

Image Recognition
Static Hand Gesture Recognition Software
Bachelor's Thesis
Mgr. Jan Kapoun

Thesis Supervisor: Ing. Václav Novák, PhD.

University of South Bohemia, České Budějovice

Faculty of Education

Department of Informatics

2010

Declaration

Hereby, I declare to have elaborated this Bachelor's thesis on my own, using only the resources and literature listed in the cited resources section.

I declare that according to the Act No. 111/1998 Coll., article No. 47b, as amended, I agree with the publication of this Bachelor's thesis in unabridged and electronical form in the publicly accessible section of the STAG database that is run by the University of South Bohemia on its web pages.

Furthermore, I agree with the text of my thesis being compared against the database of qualification theses, Theses.cz, that is run by the National Register of University Qualification Theses (Národní registr vysokoškolských kvalifikačních prací), and by the anti-plagiarism system.

České Budějovice, 1 December 2010

Author's signature:

Acknowledgement

I would like to thank Ing. Václav Novák, PhD for his support and advice.

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH
Pedagogická fakulta
Katedra informatiky
Akademický rok: 2008/2009

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Mgr. Jan KAPOUN**
Studijní program: **B1802 Aplikovaná informatika**
Studijní obor: **Výpočetní technika**

Název tématu: **Rozpoznávání obrazu**

Z á s a d y p r o v y p r a c o v á n í :

Počítačová technologie založená na automatické identifikaci obrazců a vzorů. Obraz se snímá vhodným periferním zařízením po jednotlivých polích, převádí se na digitální signál, který se následně zpracovává počítačem.

Úkolem diplomanta je napsání software pro rozpoznávání gest při snímání kamerou. Student vybere optimální algoritmus, implementuje ho do programu a provede srovnání s podobnými rozpoznávacími systémy. Provede porovnání jím naprogramovaného postupu a ostatních software. Nalezne metody hranice pro možnost použití. Zejména se zaměří na věrohodnost a osobitost gest jedinců.

Rozsah grafických prací:

Rozsah pracovní zprávy: 60

Forma zpracování bakalářské práce: tištěná

Seznam odborné literatury:

1. Microsoft [online]. Trvale aktualizováno. Microsoft, c1980-2009, 2009* [cit. 2009-04-10].
Dostupný z WWW: <<http://www.microsoft.cz>>.
2. PETZOLD, Charles. Windows Presentation Foudation: Aplikace = Kód + Markup. 2008. vyd. BRNO: Computer Press, a.s., 2008. 928 s., CD. Microsoft. ISBN 978-80-251-2141-2.
3. Současné metody rozpoznávání obrazů [online]. 2009, 2009 [cit. 2009-04-10].
Dostupný z WWW:
<http://www.odbornecasopisy.cz/index.php?id_document=33951>.

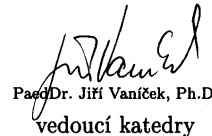
Vedoucí bakalářské práce: Ing. Václav Novák, CSc.
Katedra informatiky

Datum zadání bakalářské práce: 21. dubna 2009

Termín odevzdání bakalářské práce: 30. dubna 2010



doc. PhDr. Alena Hošpesová, Ph.D.
děkanka



PaedDr. Jiří Vaníček, Ph.D.
vedoucí katedry

V Českých Budějovicích dne 21. dubna 2009

Abstract

This thesis deals with recognition of static hand gestures in a real-time video feed. It is an accompanying paper to the application that I created. Within the framework of this thesis, you will find necessary algorithms for background segmentation and image recognition that were implemented in the application. This thesis also describes problems that I came across when writing the software and also some dead-ends I found when trying to choose the best algorithms. The application source code along with images and a video presentation can be downloaded here: [**http://jankapoun.own.cz/soft_AI.php**](http://jankapoun.own.cz/soft_AI.php)

Content

<u>IMAGE RECOGNITION.....</u>	<u>1</u>
<u>STATIC HAND GESTURE RECOGNITION SOFTWARE</u>	<u>1</u>
<u>BACHELOR'S THESIS</u>	<u>1</u>
<u>MGR. JAN KAPOUN</u>	<u>1</u>
<u>DECLARATION</u>	<u>2</u>
<u>ACKNOWLEDGEMENT</u>	<u>3</u>
<u>ABSTRACT.....</u>	<u>6</u>
<u>1. MOTIVATION & GOALS & TOOLS.....</u>	<u>9</u>
1.1 GOALS	9
1.2 SCOPE OF THE BACHELOR THESIS	10
1.3 PRE-REQUISITES.....	10
1.4 NECESSARY TOOLS & SOFTWARE	10
1.5 HARDWARE REQUIREMENTS.....	12
1.6 THE OPENCV LIBRARY	12
<u>2. RELATED BACKGROUND LITERATURE</u>	<u>13</u>
2.1 BRADSKI, GARY; KAHLER, ADRIAN: LEARNING OPENCV, COMPUTER VISION WITH THE OPENCV LIBRARY.....	13
2.2 CHEN, QING: EVALUATION OF OCR ALGORITHMS FOR IMAGES WITH DIFFERENT SPATIAL RESOLUTIONS AND NOISES	13
2.3 SUCHÝ, VÁCLAV: ROZPOZNÁVÁNÍ TEXTU V OBRAZE	13
<u>3. CURRENT STATUS OF IMAGE RECOGNITION</u>	<u>14</u>
3.1 MACHINE VISION	14
3.2 MILITARY APPLICATIONS AND SPACE RESEARCH	15

3.3 MEDICINE	16
3.4 DRIVERLESS CARS	16
3.5 VIDEO SURVEILLANCE.....	17
3.6 OPTICAL CHARACTER RECOGNITION (OCR)	18
<u>4. HOW TO RECOGNIZE?.....</u>	18
4.1 HOW TO MATHEMATICALLY REPRESENT AN OBJECT?	20
4.2 FEATURE EXTRACTION	21
4.2.1 FEATURE EXTRACTION BASED ON COLORS	21
4.2.2 FEATURE EXTRACTION BASED ON RAPID BOOST CLASSIFIERS	22
4.5 FEATURE EXTRACTION BASED ON BACKGROUND SEGMENTATION	23
4.6 CONTOUR FINDING	25
4.7 THE BOUNDING RECTANGLE	26
<u>5. THE STATIC HAND GESTURE RECOGNITION APPLICATION</u>	27
5.1 APPLICATION'S LAYOUT	27
5.2 APPLICATION'S CONTROLS	29
5.3 FEATURE RECOGNITION	30
5.4 APPLICATION'S WEAK POINTS	35
5.5 USABILITY OF THE APPLICATION AND SOME THOUGHTS ON THE FUTURE OF VISUAL RECOGNITION	37
<u>6. OTHER HAND-GESTURE RECOGNITION SYSTEMS</u>	39
6.1 HANDVU	39
6.2 PROJECT NATAL/KINECT.....	40
6.2.1 KINECT'S TECHNOLOGICAL BACKGROUND	40
6.2.2 KINECT'S HISTORY.....	42
6.3 SONY AND THE MOVE SYSTEM	42
<u>7. BENEFITS TO THE READER.....</u>	44
<u>8. CONCLUSION.....</u>	44
<u>LITERATURE:</u>	45

1. Motivation & Goals & Tools

My interest in this kind of software dates back to 2008 when I started looking for some information relevant to image recognition software. Soon enough, I found a couple of papers dealing with OCR (Optical Character Recognition). There was one PhD thesis made by a student from Ontario, Canada, and, most importantly, another bachelor thesis written by a VUT (technical university) student from Brno that also dealt with OCR. His work used the OpenCV library quite extensively. OpenCV library constitutes quite a handy way of how to create image recognition software. This student's work attracted my interest quite a lot and, subsequently, I decided to attempt to write such a piece of software myself, however, as OCR software is commercially used quite extensively, I wanted to create something a little bit different. This is why I finally decided to try to write a real-time static hand gesture recognition system.

