UPPSALA
UNIVERSITET

IT 13 007
Examensarbete 15 hp
Januari 2013

Object recognition using the OpenCV Haar cascade-classifier on the iOS platform

Staffan Reinius

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Object recognition using the OpenCV Haar cascade-classifier on the iOS platform

Staffan Reinius

Augmented reality (AR), the compiling of layered computer-generated information to real-time stream data, has recently become a buzzword in the mobile application communities, as real-time vision computing has become more and more feasible. Hardware advances have allowed numerous such utility and game applications to be deployed to mobile devices. This report presents a high-level implementation of live object recognition of automobile interiors, using Open Source Computer Vision Library (OpenCV) on the iOS platform. Two mobile devices where used for image processing: an iPhone 3GS and an iPhone 4. A handful of key-feature matching technics and one supervised learning classification approach were considered for this implementation. Speeded Up Robust Features (SURF) detection (a key-feature matching technique) and Haar classification (supervised learning approach) were implemented, and Haar classification was used in the final AR prototype. Although the object classifiers are not yet to satisfaction in terms of accuracy, a problem that could be overcome by more extensive training, the implementation performs sufficiently in terms of speed for the purpose of this AR prototype.

Handledare: Amen Hamdan
Ämnesgranskare: Anders Hast
Examinator: Olle Gällmo
IT 13 007
Tryckt av: Reprocentralen ITC

Contents

1 Abbreviations

- AR Augmented Reality
- FLANN Fast Library for Approximate Nearest Neighbors
- FREAK Fast Retina Keypoint (keypoint descriptor)
- GENIVI Geneva In-Vehicle Infotainment Alliance
- IVI In-Vehicle Infotainment
- OpenCV Source Computer Vision Library
- ORB oriented BRIEF (keypoint detector and descriptor extractor)
- QR Code Quick Response Code
- SURF Speeded Up Robust Features

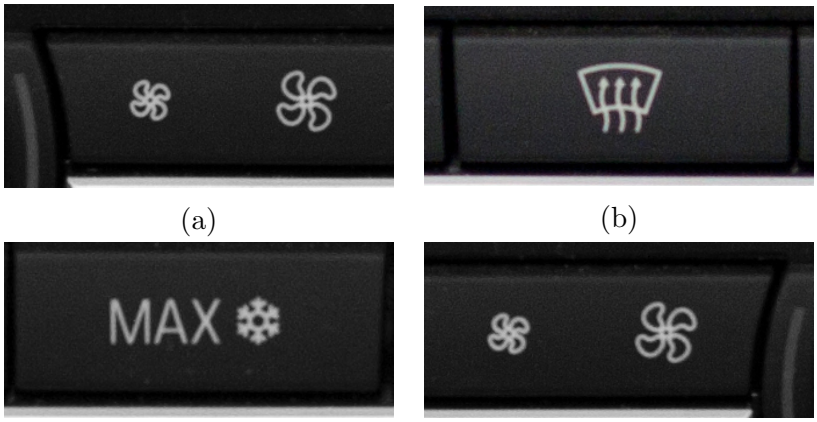
2 Introduction

In-Vehicle Infotainment (IVI) systems is an expanding field in the automobile industry. Cars from the BMW Group released in ECE/US have the feature of letting the user connect a mobile device to the head unit of the car, interweaving the mobile device with the vehicle. The BMW IVI system is soon to be released to the Chinese market (summer 2012), supporting widely used Chinese mobile device applications. Such mobile-to-car interweaving allows on the one hand the user to interact with their mobile phone through the larger display of the IVI system, accessing telephony, internet services, news, music, navigation, etc. and on the other hand allows the mobile to access virtually any data from the car head unit and system busses. This allows for infotainment applications in the other direction, providing information such as driving statistics, indicators, mileage and so on. The development of the BMW IVI system is conducted under a Linux-based open-source development platform delivered by the Geneva In-Vehicle Infotainment (GENIVI) Alliance. [1]

A number of automobile manufacturers, including the BMW Group, MG, Hyundai, Jaguar, Land Rover, Nissan, PSA Peugeot Citroen, Renault, SAIC Motor, use this Linux-based core service and middleware platform as an underlying framework. [2]

The aim of this project was to develop an object recognition module to an iPhone AR prototype application for the BMW Connected Drive Lab, located in Shanghai China. The ambition was to implement this prototype so that it could be used as a basis for an interactive diagnostics tool, handbook or similar, allowing further information about the identified objects and graphics to be layered on screen. The application, on prototype level, is a stand-alone tool not dependent on connection or communication with the existing IVI-system. Future versions of this application could be integrated with the IVI system and present diagnostic data on the mobile device. Such an AR module could be useful in a number of tools accompanying a car, e g. using the mobile device as a diagnostic tool (to check oil, washer fluid, tire pressure etc.), as an interactive car handbook or as a remote control towards the IVI system.

The project was divided into two parts, the first focused on object recognition and the second focused on the user interface interaction and graphical overlay onto the camera-provided images. The first part is presented in the current bachelor thesis, and the second part is covered by the bachelor thesis of Gabriel Tholsgård [3], with whom collaboration have been extensive. The current report describes the work of implementing image processing and applying object recognition to parts on the car dashboard provided in a video stream, and choosing an efficient approach that takes the relevant invariance properties in to account.



