Extremal Regions. Image and Vision Computing 22(2004)11, pp. 761-767.

[6] Fasel, B., Van Gool, L., Interactive Museum Guide: Accurate Retrieval of Object Descriptions. Proceedings of the 4th international conference on Adaptive multimedia retrieval: user, context, and feedback, Geneva, Switzerland, July 27-28, 2007, Springer. pp. 179-191.

[7] Lee, S.H., Choi, J., Park, J., Interactive e-learning system using pattern recognition and augmented reality. IEEE Transaction on Consumer Electronics 55(2009)2, pp. 883-890.

[8] Se, S., Lowe, D., Little, J., Mobile Robot Localization and Mapping with Uncertainty using Scale-Invariant Visual Landmarks. The International Journal of Robotics Research 21(2002)8, pp. 735-758.

[9] Brown, M., Lowe, D., Recognizing Panoramas. Proceedins of the Ninth IEEE International Conference on Computer Vision volume 2, Nicce, France, October 13-16, 2003. IEEE Press. pp. 1218-1225.

[10] Mikolajczyk, K., Shmid, C., Scale & Affine Invariant Interest Point Detectors. International Journal of Computer Vision 60(2004)1, pp. 63-86.

[11] Szeliski, R., Computer Vision: Algorithms and Applications. 2011, Springer. 832 p.

[12] Lowe, D.G., Distinctive Image Features from Scale-Invariant Keypoints. International Journal on Computer Vision 60(2004)1, pp. 91-110.

[13] Viola, P, Jones M.J., Robust Real-Time Face Detection. International Journal of Computer Vision 57(2004)2, pp. 137-154.

[14] Nixon, M., Aguado, A.S. Feature Extraction & Image Processing. 2nd edition. 2008, Academic Press. 424 p.

[15] Neubeck, A., Van Gool, L., Efficient Non-Maximum Suppression. Proceedings of the 18th International Conference on pattern recognition, volume 4, Hong Kong, China, August 20-24, 2006. Los Alamitos, CA, USA, IEEE Computer Society. pp. 850-855.

[16] Eshan, S., Kanwal, N., Bonstanci, E., Clark, A.F., McDonald-Maier, K.D, Analysis of Interest Point Distribution in SURF Octaves. 3rd International Conference on Machine Vision, 2010, pp. 411-415.

[17] Brown, M., Lowe, D.G., Invariant Features from Interest Point Groups. Electronic Proceedings of the 13th British Machine Vision Conference, Cardiff, UK, September 2-5, 2002, BMVA. pp. 253-262.

[18] YUV Image Formats [WWW]. [Referenced 27.5.2012]. Available at http://www.fourcc.org/yuv.php.

[19] Fischler, M.A., Bolles, R.C., Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. Communications of the ACM, 24(1981)6, pp. 381-395.

[20] Python Programming Language [WWW]. [Referenced 28.5.2012]. Available at http://www.python.org/.

[21] Qt – Cross-platform application and UI framework [WWW]. [Referenced 28.5.2012]. Available at http://qt.nokia.com/.

[22] What is PyQt? [WWW]. [Referenced 28.5.2012]. Available at http://www.riverbankcomputing.co.uk/software/pyqt/intro.

[23] OpenCV [WWW]. [Referenced 28.5.2012]. Available at http://opencv.willowgarage.com/wiki/.

[24] pyopencv [WWW]. [Referenced 28.5.2012]. Available at http://code.google.com/p/pyopencv/.

[25] RPyC [WWW]. [Referenced 28.5.2012]. Available at http://rpyc.sourceforge.net/.

[26] VLC multimedia player and framework [WWW]. [Referenced 28.5.2012]. Available at http://www.videolan.org/vlc/index.html.

[27] Qt Quick – including QML, Qt Creator and the Qt Quick UI runtime [WWW]. [Referenced 28.5.2012]. Available at http://qt.nokia.com/qtquick/.