1.1 Goals

The goal of this thesis is to show a way of how such a piece of software can be created. The applications, such as Microsoft Kinect or Sony Move, are commercial ones. As such, their source code is closed so that there is no way of how we could look into it (except for reverse-engineering) and see how it had been created. My idea is to start almost from the scratch. Using the OpenCV library I am going to attempt to write such software. Of course, I do not intend to try to compete with commercial applications as they have been written by teams of professionals with a PhD and years of experience in the field. What I am trying to do, is to show a way of how it could be done and to share my experience. Over a couple of months time, I have been doing some personal research in this field. This thesis is going to be something like a diary of my work. After the reader has read this paper, he/she should be able to catch

some basic ideas and concepts that are required for such a work and understand, at least basically, how such software works.

1.2 Scope of the Bachelor Thesis

This thesis deals with a visual recognition system. It basically describes the process of creating such an application from the scratch, using the OpenCV library. The methods, algorithms, necessary tools, and software are all described here. This thesis will explain the basic algorithms and the way how to put them together into a sensible piece of software.

1.3 Pre-requisites

In order to understand this thesis, you need to know at least the basics of the C++ programming language and you should also have some background in visual recognition techniques. Most notably, you should have at least some overall knowledge of the OpenCV library.

1.4 Necessary Tools & Software

In order to create this software, several tools and software applications were used. Primarily, it was the OpenCV 2.0 library which I used in connection with the Visual Studio 2008. As I found out, OpenCV exists for other languages as well, notably there are wrappers for C#, JAVA, and even Python, but I decided to use the original C++ library. The reason was that all the subroutines I needed for my software were in C++ as well so why bother with conversion into other languages, although it needs to be said that programming per se would most probably be easier in C# or JAVA. However, the Learning OpenCV book and all the subroutines are written in C++ so I took it as my language of choice.

This initialization of the OpenCV library in the Visual Studio 2008 took a while and was quite frustrating as the official tutorial describing how to set the VS 2008 in order to work with OpenCV contains several mistakes. As soon as you succeed to initialize the library, it works quite reliably.

1.4.1 Setting Up the OpenCV

This is the procedure that works for OpenCV 2.0. But it can be different for newer versions...

- Download and install OpenCV2.00 (Run the OpenCV2.00 installer and follow the instructions).
- Download and install the CMAKE v. 2.8. Run the Cmake (cmake-gui) file.
- In the graphical Cmake environment select „Source“ – select the folder OpenCV2.00. Choose some other folder for „Browse build“, e.g. OpenCV2.00Release (Caution!! do not enter the same folder – OpenCV2.00, otherwise you will get issues with compilation!)
- In Cmake, keep clicking Configure and then Generate. Then, close the Cmake.
- In the OpenCV2.00 folder, look up the .sln file. Open in Visual Studio 2008. Press F7 (build).
- Close the project and create a new project (e.g. „test“)

1.4.2 Setting Up the Visual Studio

- In Tools, Options – VC++ Directories, Include files, insert „C:\OpenCV2.0\include\opencv“
- In Library files – insert „C:\OpenCV2.0Release\lib\Debug“ (This must reference to the OpenCV folder compiled by Cmake)
- In Source files – insert „C:\OpenCV2.0\src\cvaux“, „C:\OpenCV2.0\src\cxcore“, „C:\OpenCV2.0\src\cv“

- In Project, Properties – Linker, Input, Additional Dependencies, insert „cv200d.lib cxcore200d.lib highgui200d.lib“
- From the „OpenCV200Release/bin/Debug“ folder (created by Cmake) copy all .dll files into the folder with the actual .exe application that you want to create.

1.5 Hardware Requirements

In order for this software to work, you can use hardware that meets normal hardware requirements as of 2010. No super special hardware is required, the application runs on normal hardware pretty much nicely. However, there is one thing that needs to be pointed out – the camera. This is really very important as the camera must deliver a stable signal without too much change in luminosity (lightness). This caused lots of problems in my Lenovo notebook with an integrated camera that changes the signal luminosity quite randomly and, in consequence, causes the software to fail as the signal stability is very much important in order for the application to work correctly.

1.6 The OpenCV Library

This is the library without which the creation of the whole application would be impossible, or, better to say, quite difficult and laborious. Here is a short description as found in Wikipedia:

“OpenCV is a computer vision library originally developed by Intel. It is free for use under the open source BSD license. The library is cross-platform. It focuses mainly on real-time image processing. If the library finds Intel's Integrated Performance Primitives on the system, it will use these commercial optimized routines to accelerate itself.”[6]

The OpenCV contains all the algorithms I used for this application, notably then algorithms working with background subtraction, Hu moments, contour finding, and camera input.

2. Related background literature

This section describes the most relevant background literature that I used in my thesis.

2.1 Bradski, Gary; Kahler, Adrian: Learning OpenCV, Computer Vision with the OpenCV Library

This is a textbook on the OpenCV library. It contains description of this library and its practical usage, as well as it is a perfect introduction into the visual recognition problems and solutions. Especially, the parts describing camera input, image segmentation using the average and “codebook” methods, contour finding, and recognition based on Hu moments were of particular interest to me.

2.2 Chen, Qing: Evaluation of OCR Algorithms for Images with Different Spatial Resolutions and Noises

This is a PhD thesis that caught my attention when looking for some relevant materials. Within the framework of this thesis, Chen wrote a couple of OCR algorithms and tested them under various conditions. He tested algorithms based on e.g. chain codes, Hu seven moment’s invariants, complex moments, etc.

2.3 Suchý, Václav: Rozpoznávání textu v obraze

This is a Bachelor’s thesis dealing with the OCR issues. Written by a VUT student (Technical University of Brno), this thesis was of particular interest to me, as Suchý used the OpenCV library and a number of algorithms that were relevant for my work. Unlike me, he used an artificial network (FANN) in addition to OpenCV in order for the individual characters to be recognized. However, he reported that his application was not particularly successful at recognizing individual characters.

3. Current status of image recognition

Image recognition is a recent and fast expanding field of science. There are also significant commercial applications like face recognition software used for notebook security (log on) or the Microsoft's new home gaming system (project Natal/Kinect) that scans user's movements and motion in order to react accordingly and thus provide better gaming experience. There are naturally also applications used in video surveillance security, be it the surveillance of objects against intruders or surveillance in casinos that look for suspicious guests' behavior, or the surveillance software used in subways as a pre-caution against possible terrorist attacks. In the following sections, you will find several fields where image recognition is used quite successfully.