(a) (b) (c) (d)

Figure 1: These objects were chosen for object recognition and represents four buttons for climate control on the car dashboard.

2.1 Objectives

The goal of this project was to build an AR application for an iOS mobile device, using its built-in camera and project an OpenGL animation interface as an overlay on the detected objects, more specifically:

I To construct a prototype module able to recognize four objects (fig. 1) on the car dashboard using OpenCV on the iOS platform.

II To present an augmented reality OpenGL animation overlay on the camera image representing the detected object (approached in [3]). Those OpenGL animations should also be allowed to interact with, they should work as buttons.

III To combine the implementation of the two previous goals, achieving a generalized AR prototype.

2.2 Limitations

Within the scope of this thesis project it was not intended to construct a finished application (ready for the market). The finished project is a prototype for object recognition and displays animations as an overlay on the camera image. The object recognition task was limited to identify four buttons on the cars climate panel (fig. 1). This prototype is not integrated with the Linux-based core service or middleware, but where intended to be a stand-alone iOS application, partly

because there are no wireless sensors between the car head unit and the phone today, but more importantly to limit the work to fit the time frame of the project.

3 Background

3.1 Augmented Reality

AR is the idea of adding computer-generated data (e g. graphics, sound, GPS) to real-time data; in contrast to Virtual reality where the real world is substituted by a simulated. With object recognition such AR application, called vision based, can become interactive by adding on a user interface to some detected objects on the camera image. Another common way of knowing what to display is the location-based approach, often using GPS [4].

In a early stage of this project the location-based approach was also considered, more precisely the possibility to get the local coordinates of the telephone in respect to the cars head unit, and combine this info with the tilt of the phone (accelerometer) and construct a 3D map of coordinates, tilt and corresponding objects. Such a system would perform really well in terms of computational complexity (constant), but would be exhausting to construct, and each car model would need its own coordinate map. As earlier mentioned there is no wireless communication between the phone and the head unit, and even if there would be, the issue of getting the exact local 3D coordinates might be hard to solve, it would probably require multiple positioning sensors.

The vision based approach was chosen, and the various aspects of AR is more thoroughly described in [3]. It was specified that this iOS AR application was to be made without using markers or AR-tags, but instead use object recognition methods more commonly used in other areas – in augmented reality terms called natural feature tracking [4].

3.1.1 Augmented Reality applications in cars

AR seems to be more and more explored in many areas: interactive commercial apps (QR-tags), game set in the real world, GPS-based AR applications highlighting landmarks and such, to mention a few. Within the auto industry ideas has been proposed on combining data from outside sensors with projections of AR on the windshield, highlighting traffic signs, pedestrians and other traffic hazards, or to use the back seats windows as computer screens highlighting landmarks and enable interaction. From the driving seat it would obviously not be safe to interact with a touch screen windshield, it could instead be a user scenario for passengers, and the same applies to interaction with a handheld device to control the infotainment system. The issue is mainly how to project large images on a windshield.

BMW makes use of AR today by providing a smaller Head-Up-Display in the front window (projected for the driver) with some essential information such as speed and navigation information, which is an interesting feature mainly in terms

of safety by keeping the driver's eyes on the road.

Even though these AR application scenarios are interesting, they do not directly touch the work of this thesis, which instead implies a scenario where the car is parked or where it is the passengers using the application. During the work process of this thesis, three scenarios for how this prototype could be used have been discussed: that it might be incorporated in a diagnostic tool, a car handbook or a remote-control-application.

The maybe most obvious application would be to use a handheld device as a remote control, since such devices adds wireless communication to the interaction with the car. A remote control could be useful from the back seat, and here AR could be used. A diagnostic tool could preferably also be developed implementing a traditional UI, and such a feature is already implemented in the existing IVI-system. AR may not be interesting primarily for its practical use here, but it might add a feeling of "real" interaction which could be desirable in an application used for showing features of the car from outside, or for car game applications.

Today the interaction with BMW infotainment system is done through a joystick or via the steering wheel and it is also possible to interact with the IVI-system via the telephone device connected by USB cable.

3.2 Object recognition

One of the main fields within image analysis is that of recognizing objects, and there are a variety of such techniques coming from the field of computer vision and machine learning, which are two fields that often seems to go hand in hand, decision trees, cascade classifiers etc. can be trained based on features (distinct subregions in the image) and so forth. The following paragraphs are aimed to be a very brief introduction to two approaches: feature-based keypoint extraction techniques and one supervised learning approach based on rejection cascades.

3.2.1 Local invariant feature detectors

An often used group of object recognition methods are the local feature-based [5]. They are built on the notion of trying to limit the set of data for processing and also provide invariance to different transformations. Instead of considering global features, the local feature detection approach finds anchor points in the image whose local region are defined by some differences in intensity or texture. These intensity or texture changes are found by running a kernel over the target image; a kernel is a region of pixel computations computing the amount of gradient change (the derivative) within this region. These features or interest points can, depending on the detector, be corners, blobs, T-junctions etc., and these should

in the ideal case be identifiable after the image has transformed in different ways. [5]

Let's look at the Harris corner detector as an example. Intuitively a perpendicular corner would be found when the horizontal and vertical gradients (within a sub-window of pixels) sums up to a large value, since in that case there is contrast in both horizontal and vertical directions – but a rotation of the image would hide this feature if the measurement was done perpendicular. The Harris Corner detector (in this example) compensate for such a rotation by applying eigenvalue decomposition to the second moment matrix:

$$M = \begin{bmatrix} I_x^2(x) & I_x I_y(x) \\ I_x I_y(x) & I_y^2(x) \end{bmatrix}$$

I_x and I_y are the respective sum of pixel intensity change in the x and y direction at point x. If M has two large eigenvalues it is centered a round a corner. [5]

In contrast to many typical Template-based methods, local feature-based can exhibit many types invariance, even to partial occlusion [5].