3.1 Machine vision

Machine vision is a highly practical subfield of the computer vision. It is often used in production factories such as automobile manufacturing facilities, computer chip manufacturing etc. or even agriculture. As such, the machine vision has usually very straightforward range of tasks. Machine vision software can guide the robots that assemble the cars and look up locations to weld or it can guide a robot's arm in the painting shop. Another scope of application is e.g. the defect detection. In such a facility, there is usually an assembly line with products moving on it. Then, there is the location where the actual defect recognition takes place – usually there is a camera, some computing hardware and an actuator that ejects a defective part out of the assembly line. Recently there have even been specialized “smart cameras” with built-in computer hardware and machine vision software. Such a solution is usually cheaper than a classical pair of a camera and a computer and thus may be preferred by factory managers. Cameras are usually just black and white, although the use of color cameras has been taking off recently. Such a system has, sure enough, its advantages: it never becomes tired or distracted and is capable of providing its

service 24 hours per day thus becoming cheaper than a regular human worker. On the other hand, it has its disadvantages too: it is still not as good as a human worker when it comes to visual inspection under difficult conditions (e.g. not enough light). Human workers are still better at performing under non-standard situations, because they can make assumptions or other intelligent actions which a machine is still incapable of doing. See related information in [8].

3.2 Military applications and space research

To a large extent, computer vision is used in military as well. One of the examples can be found in [9]. In this paper, a group of university graduates designed an autonomous system that can track moving airplanes or missiles in the sky and shoot them down with a revolving turret. To construct this system, they used commercial components, such as a standard PC with the Windows operating system and a USB web cam. Their system is developed in the MS Visual Studio .Net and is fully autonomous, i.e. it can detect, track, and shoot down suspicious objects. Another example would be the autonomous system for missile guidance. Traditionally, a missile is shot off onto an actual target. With autonomously guided missiles, the missiles are shot into a particular area. Equipped with an onboard camera and a computer, they are able to autonomously decide on a target to be attacked. See related article in [10]. There are also autonomously guided vehicles, unmanned aerial vehicles (UAV) or even submersibles. The value for military is obvious: machines can be used instead of actual soldiers, thus saving human lives. Lockheed Martin is one of the military companies using computer vision along with OpenCV in their devices. Computer vision algorithms find their successful application in space research as well. Equipped with two stereo cameras, the Mars Rover was successfully exploring the planet of Mars.

3.3 Medicine

Medicine is another field where image recognition is widely used. There are a number of methods when images are made and retrieved from devices, e.g. magnetic resonance imaging, ultrasound, tomography, just to name a few. In large hospitals, lots of such images can be created on a daily basis. Subsequently, such images need to be examined by a physician who can decide whether there is anything wrong or whether the patient is healthy. This is where image recognition comes in - it can substantially save the physicians the laborious work when inspecting such large number of images. There are special algorithms which are capable of determining whether there is e.g. a tumor in a particular image. Images automatically identified as “healthy” can be omitted and only relevant pictures can be send to the doctor, thus saving a lot of work and time. Of course, algorithms must be programmed in such a way that there is a low number of false alarms and possibly a zero occurrence of false negatives (images with a tumor in them that would be identified as “healthy”).

3.4 Driverless cars

Driverless cars would be a great application of the computer vision. A driverless car usually comprises of an ordinary car with special equipment. It can be just one or a pair of cameras and a computer (and of course some servo equipment needed for steering, clutch, brake and gas pedals) or there can be some additional devices such as GPS navigation module, radar or laser equipment that is used to acquire as much data from the surrounding environment as possible. From the computer vision’s point of view, the camera, computer and some CV software are of interest. Why are driverless cars so promising? They would eliminate the human factor. Humans are prone to commit mistakes – they can drive too fast or under the influence of alcohol and drugs, which would never happen with computer driven cars. Also, instead of

driving, people can engage in other activities, such as work with a notebook or conversation with a partner, thus being more productive.

There are a couple of projects that tackle the driverless car issues with the most known being the US DARPA Grand Challenge project, Italian ARGO, Euro EUREKA Prometheus Project, or the Carnegie Mellon University team's driverless car project. In all of them, scientists attempted to build a fully automated car capable of independent driving in heavy traffic with results being quite promising: The Carnegie team's car was able to drive a 5000 km journey with 98.2% of it being autonomously driven by a computer! However, in spite of all these advances, driverless cars still remain a promising field of research – they are still not good enough to be put into market for commercial use. Also, there are issues unrelated to technology: A) people are afraid of trusting a computer to drive their car safely and B) there are legal issues – who should be liable in case an accident happens? Should it be the car's owner or the manufacturer? Related information on driverless cars can be found in [11].

3.5 Video Surveillance

Video surveillance is going to play a major role in public security in future. What is actually video surveillance? Traditionally, video surveillance comprises of a camera or a set of cameras connected to an industrial TV with human personnel watching the TV screen and analyzing the scene. However, with the onset of computer vision, the situation changed. Nowadays, smart surveillance systems are used. They also consist of a set of cameras, however, they are connected to a computer running a special surveillance software. What is the software capable of doing? It grabs the images sent by a single or by a set of cameras, then it analyses the scene, i.e. subtracts the static background (buildings or other static objects in the background) and focuses the objects of interest. These can be humans or cars or some other objects. Modern surveillance systems are even capable of analyzing human behavior. As you can find in [12], they created an application that tracks humans in a given

scene, subtracts their silhouettes from those of cars and then analyses whether the humans are moving at a constant speed past the cars (meaning that the humans do not pay too much attention to the cars, which is a normal, non-suspicious behavior) or whether they stop at the cars (suspicious behavior). In their work on surveillance, they also mention the ethical aspects of such systems: In public areas, people would be constantly monitored by intelligent systems capable of analyzing their actions. Thus, people would lose a great deal of their privacy. On the other hand, such intelligent surveillance systems would probably contribute to lowering the crime rate. But that's the problem of modern democratic societies: should people be given more privacy and freedom with the crime rate being relatively high or should the privacy and freedom be restricted with safety and security made better?

3.6 Optical Character Recognition (OCR)

OCR is one of the most widely used image recognition applications. First algorithms and machines intended to read the printed or handwritten text were invented already in 1929 by Gustav Tauschek and later by Handel. Today the problems related to character recognition are largely considered as solved and accuracy rates reach above 99%. However, for Arabic or Asian scripts, there are still issues to be dealt with. Nowadays, OCR software constitutes standard equipment of most scanner devices sold at the market. More information can be found in [13].

4. How to Recognize?

In the beginning, I had absolutely no idea of how anything like recognition can be performed and achieved. Soon, I discovered that in order for a computer to recognize something in the screen, we have to somehow describe the object that we want to be recognized. It needs to be described mathematically. Yet, this was something I just could not put my finger on. Thus, I started some research,

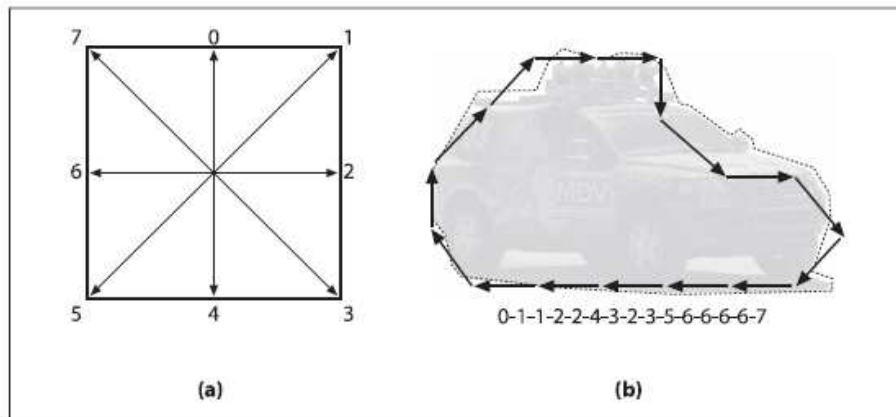
mostly in the internet and partly in our university/national library. I discovered some books, all of which were written for people who already knew quite a lot about the problems related to computer visual recognition. The books were full of mathematical formulas but I needed some rudimentary guidance, some basic ideas on which I could build my software. So I started to search a little bit more specifically – I wanted to know something more concrete about the OCR. How is the OCR actually performed? What are the basic principles? Unfortunately, most of the material I found over the Internet was hopelessly general describing general facts, like e.g. “Scan the page and perform a topological analysis of individual characters.” But what is exactly a “topological analysis” and how to perform it? Luckily, I soon discovered the Suchy’s thesis [4] which deals with a problem of how to create an OCR application. In his thesis, Suchy created a working application, simple as it might be, which, however, used principles that were of major importance to me. Actually, the principles that lie behind the basic recognition are quite simple, once you know them. Also, he used the OpenCV library, which I had no previous experience with and which proved to be extremely important as well. Suchy used two major parts in his thesis: the OpenCV and the FANN (neural network). I was considering for quite some time to use in my own application and thesis these two components, too, but then I realized that a neural network will actually not be needed at all. So how does a visual recognition actually work? It is quite simple once you know how to do it. The steps are as follows:

- scan the image of interest (e.g. a letter)
- subtract the background (i.e. everything we are not interested in) – several methods can be used for this task as described later in this thesis.
- find and store contours of the given objects – several methods can be used for this.

- Represent the contours in a given mathematical way. Suchy used the Freeman chain code. I chose to have contours represented as a sequence of vertices from which I later computed the Hu moments, which seem to deliver quite reliable results.
- Compare the simplified mathematical representation to a pattern stored in computer's memory. A neural network can be used for comparison purposes if there are many values and relationships between them that would be difficult to analyze otherwise.

4.1 How to mathematically represent an object?

In order for an object to be recognized, we need to represent it mathematically. How to do that? Luckily, in my case, we only need the object's contours which we can then turn into a row of numbers that can be subsequently stored in memory or further computed with. For this purpose, we can use the Freeman chain code. A definition of a Freeman chain code [1]: *“With a Freeman chain, a polygon is represented as a sequence of steps in one of eight directions; each step is designated by an integer from 0 to 7.”*. In the following picture, you can see what Freeman chain code is all about (adopted from [1]):



This is a relatively simple method which Suchy used in his work. However, another method seemed to fit my purposes better – the Hu moments. They will be explained later in this thesis.

4.2 Feature Extraction

Feature extraction constitutes the basic and fundamental part of the software application that we are tackling here in this thesis. What does that actually mean? It means that in order for a particular hand gesture to be recognized, we need to find out where in the image is the hand actually located. That might seem as an easy task for a human but it is not that easy to program it into a computer.

So how can a computer actually distinguish between the background and the hand? How can a computer find a contour of a hand in a particular image? To resolve this problem, I was considering several approaches and, in course of the time, I think that I found the right and appropriate solution. But let us take a look at the ideas that I was playing with:

4.2.1 Feature extraction based on colors

The very first idea that came to my mind was to extract the hand features out of the image just by colors. Look at your hands – they are pinkish – so we could simplify things down and say that everything pinkish in a given image is the hand. This approach, however, brings several problems:

- In addition to the hand, there is also the face in the image, which is pinkish, too. So how do we tell what is a hand and what is a face? Yes, we could use some face recognition routines, find the face and take it out from the image. But that seemed to be rather a complicated and clumsy procedure and, besides, face recognition would eat some computing resources. There is also another problem related to colors – try recording an image of a hand in various light conditions. The hand will have quite different a color when recorded outdoors and then indoors. In dark conditions, the hand might almost be black; on the contrary, when there is plenty of light, the hand might be almost white.

- There are problems with computer cameras. I was using just a cheap webcam integrated into my Lenovo notebook. Despite my liking and respect for this brand, the built-in camera really “sucks” – the signal is very noisy and, moreover, it changes the lightness of a recorded image quite randomly. This is not a problem if you use the camera for your Skype video-telephony only, but when you decide to use it for something more serious, like an application that requires quite stable input signal, significant problems occur. I spent some 2 weeks trying to find a bug in my application and then I discovered slight changes in input image’s lightness signal. When launched on another computer with another webcam, the problems disappeared, at least partly. Why is the changing image lightness actually a problem? You will find out soon when we talk about the segmentation methods used in this software.

All of these factors made me leave and discard this idea with colors completely and I started looking for something that could yield better results.

4.2.2 Feature extraction based on rapid boost classifiers

This approach was used in a subpart of OpenCV, the face recognition. See [1] for more information. There is an already trained face recognition classifier included in this library that works fairly well. The only problem is that it is not too well documented in the OpenCV, so it’s problematic to train one’s own classifier. Finally, I found a paper in the internet (see [14]), documenting how such a classifier could be created. Basically, it works as follows:

- Take some 10,000 images and divide them into two groups.
- Let’s say group 1 will be 7,000 positive images (images where the searched-for feature is present),
- group 2 will be 2,000 negative images (where the searched-for feature is not present) and

- group 3 will be 1,000 images for testing purposes (some are positive some not, the software is expected to find the positive images).
- With these images prepared, you can train your classifier similarly to how you train a neural network.

This sounds easy in theory but going through the original document describing how to train the document in practice, I found it very complicated – something I would lose a lot of time for, with the result being uncertain. Moreover, the author of that paper reported that his classifier did not perform well. This is why I discarded this idea completely.

4.5 Feature extraction based on background segmentation

I finally turned my attention to an approach that uses the background segmentation. The idea is quite simple: first, let's have the computer learn the static background and then subtract it from the live video feed. Like this, things get simplified down quite extensively because we instantly know where to look for a hand – the hand is a new, moving object so it is immediately apparent.

There is actually only one serious problem: the noise in the image. Why is it a problem? The computer learns a static background – all the objects in the image that do not move. There might be a wall, a tree, a window, anything. So the computer actually remembers all the pixels in the image. All these pixels are supposed to be static – they should stay without any change over the time. If there is any noise, and there always is some, this becomes a serious problem because the pixels and their colors change at random. Generally, human eyes are capable of ignoring this noise (to some extent) but computers work with exact values, so the noise disrupts a background-subtraction algorithm quite a lot.