Object recognition techniques are often applied on gradient images for several reasons: gray scale images are generally more robust to variation in illumination, and matrixes of singular brightness values are more efficient in terms of memory consumption and less demanding in terms of processing speed [5].

After extracting local features from two imaged, those set of feature are compared to find out what feature in one image correspond to feature in the other, and if a threshold (number of features) is exceeded a match is found. [5]

3.2.2 Speeded Up Robust Features

The first choice of recognition method was SURF [6] – a scale-invariant feature-based technique. It is an approach designed for efficiency that at the same time have a high level of invariance to the transformations and conditions expected in the setting given [5].

SURF uses box-type filters using integral images to approximate the second-order Gaussian derivatives of the Hessian-matrix [5]. That is, SURF exploits the same principle of looking for two eigenvalues with the same sign (as described above for the Harris Corner detector), only it is based on approximations for each entry of the Hessian matrix using one of three filters, fig. 2(proposed in [6]) :

$$H(x, \sigma) = \begin{bmatrix} L_{xx}(x, \sigma) & L_{xy}(x, \sigma) \\ L_{xy}(x, \sigma) & L_{yy}(x, \sigma) \end{bmatrix} \qquad (1)$$

If we let D_{xx}, D_{yy} and D_{xy} be approximations for L_{xx}, L_{yy} and L_{xy} respectively, the determinant of the Hessian matrix can be approximated as

$$det(H_{approx}) = D_{xx}D_{yy} - (0.6D_{xy})^2 \qquad (2)$$

for a Gaussian with $\sigma = 1.2$ (finest scale) and a filter (fig. 2) of 9×9 pixels. Here, 0.6 represent a balancing of the relative weights with scale, computed as:

$$\frac{|L_{xy}(1.2)|_F|D_{xx/yy}(9)|_F}{|L_{xx/yy}(1.2)|_F|D_{xy}(9)|_F} = 0.6 \qquad (3)$$

Were $|x|_F$is the Frobenius norm.

For scale invariance these filers are applied to different scales over the image. The processing time over different scales over some point is size invariant due to the use of integral images [7]. To get rotation invariance each interest point is given an orientation vector and this vector is the maximum of response (gradient change computed with Haar wavelets) around the point. When the extraction of the descriptor is done some matching technique is applied to find correspondence between the descriptors (feature vectors); when using OpenCV the FLANN library (Fast Library for Approximate Nearest Neighbors) can be used, applying nearest neighbor search.

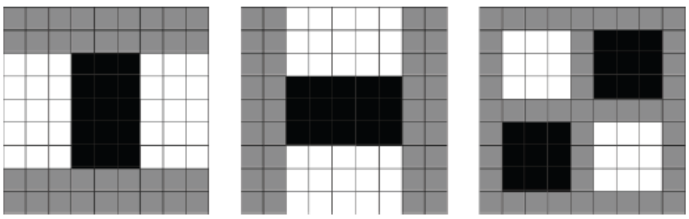


Figure 2: These *filters* (kernels) approximate the Laplacian of Gaussians ($L_{xx}L_{yy}$ and L_{xy}) by applying the weights of -1 to the white areas, 2 to the black and 0 to the gray areas.

3.2.3 Haar classification

Many approaches based on machine-learning have the advantageous of being computationally more efficient in object detecting, although generic detectors (feature matching) often performs better in terms of localization, scale and shape [8]. Haar classification is a tree-based technique where in the training phase, a statistical boosted rejection cascade is created. Boosted means that one strong classifier is created from weak classifiers (see fig. 3), and a weak classifier is one that correctly

gets the classification right in at least above fifty percent of the cases. This buildup to a better classifier from many weak is done by increasing the weight (penalty) on misclassified samples so that in the next iteration of training a hypothesis that gets those falsely classified samples right is selected. Finally the convex combination of all hypotheses is computed. (fig. 3)).