Luckily, I found a way how to deal with this. We can have the computer learn the static background for a period of several seconds and let it acquire several tens of images, which we can afterwards:

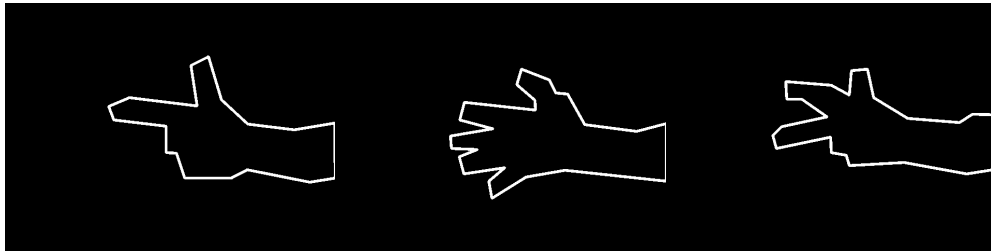
- make average – like this, we get rid of the noise quite effectively because every individual pixel will have its color set quite exactly. For example, if there is a pixel that is normally black and you get 9 black readings from the camera and 1 red reading (noise) and then you average these results, you get something that is very close to black. With some rounding off, you get black. So, the noise is eliminated. Subsequently, we save this information. This method, however, proved ineffective because it still left quite a lot of noise in the resulting picture. I abandoned this algorithm completely as it did not suit my needs.
- analyze and learn the extent(s) within which a particular pixel actually changes. For example, let us assume that the RGB value of a given pixel ranges between 128,23,30 – 128,40,100. We can remember this range for future reference and label it as “normal” for a given pixel. Then, in real video feed, we can ignore all those pixels that have their value within this range because that means that nothing new happens in the image. If, on the other hand, a new object appears in the image, the pixel’s value exceeds the given range and we can consequently extract these pixels out of the image. This is actually the idea behind the method of “codebooks” – learn the extent within which a particular pixel oscillates. There can be a number of extents within which a pixel can oscillate. For instance, we can have the 3 intervals for, let’s say the B (blue) segment of a color, for a given pixel – “0-10”, “55-83”, and “230-245”. These are the intervals we have learnt from the background. Subsequently, we compare every pixel from the live video feed and in case its values fall within one of these intervals, we can say that this particular pixel represents the static background. If, on the other hand,

the values do not fit into one of these intervals, we can evaluate the pixel as representing a new object that just entered the scene.

Additional information on average and codebooks methods can be found in [1].

4.6 Contour finding

Contour finding is a very important activity within the scope of this application. Let's start with a definition of what a contour is and what such a contour is actually good for. A definition found in [1]: *“A contour is a list of points that represent, in one way or another, a curve in an image. This representation can be different depending on the circumstance at hand“*. The following images represent a contour of real video images as made by the OpenCV's functions `cvFindContours()` and `cvDrawContours()`:



What do I need these contours for? Within the scope of this application, contours are very important – they represent a simplified version of a real-video image that can be compared against some pattern stored in memory. The solution used in my application is a slight variation: first, I created contours of a hand during a live-video feed and, in the real-time, computed the Hu moments of these contours. Subsequently, I stored these Hu moments in memory for future reference and comparison.

For the simplicity's sake, I also had the contours polygon-approximated, i.e. I used the `cvApproxPoly()` to simplify a given contour in terms of this contour having fewer vertices. The contour looks somewhat smoother and more elegant, however, as I empirically found out, the approximation has virtually no effect

on the recognition process itself, although the recognition process might run faster with contours approximated.

4.7 The Bounding Rectangle

Looking at the graphical representation of the application shown in a window in the screen, you immediately spot the bounding rectangle in the left part of the screen. What is it good for and why did I use it? Well, it is pure problem simplification on my part. During the development of the application, I soon realized that it would be so much easier to recognize a hand gesture if we knew, at least approximately, the location of the hand. Although the background subtraction method works fairly well, there are also other moving objects in the image that could be identified as the image forefront and thus not be removed from the final image to be recognized – the face and the upper body part. Of course, one could have the computer learn the Hu moments of the contours of the face and the upper body and then ignore them in the final image, yet, the problem is somewhat more complicated: if you try to scan a complete image, subtract the background, and then find a contour of the foreground object, you realize that you get one big contour describing the head, the upper body part and the hands. This is not useful at all – how do you recognize where is the hand and what is the head? How do you split the contour? Well, we could take this big contour and compute the Hu moments from this, but in this case, the hand itself would be just a small portion of the total contour and would therefore have only little effect on the final Hu moment value, let alone the fingers which are of extreme importance for recognition of a particular gesture but get literally lost in the big contour – they are just too small. This problem could probably be solved in a satisfying manner if I had the time to tackle it more thoroughly, however, this was not my goal – I wanted to devise a quick and dirty application that would perform the actual recognition; I was more interested in the general principles. So, this is why I used the bounding rectangle, which has a simple purpose indeed: place your hand into the

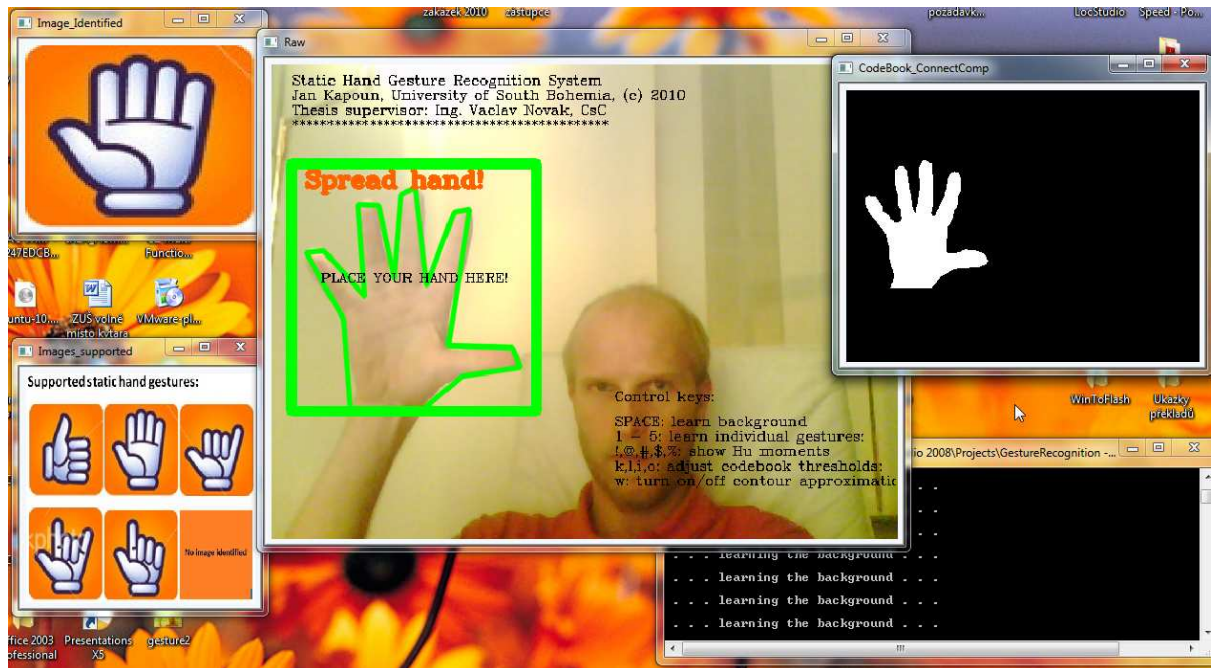
rectangle and make sure no other objects get in the way. The system identifies a new object and tries to find its contours. Subsequently, Hu moments are computed and compared against values stored in memory. If a particular gesture is recognized – bingo! – if not, the object is ignored. This is a huge simplification because we always know that if something moves in the rectangle, it's most probably a hand and we can immediately compute its characteristics. Also, we get a contour in the required detail where fingers have quite a dominant influence in the computation of Hu moments and the subsequent recognition.

5. The Static Hand Gesture Recognition Application

The application can be downloaded here: http://jankapoun.own.cz/soft_AI.php. MS Visual Studio 2008 must be installed on your system in order for the application to be launched successfully.

5.1 Application's Layout

After the application is launched, you will see a couple of windows:



Application's layout

There are two windows on the left:

The upper window shows a pictogram of the particular gesture identified by the recognition routines. If a given gesture is recognized, gesture's description and its pictogram will appear in that window.

The lower window shows the gestures that this application can recognize. There are five gestures that can be identified: thumb up, spread hand (hand with its fingers spread), phone (hand with the thumb and little finger raised), devil (hand with the thumb, little finger and index finger raised), scissors (hand with the thumb, index finger and middle finger raised). Other gestures might be defined per request, however, gestures' description is hard-coded and will therefore not match the visual representation in case you change it.

The largest window is actually the most important one. It is the main application's window. There, you will see the live video feed as recorded by the camera. A red bounding rectangle appears in the left part of the window. It's function is further explained later in this thesis. If you place your hand into this

bounding rectangle, the software will try to recognize the gesture being shown. In case a particular gesture is recognized, the rectangle will turn green and a red description will appear right next to the gesture.

The other window is as large as the application's window, however, is hidden behind it. This is the window where the subtraction algorithm is shown in black and white. As you can observe, only new objects (foreground) are shown there. The last window is the console. Here, you can see the output of certain algorithms as well as the output of several commands. When interacting with the keyboard, you should always keep the focus on the main application's window (video feed), not on the console, because the keystrokes will not be read here.

5.2 Application's Controls

In order to control the application, there are several keys you can press and control the application in the desired way.

Here comes the list:

Space bar – re-learns the background. Use this option if you want to let the computer learn the background again. You might want to take advantage of this function after you reposition your camera, because this action disrupts the learned background structures stored in the computer's memory. After you press this key, step away from the camera view, otherwise the background will not be correctly learned. The learning takes a few seconds during which the application freezes.

'w' – turn on/off the contour approximation. If you turn this option on, the contours will have fewer vertices and will appear somewhat better shaped. If you turn it off, contours will be ragged. This has little effect on the recognition capability, however. It's more or less a cosmetic issue.

1 to 5 – press these keys if you want to let the computer learn new gestures. For instance, press key '1' in order to let the computer learn a new version of the 'thumb-up' gesture (the thumb-up gesture is the hard-coded gesture assigned to

the key '1'). Place the thumb-up gesture into the bounding rectangle and press '1'. The application will scan some 100 frames during which time you should move your hand inside the rectangle a little in order for the computer to learn possible deviations in the way how this particular gesture is scanned and recognized by the system. Numbers (Hu moments) will appear in the console. After this procedure (which takes a couple of seconds) is finished, you can start using this new gesture. Of course, you can also show some other gesture (e.g. the 'victory' sign) and the system will learn it, but the descriptions are hard-coded and therefore will not match your gesture. Of course, the application could be easily adjusted to give the user the possibility to enter new description, but this is an experimental piece of software and user-friendliness was not the concern here.

Shift + 1 to 5

5.3 Feature Recognition

Along with background segmentation, this is the most important part of the software. Originally, I was inspired by the Suchy's bachelor thesis which deals with optical character recognition – OCR. Suchy uses the Hu moments together with a neural network – FANN (Fast Artificial Neural Network).

Let's make a little digress and explain what the FANN and the Hu moments actually are all about.

FANN – this is an open neural network which is free to download and easy to use. It can be found under the following address: <http://leenissen.dk/fann/html/files/fann-h.html>. Suchy used it in his software to learn the Hu moments values of all 26 alphabet characters. He describes the process of using the network and the difficulties he had when setting it up. In my experience, the FANN is really quite easy to use and implement into one own's code. There are also several settings which allow you to tune the network up according to your needs. Well, the examples that are part of the FANN package worked quite fine but as soon as I attempted to make it work according

to my needs, the network did not yield expected results. Now, I believe the number of input, intermediate, and output neurons needs to be adjusted in order for the network to work right. However, I did not do that and I left the idea of the neural network completely. Why? Because I found out that for my purposes the neural network will most probably be not needed as there are only 5 gestures/contours that need to be recognized. I will tell more about my reasons for abandoning the network as soon as I explain the Hu moments.

First of all, a definition of what a moment actually is: *“Loosely speaking, a moment is a gross characteristic of the contour computed by integrating (or summing, if you like) over all of the pixels of the contour. In general, we define the (p, q) moment of a contour as*

$$m_{p,q} = \sum_{i=1}^n I(x,y)x^p y^q$$

Here p is the x-order and q is the y-order, whereby order means the power to which the

corresponding component is taken in the sum just displayed. The summation is over all of the pixels of the contour boundary (denoted by n in the equation). It then follows immediately that if p and q are both equal to 0, then the m00 moment is actually just the length in pixels of the contour.” [1]

A small example to demonstrate the Hu moments: Consider the following graphical representation of several letters: (adapted from [1])



Here, we have a table with Hu moments values of individual letters as returned by one of the OpenCV functions (adapted from [1]):

	h_1	h_2	h_3	h_4	h_5	h_6	h_7
A	2.837e-1	1.961e-3	1.484e-2	2.265e-4	-4.152e-7	1.003e-5	-7.941e-9
I	4.578e-1	1.820e-1	0.000	0.000	0.000	0.000	0.000
O	3.791e-1	2.623e-4	4.501e-7	5.858e-7	1.529e-13	7.775e-9	-2.591e-13
M	2.465e-1	4.775e-4	7.263e-5	2.617e-6	-3.607e-11	-5.718e-8	-7.218e-24
F	3.186e-1	2.914e-2	9.397e-3	8.221e-4	3.872e-8	2.019e-5	2.285e-6

Hu moments – they are a powerful tool for recognition purposes. They consist of 7 values which range between 1 and 0. The higher is the Hu moment, the smaller the number. So, if you have a contour of a given object, you can pass this contour to a subroutine that computes the Hu moments and you get 7 values returned. Then, it's a piece of cake, you can compare these values with some pattern that you have stored in memory and decide what the contour actually is.