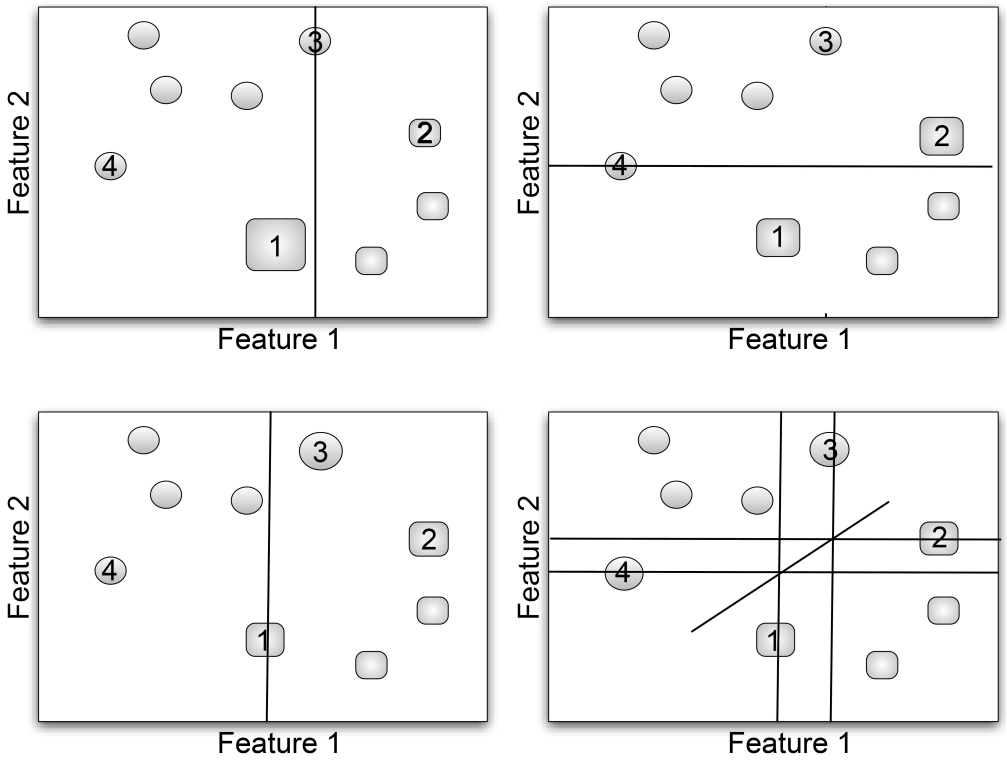
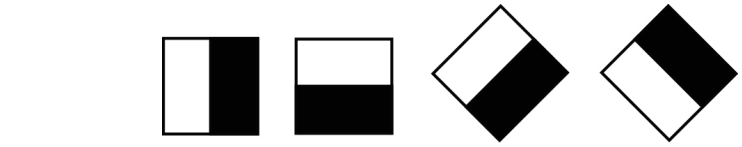


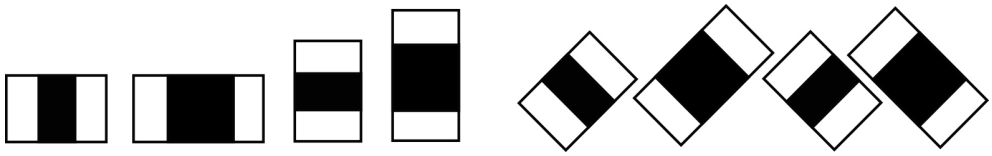
Figure 3: *Example illustrating boosting.* (a) A hypothesis (line) is selected, misclassifying object 1, the weight of object 1 is increased which will affect the choice for picking the next hypothesis (a cheap hypothesis will be selected). (b) The next hypothesis misclassifies object 2, and the weight is then divided between 1 and 2. (c) The next hypothesis misclassifies object 3, and the weight is now divided between 1, 2 and 3. (d) After picking a last hypothesis the convex combination of all hypotheses is computed. This example is based on [9].

There are four boosting methods available for Haar training in OpenCV: Real Adaboost, Discrete Adaboost, Logitboost and Gentle Adaboost.

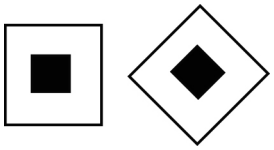
That Haar classification uses a rejection cascade means that the final classifier consists of a cascade of (many) simple classifiers, and a region of interest must pass all stages of this cascade to pass. The order of the nodes is often arranged after



(a) Edge Features



(b) Line Features



(c) Center-sorround Features

Figure 4: These Haar-wavelet-like features are computed by adding the light regions and subtracting the dark regions [8]. The image is originally from [10].

complexity, so that many feature candidates are ruled out early, saving substantial computation time [8].

As input to these basic classifiers, that builds up the cascade, comes Haar-like features that are calculated according to the figures in fig. 4.
When the application is running a pane of different sizes is swept over the image, computing sums of pixel values based on these features, using integral images, and applies the trained rejection cascade (see next chapter).

4 Methods

OpenCV is a computer vision library written in *C* and *C++*, and contains over 500 functions associated with vision, and also contains a general-purpose machine-learning library. OpenCV was chosen since it is open-source and free for academic and commercial use, and it is widely used and well documented. [8] Most importantly OpenCV compiles on the iOS platform, along with the additional frameworks AVFoundation, ImageIO, libz, CoreVideo and CoreMedia.

Initially many different approaches to object recognition where considered since the OpenCV interface makes it easy to switch between methods. Developing for mobile devices increases the demand for efficiency and the key criterion were of course to choose a well performing technique in terms of efficiency and level of invariance, with more emphasis on efficiency as live video would be processed and furthermore CPU and RAM is comparatively limited on the intended devices.

SURF and Haar-feature classification became the main candidates for recognition; SURF because it is designed for efficiency and meets all intended invariance requirements [5], and the Haar classifier since it is very efficient, but then rotational invariance is slightly compromised [8] depending on what features are used under training. As described in more detail in chapter four the Haar classifier was finally chosen. Both make use of integral images, e g. sum of pixel values in a rectangular region of an image.