Of course, it's not that easy as the contours never exactly match the pattern you have stored in memory. This is why Suchy used the neural network, as he had to find and compare 26 approximate values (each for one character). However, in my case, the problem is pretty much simplified – the software recognizes 5 gestures only. Accordingly, I decided that a neural network was actually not necessary and discarded this idea completely. Instead, individual gestures were empirically tested as to what Hu moments they yield. For that reason, a simple routine was implemented into the code and run several times in order to find out individual Hu moments for each gesture. This, surprisingly, works pretty well. The routine looks as follows:

In the first part, we initialize the individual gestures with empirically found values:

```
void InitGestureMoments()
{
    //empirically found values for various gestures
```



```
thumb.L1 = 0.172611;  
thumb.H1 = 0.210969;  
thumb.L2 = 0.00155066;  
thumb.H2 = 0.00358486;  
thumb.L3 = 0.00079;  
thumb.H3 = 0.00234453;
```

```
gun.L1 = 0.203805;  
gun.H1 = 0.249095;  
gun.L2 = 0.00531133;  
gun.H2 = 0.015;  
gun.L3 = 0.0008;  
gun.H3 = 0.0055;
```

```
...
```

```
}
```

As you can see, here we assigned the low (L1-L3) and high (H1-H3) Hu values. These values were empirically found when experimenting with the software. I just simple showed a gesture and let the software print the values into the console.

Basically, I let the software read some 100 Hu moments values of a particular gesture, then made an arithmetic average and took 120% as the high value and some 80% as the low value. Why are high and low values needed? Because if you show a particular gesture, the Hu moments will vary with each frame slightly (the hand can be turned slightly clock-wise or anticlock-wise, it can be in a certain distance – all of this influences the actual Hu moments, hence some range (high and low) is needed.

The relevant piece of code that calculates the average and percentage looks as follows (“loops“ stands for the number of learning cycles):

```
/**
```

```

* This generally learns a gesture
*/
void auxLearnGesture(GestureData * gd, bool * pLearning)
{
    auxInitHuMoments();

    if (gd->counter == 0)
    {
        gd->H1 = gd->sumH1/loops/100*110;
        gd->L1 = gd->sumH1/loops/100*90;
        gd->H2 = gd->sumH2/loops/100*140;
        gd->L2 = gd->sumH2/loops/100*60;
        gd->H3 = gd->sumH3/loops/100*150;
        gd->L3 = gd->sumH3/loops/100*50;
        gd->H4 = gd->sumH4/loops/100*200;
        gd->L4 = gd->sumH4/loops/100*10;
        gd->H5 = gd->sumH5/loops/100*250;
        gd->L5 = gd->sumH5/loops/100*5;

        ...
    }
}

```

Once you have the appropriate Hu moments values, it is very easy to compare them with a live-feed values. It is just a sequence of if-else:

```

...
if ((huMom.hu1 >= thumb.L1) && (huMom.hu1 <= thumb.H1) &&
    (huMom.hu2 > thumb.L2) && (huMom.hu2 < thumb.H2) &&
    (huMom.hu3 > thumb.L3) && (huMom.hu3 < thumb.H3))
{
    cvShowImage("Image_Identified",img_id1);
    match = true;
    auxPutText(rawImage, "Thumb!");
}
...

```

(Here, we identified the „Thumb“gesture.)

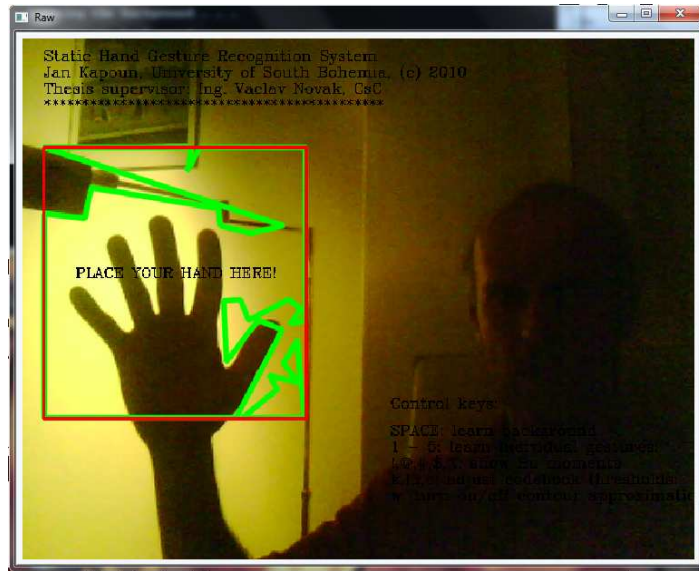
Basically, the procedure with Hu moments can be summarized as follows:

1. Run the application, show a particular gesture and press the 1 – 5 key (to let the software learn the gesture).
2. The application will scan your gesture for some 100 times (this can be adjusted).
3. Hu moments for each frame and gesture will be calculated.
4. An arithmetic average of Hu moments will be calculated.
5. Hi and Low values will be determined based on the arithmetic average and, subsequently, will be stored in memory.
6. The calculated Hi and Low Hu moments values will be used for future reference (live video feed and recognition).

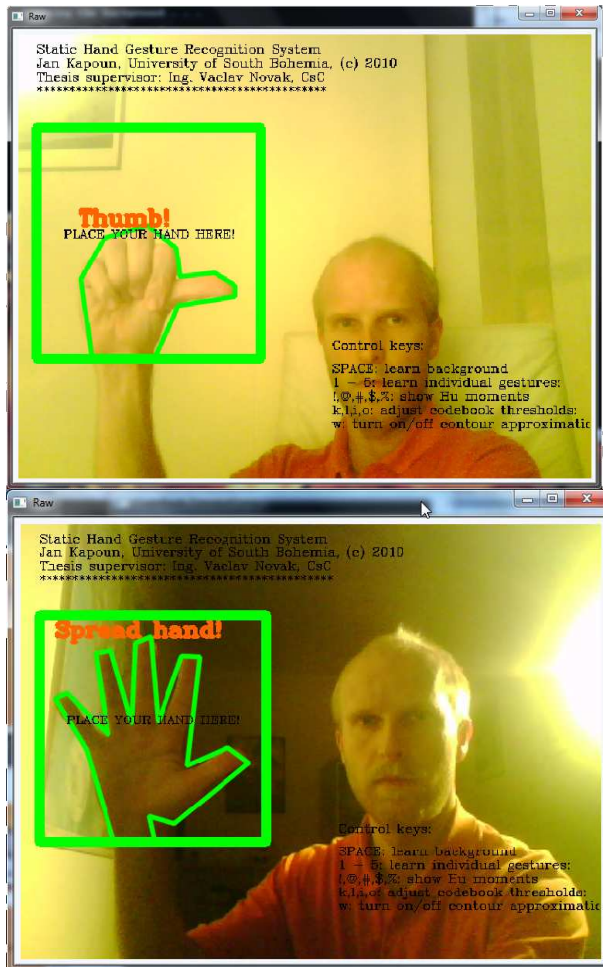
5.4 Application's weak points

The major application's weak point is the sensitivity to lighting conditions. The application works best when being run under moderate light conditions, e.g. when the environment is lit just by a small 20-40W lamp. It must not blind the camera and your hand must not get between the source of light and the webcam because the hand would cast shadows which would negatively influence the recognition process. It doesn't work well under natural light conditions because they tend to change (clouds that cast shadows and constantly change the light intensity). To sum up, the application works best under moderate and stable lighting that illumines the webcam from the side (no shadows cast by your hand) and does not blind the webcam. Why is it so? It is related to the background subtraction algorithm which basically subtracts the constant background from other moving objects (hand). When the lighting conditions change, the system suddenly does not know how to handle the individual pictures, because their RGB values can change dramatically and thus the system does not know if they still represent the constant background or if they

represent a new object that entered the scene. In the current version of this application, the system always assumes that such pictures represent a new objects and the whole algorithm fails to recognize anything whatsoever.