4.1 Choosing recognition method based on performance and invariance properties

The buttons to be recognized (fig. 1) are by themselves unchanging, although there may be ambient light and background light, these objects do not deform. From the drivers and passenger seat there is essentially only one vantage point, you can obviously not go around a car dashboard or turn it. As the application where to be deployed on a hand held device, which can be twisted, a method that was not too sensitive to rotation had to be chosen. And it was also considered that the iPhone screen is displayed in two modes landscape mode and upright portrait mode. Another property needed to be considered due to that the device is hand held was scale. The mobile phone could be brought close to the panel or be held from a distance. The car dashboard is mainly illuminated by natural light from outside, or by night from lights in the vehicle, so the method of choice needed to have a high level of illumination invariance.

With this in mind our first choice, as earlier mentioned, was SURF (Speeded Up Robust Features detection). But some time into the project, the SURF implementation was perceived not to meet the requirements in terms of speed. It

performed well on a laptop but not on the iPhone 3GS, hence the OpenCV Haar classification technique became the choice of approach.

4.2 Data collection and sample creation for Haar training

With OpenCV comes a utility for creating traning samples from one image, which can automatically apply alteration in background, lightning and rotation for the output images. This approach is suiting for cases where the object does not vary in appearance, e g. training on logos, labels and such. The documentation for using the createsamples utility could be found in the OpenCV install directory under *OpenCV/apps/HaarTraining/doc*, though the information given where sparse, e g. all parameters where not explained.

First, one high-resolution image where captured on the climate panel with a Canon EOS 30D, from which smaller images of the individual buttons where cropped. The fact that another camera was used to collect the samples (than of the handheld device's) did likely not impact the performance of classification, since the output samples are very minimized to make the training phase feasible. This would probably have been an issue with a key-point matcher recognition method.

Random negative sets where chosen so that they varied in gradients. How the samples should vary could be controlled by a number of command line parameters:

- Name of the output vector containing the positive samples for training
- Source image (image of button)
- A path to the directory with negative background samples
- How many positive samples the output vector should contain
- A number of parameters on how the object image and background should be illuminated
- Maximum rotation angles for the object image on the background image in radians
- Output sample width and height

The vector created, in our case only contained 500 samples but where relatively big: width 50 pixels and height 40, since we could see that the white text and logos on the buttons almost disappeared when we trained with more typical values [8] (like 20 by 20 pixels).

4.3 Haar training

The OpenCV createsamples-utility outputs a vector with samples, this vector is used as input to the OpenCV haartraining utility. Some of the important command line parameters for this utility are the following:

- File name of vector with positive samples
- File name with background image names (not containing objects)
- Number of positive samples in the sample vector
- Number of negative samples desired
- Number of training stages
- Minimal desired hit rate for each stage
- Maximal desired false alarm rate for each stage
- Number of splits, the number of levels in each nod of the cascade
- The size of dedicated memory in MB
- Whether the object contains symmetry or exhibit non-symmetry
- Mode, what set of haar features (see fig. 4) to use (Basic [only upright], Core or All) to bee used
- Sample width and height

The background images are used to create negative samples, from panes of these images, and here one hundred images were chosen from a private photo library which showed variance in gray scale gradients. The number of training stages combined with minimal desired hit rate and maximal desired false alarm rate will determine the total hit and false alarm rate; for instance using a false alarm rate of 50% and a hit rate of 99.9% would yields a total false alarm rate of $0.5^{20} = 9.6e - 07$ and a hit rate of $0.999 = 0.98$ though the cascade. [8] The number of splits is the number of levels in each nod of the cascade, this is typically one (a "decision stump"), but can be two or three for more complex objects. More dedicated memory and object symmetry – if the object exhibits symmetry properties – can be used to speed up training. Sample width and height should be the same as from when the samples where created. Mode can be used to explore more advanced features, but this is also a trade-off against slower training – for this project only the basic upright features where used (the upper three in fig. 4]).

4.4 Work diary

In the middle of February a first meting was held at the BMW office, where a handful of topics was discussed, this recognition task being one of them. This task demanded some insight and knowledge of Objective-C, C++ and Computer Vision; most research were done on the latter as this was an area where the experience was limited, where as the IDE Xcode, Objective-C and C++ where familiar topics.

After a phase of project planning and writing and handing in a proposal, different AR libraries was considered, such as the computer tracking library ARToolKit, however the decision was made to use OpenCV for recognition (as discussed earlier) and OpenGL ES for animating the UI [3]. The complete task was divided in to two thesis projects, including the respective subtasks of recognition and graphical overlay and interaction.

The first weeks after planning, middle of march, where a learning period for basic OpenCV, reading [8] and experimenting in OpenCV compiling on a laptop. The structure of OpenCV lets one easy switch between different detectors (feature extractors) and feature correspondence methods, making it easy to test different approaches. The decision to use SURF for the iOS implementation was made based on correspondence with our topic reviewer and a performance chart in [5] stating that SURF is rotation and scale invariant, very efficient but slightly compromised regarding repeatability, location accuracy and robustness.

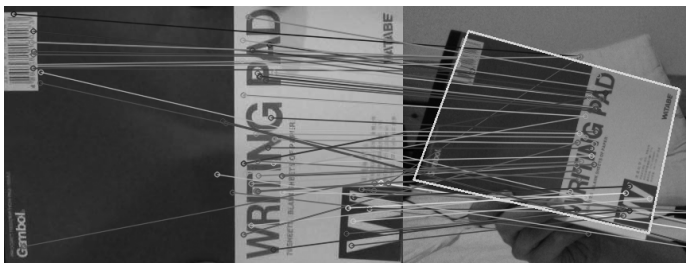


Figure 5: This image is a screenshot from a running application on the PC implementing SURF, detecting a writing block.

When porting to the iOS platform a preexisting open source iOS project was used developed by the developer group Aptogo [7] integrating OpenCV with iOS, including a precompiled OpenCV framework. On the iOS platform there are several aspects regarding OpenCV that differ from a PC environment, especially in using the highgui module, which normally handles interaction with the operating system: accessing cameras, handle user events, displaying images and graphics in images etc. On the iOS no video preview is supported, and frames have to be pulled and display manually from the hihgui *VideoCapture* class. This preexisting project was designed to be re-useable by sub-classing its integration feature and

allowed to directly use the only slightly modified OpenCV code from developing on the laptop.

Through April and May, weekly meetings were held, where the working progress was reported to the thesis project supervisor and deliverables for the coming week where set.

In the middle of April a SURF implementation was running on the device. At this point the object in the car to be recognized had not been specified, and in the meantime a tablet/writing pad was used for testing, which differed a lot from the final objects that where chosen. This implementation was performing well on the laptop but did not work well on the handheld device: it only measured to be able to process 0.5 frames per second. This was a major concern since the aim was to recognize four objects. Then an effort could have been put to optimize the implementation, but instead other approaches was looked into, using a better performer (in terms of processing) based on a learned boosted rejection cascade, and crucial for this decision being made was knowledge of the automatic sample generator for the Haar Classifier. (See 4.2)

The training (see 4.3) of the final classifiers took about three days to complete on two computers, running two parallel training jobs on each (one for each classifier). Before this the sample-creating- and training -utility was tested on the (just mentioned) writing pad (fig. 5) with fairly good results. But the objects that finally was picked for recognition (fig. 1), turned out to be harder to produce good classifiers from, probably because they were to small and lacked "blocky" features [8]. So finally the set of classifiers were too tolerant. In retrospect a larger and easier targets than the small buttons, e g. the entire climate control panel or similar, should have been chosen for the recognition task.

The last week was dedicated to merging the UI code with the object recognition code.

5 Results

This chapter focuses on implementation and the final outcome of the project, which is discussed further in the next chapter.

5.1 Performance and accuracy

The application performance were measured in fps, but the level of accuracy were only visually inspected by simply looking at the running application:

Method	fps	Object	Acurracy
Haar Classification	0.6	4 Buttons (fig. 1)	Poor, finding lots of false positives outside the intended environment.
Haar Classification	1.8	Tablet (fig. 5)	Ok from a straight forward angle, to sensitive to changes in rotation and change in angle.
SURF	0.5	Tablet (fig. 5)	Good, insensitive to changes in rotation and change in angle but unresponsive.

This table is not intended as a measurement of how the methods perform in general, but rather only presenting the result, which is implementation and platform specific. It would probably not be meaningful to compare such distinctive methods on a general basis (since the environment differ from task to task).

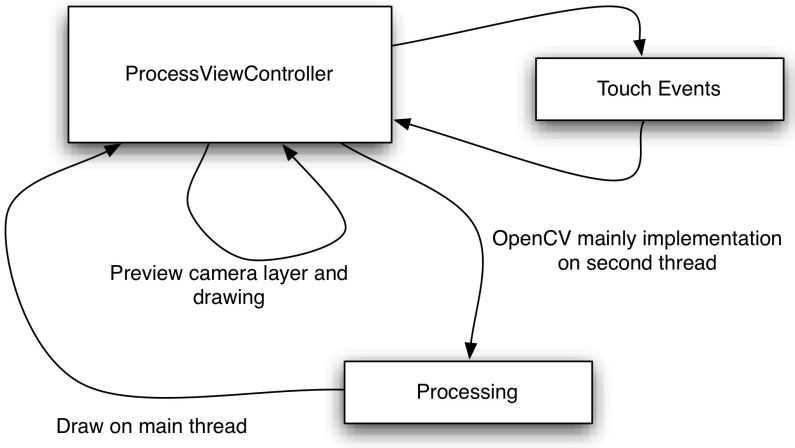


Figure 6: System overview.

5.2 Implementation and system design

Developing for iOS application using OpenCV differs from a PC environment in that the iOS frameworks are used for UI and accessing hardware through delegates, instead of directly using the OpenCV highui module for interaction with the operative system. In the OpenCV highui, video capture is supported by the class *VideoCapture* from where video frames can be grabbed as *cv::Mat* objects (data structure containing an image).

In this implementation the view-controller *VideoCaptureViewController* is an AVFoundation delegate (audio and visual media framework) and previews a video layer and draws on the main thread, as the diagram below indicates, but dispatches the classification OpenCV processing work to its method *processFrame:videoRect:videoOrientation* on a different thread. The reason is that if processing, camera overlay and UI all would have been handled on the main queue the frame rate would have been low. Also touch events are handled on its own queue to achieve interaction responsiveness.

VideoCaptureViewController also handles images instantiation and conversion to gray scale.

Even though the *processFrame:videoRect:videoOrientation* method searches for all four objects for each frame that is pulled – e g. it applies the *cv::cascadeClassifier* method *detectMultiScale* on the image for all trained classifiers – it performs quite well, between one and two frames per second. Here, some effort could have been put to implement a cache for what objects should be prioritized based on what objects recently was found if time had been sufficient. But as of now the *Pro-*

Figure 7: A screenshot of the running application (with some texture error, see discussion in next chapter).

cessViewController instantiate four Cascade Classifier objects, one for each xml-cascade-classifier file that was trained. The method *detectMultiScale* is run on the cascade object with the following parameters: image object (a *cv::Mat*), scale factor (how much the search pane size is reduced for each iteration), vector of rectangles who are updated as a side effect if there is a match, minimum neighbors (collections with fewer features are treated as noise) [11].

The vital part of the Haar classification is the following:

```
//Create a path to the classifier
NSString *cascadePath = [[NSBundle mainBundle] pathForResource:

cascadeFilename ofType:@"xml"];

...

// Instantiate a vector to hold the coordinates of the matches
std::vector<cv::Rect> objects;

// Apply detection.
cascade.detectMultiScale(mat, objects, 1.1,3,

CV_HAAR_FIND_BIGGEST_OBJECT, cv::Size(10, 10));

// When the processFrame:videoRect:videoOrientation have

//processed the image it dispatches back to the main thread
//a vector of rectangles for where to draw the animations:
```

When the processFrame:videoRect:videoOrientation have processed the image it dispatches back to the main thread a vector of rectangles for where to draw the animations:

```
// Draw on main queue
dispatch_sync(dispatch_get_main_queue(), ^{
        [self displayObjects: objects
                forVideoRect: rect
          videoOrientation: videoOrientation];
    });
```

6 Discussion

The result in table 5.1 shows that the Haar Classification implementation in the top row performed well in terms of speed, four images with a frame rate of 0.6 s which accounts for 2.4 cascades each second. This can be compared with the SURF implementation with a frame rate of 0.5 per second, which implies it would take about two seconds to process one image and around eight seconds to process four. One reason for the good performance in terms of processing speed for the Haar Classifier (trained on the buttons) might be that the cascades were simple, but this is also the reason for the poor performance in regards to accuracy.

The first Haar Classification implementation, trained on a tablet (second row), showed good performance in speed (1.8 fps) and acceptable performance in accuracy, and the cascade was more complex, probably due to that the object (the tablet) had more distinct features. This was what was wanted when the last classifiers (on the buttons, top row) were trained, but they underperformed, which was expressed in that they where oversensitive: the different objects where sometimes confused and false positives (objects not trained on) could easily be found outside the control panel.

The Haar Classifier is often used for detecting pedestrians, body parts or faces (the Haar Classifier is some times called the Face detector [11]) and such classifiers are included within OpenCV. But it is also said to work well on logos and such with a typical viewing point [11]. And it was partly this that was noted for choosing the method. Furthermore cascade classification for Haar-like features is said to work well for “blocky” features with characteristic views [8]. This could easily be misinterpreted – it could mean ”blocky” as in sharp edges, or larger blocks with similar gray scale. The number of training samples could have been increased if the object would have been more distinct, since then the samples could have been reduce in size. Furthermore, the classifiers would probably have been stronger if they would have been trained with more advanced features.

When the decision to use Haar Classification for the finale version of the prototype was made, it was based on the result of the test made on the test object (the writing pad, entry two in the table), and the decision to use the final weeks of the project to train Haar classifiers was made before it was specified what object to be processed in the final version of the prototype. The buttons on the dashboard lacked sufficiently distinct features and it turned out not to be a good (but hasty) choice of set of objects (this is discussed more in the next chapter).

But considering the overall result, the outcome of the use of a Haar classifier was in the end closest to the project specification.

As for the screenshot in the previous chapter depicting the running application, there are two possible reasons why there are texture errors in the animation of the dashboard buttons, one might be that the data of the texture file is incorrectly read in; the other is that the settings of the view rendering are incorrect. This is more thoroughly discussed in [3], and these issues should be relatively easy to fix.

7 Conclusions

The finale object recognition module with xml cascades trained on the buttons (fig. 1) is performing below expectations. The over all experience is that even if the module finds the objects and often distinguish the different keys, it often finds false positives outside the climate control panel, e g. objects with white marks on black background, like on a computer Keyboard.

It was wanted for the final classifiers (top row in 5.1) to performed equally well as the test classifier (second row in 5.1), but this was not achieved since the target object turned out to be difficult to train. For the purpose of this prototype it would have been better to choose some larger objects in the car for the recognition, for instance the whole climate panel or the head unit.

7.1 Assessment of the work process

Object recognition is not a trivial task even when using high level libraries since each setting has it's own requirements and there is no best overall method. It is probably preferable to have a more pragmatic and practical approach, based on testing, rather than a strictly theoretical. In retrospect there are some things that could have been handled differently in order to achieve a better result in the end.

A more thorough planning and research phase could have lead to a better approach. As experience was slight in the field of computer vision, a great deal of effort went in reading up on OpenCV while studying computer vision theory and to test various methods on different object with "quick hacks". Although writing code is a great way to learn, one often gets sidetracked, and it is time-consuming to work on code that will never be used.

From the start it was only decided that the focus would be direct towards the dashboard. And that was clearly a mistake not to early decide exactly what objects to process. The approach was instead to think of what invariant properties should be considered for some object on the dashboard, and then to choose some method that meet the requirements and was fast (above robustness). If it had been specified early what objects would be processed, it would have been easier to tackle the task from a more practical approach.

It would have been good to conduct an own performance evaluation of the different methods (e g. Keypoint matchers) that took into account the circumstances specific to this task.

Another thing that could have been handled differently was the planning of the overall system. Rather than divide UI and interaction and object recognition into two subtasks, each expressed in separate programs, the entire system design

should have been outdrawn from start, and allow for smaller changes as the work proceeded. When merging the two projects a lot where done in an ad-hoc fashion, which resulted in that the final project was not very modular and generic.

7.2 Future work

If more work were to be done on this project the main priority would be to develop better classifiers, choosing a larger object, have more samples and use more advanced feature (not only up right ones). This could be done with a little effort.

An alternative approach to improve the recognition robustness of the cascade classifier, is to look into optimizing a implementation of a local keypoint extraction-based technique. The FAST and SUSAN detectors would then be two main candidates, based on a performance chart in [5] (page 257). Recently, in the 2.4.2 release of OpenCV (released in july 2012), a new keypoint descriptor called FREAK has been added to the library and is clamed to be very fast and " superior to ORB and SURF descriptors"[13]. The main issue for such approach would be to scale it up and make it able to identify a big set of objects.

On the other hand natural feature tracking (marker-less AR) is more difficult than marker-based tracking (using QR/AR-tags) in most, if not all cases where marker-based techniques are applicable, since such tags are created for recognition (containing clear features).

Maybe it was not realistic to think that a local feature based approach would be a good base for a system able to identify many objects. The issue of what object recognition systems are buildable is very specific to situation and environment. It would be achievable to construct a real time object recognition system based on local features for a small set of objects, but for a virtually unlimited set (e g. all components of a car) it would be more realistic to use a code based method or a machine learning approach.

So in hindsight, another approach would be to try some well performing libraries designed for AR-tags, like ARToolKit; or to use some feature based OpenCV technique for marker recognition. A way of placing out markers in a car without cluttering it with barcodes could be to paint them with near-infrared color and have some near-infrared light source in the car (in whatever wavelength that would suit the devices camera). Furthermore, with very robust markers, it may be possible to place out a reduced number of markers, and based on their position in the camera view coordinate system figure out what object is referred to, and then also make use of the information from the accelerometer (the tilt of the device). Those markers could also be used by different applications, for different purposes, specific AR-car-games, AR UI's and so on.

8 Acknowledgments

I am grateful to the people at BMW Connected DriveLab in Shanghai, to my supervisor Amen Hamdan, Senior Manager SW Development, and Philipp Holzer, Specialist in Human Computer Interaction, and to Alexis Trolin, Head of BMW Group ConnectedDrive Lab, for giving me the opportunity to do this internship at your very inspiring office.
I am also thankful to Associate Professor Anders Hast, topic reviewer, and Dr Björn Reinius for feedback on this thesis.
This project was supported or was made possible by a stipend from Bröderna Molanders stiftelse, and I would like say a big thanks to the carers of this foundation.

References

[1] GENIVI®, June 2012,
http://www.genivi.org/faq

[2] Wuelfing B. "CeBIT 2009: BMW and Partners Found GENIVI Open Source Platform", Linux Pro Magazine. March 2009
http://www.linuxpromagazine.com/Online/News/CeBIT-2009-BMW-and-Partners-Found-GENIVI-Open-Source-Platform

[3] Tholsgård G. *"3D rendering and interaction in an augmented reality"*, Bachelor thesis under preparation (December 2012), Uppsala University

[4] Ronald T. Azuma, *"A Survey of Augmented Reality"*, Presence: Teleoperators and Virtual Environments 6, 4 (August 1997), 355-385. Hughes Research Laboratories
http://www.cs.unc.edu/~azuma/ARpresence.pdf

[5] I Tuytelaars T. Mikolajczyk K. *Local Invariant Feature Detectors: A Survey*, Foundations and Trends in Computer Graphics and Vision Vol. 3, No. 3 (2007) 177–280

[6] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool *"SURF: Speeded Up Robust Features"* Computer Vision and Image Understanding (CVIU), Vol. 110, No. 3, pp. 346–359, (2008)
http://www.vision.ee.ethz.ch/~surf/eccv06.pdf

[7] Christopher Evans, *"Notes on the OpenSURF Library"*, CSTR-09-001, University of Bristol, January 2009
http://www.cs.bris.ac.uk/Publications/Papers/2000970.pdf

[8] Bradski G. Kaehler A., September 2008, *Learning OpenCV – Computer Vision with the OpenCV Library*, O'Reilly Media

[9] Lecture by M K. Warmuth, November 2011
http://www.youtube.com/watch?v=R3od76PZ08k&list=EC2A65507F7D725EFB&index=29&feature=plpp_video

[10] OpenCV Wiki, June 2012,
http://code.opencv.org

[11] OpenCV Documentation, June 2012,
http://opencv.willowgarage.com/documentation/cpp/objdetect_cascade_classification.html

[12] OpenCV Documentation, June 2012,
http://opencv.willowgarage.com/documentation/cpp/
clustering_and_search_in_multi-dimensional_spaces.html#
fast-approximate-nearest-neighbor-search

[13] OpenCV News, July 2012,
http://opencv.org/opencv-v2-4-2-released.html