Poor performance – due to poor lighting, contours are misinterpreted (the hand gets into the light's way, changing the amount of light that strikes the camera which influences the application in a negative way)



Good lighting, good performance (camera is well-lit, the hand does not get into the light's way)

5.5 Usability of the application and some thoughts on the future of visual recognition

5.5.1 Practical usability of the application

Could the application I created within the framework of this bachelor thesis be used in real life? I hardly think so. The practical application's usability in real life is practically zero, because the application has been created as an exercise in visual recognition techniques and hence its primary purpose is to study this phenomenon. However, the application could probably be adjusted to serve

some practical purpose. I was originally thinking about turning the application into some sort of a video player that could play a movie and, at the same time, detect user's hand movements in order to stop/pause/play a given movie. However, I can hardly imagine a user willing to deliberately place his/her hand scanned by a video camera into the bounding rectangle and thus be inevitably disturbed during the replay of a movie. The application would have to be professionalized, i.e. the camera would have to detect user's hand gestures without the aid of the bounding rectangle. Furthermore, the hand gesture detection would have to be made nearly perfect – the user only needs to raise his/her hand and produce a gesture. And that's it. The recognition must be performed immediately and without mistakes, otherwise the user would become irritated and would stop using the application.

5.5.2 Some thoughts on future of visual recognition

Generally speaking about visual recognition software: Although there seem to be professional systems made commercially available (Microsoft's Natal/Kinect, HandVU, face recognition systems in cameras or notebooks), I remain a little skeptical about their usability and user-friendliness. Imagine yourself working with such a system – your hands would float in the air constantly (as seen in the *Minority Report* movie). That would become physically very tiring after some time. In my opinion, user's hands must remain leaned against some sort of a rest (a table for instance) and the software must detect just the motion of user's fingers just like the current devices (mouse, trackball, touchpad..) do. Subsequently, we could give up using the above referred devices to control a computer – bare hands would be enough. On the other hand, try typing on a virtual keyboard (as seen on iPad for instance) – I believe that a classical keyboard is still much more comfortable, because people are just used to control something by touching it. This is why I believe that visual recognition will not be used in near future to directly control a computer

in terms of replacing computer mouse devices, trackballs, or touchpads. At least not until it is made perfect. I could imagine using visual recognition instead of a TV remote control to just switch channels or to navigate a simple internet browser (without too much typing), however, for text input tasks, for accurate graphical input (tablet, mouse, trackballs), or for other accuracy-demanding tasks, physical devices will still be needed.

On the other hand, Microsoft's Natal/Kinect system wants to make users play games without any physical controllers – the only controller should be their bodies and voice. In this sense, physical weariness caused by the necessity to move one's body just to control a computer might be a wanted thing here – a user playing virtual tennis might want to be physically tired afterwards! Besides, Microsoft promises a whole new game experience when playing and taking advantage of one's own body used as a computer controller. On the other hand, isn't it easier to just go and play a real tennis game in a real tennis court?

6. Other hand-gesture recognition systems

6.1 HandVU

HandVU is a fairly advanced system built on top of the OpenCV library. It can be downloaded, including documentation, here: <http://www.movesinstitute.org/~kolsch/HandVu/HandVu.html#download>

Here is the description provided on the website: *"This software collection implements a vision-based hand gesture interface. Called HandVu, it detects the hand in a standard posture, then tracks it and recognizes key postures - all in real-time and without the need for camera or user calibration. The output is accessible through library calls, a client-server infrastructure in a custom format and as OpenSound Control packets, allowing control of music applications that use this format.*

The software package consists of the main HandVu library and several applications that demonstrate video capture with OpenCV's highgui, DirectShow, and soon the ARtoolkit. If your application includes video processing, you would just make a few library calls and hand gesture recognition results will be available to you. If your application does not use camera input already, you can run one of the sample programs on the side and interface to your application with a networking interface. Functionality to capture screenshots and snapshots of the tracked object is available, allowing data collection for research or other purposes."

6.2 Project Natal/Kinect

Project Natal/Kinect is a Microsoft's endeavor in the field of visual recognition. Microsoft is planning to use this system on its Xbox gaming platform. The system itself is an advanced combination of visual gesture recognition, voice recognition, facial recognition and 3D motion capture. Consequently, the user can use his/her voice a body motion to control the Xbox gaming console.

6.2.1 Kinect's Technological background

If you want to use the Kinect capabilities with the Xbox, you need to buy a special set of equipment. There is a camera, depth sensor and a microphone built into the equipment. Moreover, Kinect uses also an infrared sensor and a monochrome CMOS sensor in order for the Kinect to work under various light ambient conditions.

This is something I see as a very elegant solution because in the beginning of my development of my application, I wanted to use a standard RGB camera and its RGB video picture. However, the colors changing under various light ambient conditions caused a lot of trouble to the gesture recognition algorithms I was planning to use. Here, Microsoft uses a monochrome CMOS sensor in

order to get rid of unnecessary colors. It's actually a hardware solution which simplifies further image processing.

Here is a little bit more on Kinect's hardware [15]:

"...the Kinect sensor outputs video at a frame rate of 30 Hz, with the RGB video stream at 32-bit color VGA resolution (640×480 pixels), and the monochrome video stream used for depth sensing at 16-bit QVGA resolution (320×240 pixels with 65,536 levels of sensitivity). The Kinect sensor has a practical ranging limit of 1.2–3.5 meters (3.9–11 ft) distance. The sensor has an angular field of view of 57° horizontally and a 43° vertically, while the motorized pivot is capable of tilting the sensor as much as 27° either up or down. The microphone array features four microphone capsules, and operates with each channel processing 16-bit audio at a sampling rate of 16 kHz."

The whole system monitors the user in order to evaluate their movements and transfer them into a virtual world action. It not only detects user's hands, it detects his/her whole body so that the user can control his/her character in the game. The game experience should be, according to Microsoft, breathtaking as this is the first time when a user can actually play a game using his/her whole body.

This is what Microsoft writes at their webpage [16]: *"Project Natal" is the code name for a revolutionary new way to play on your Xbox 360. Natal is pronounced "nuh-tall". Project Natal' represents controller-free gaming and entertainment. It won't just be the greatest videogames to benefit from so much freedom. In addition to "Project Natal" tracking your full body movement in 3-D, it also recognizes your face and facial expressions. It can even detect the change of emotion in your voice. The device also remembers your face and voice so that you can connect to Xbox LIVE without the need to reach for a controller. This is the future!"*

Unfortunately, no detailed information about this system is available at the moment of writing this thesis, it seems, however, that it is probably a fairly developed system combining visual and audio recognition, almost ready for

commercial use. As such, it is naturally much more advanced than the application which is in the center of attention of this thesis.

6.2.2 Kinect's History

At the time of writing this thesis, the Kinect system was a fairly new thing. Introduced in 2009, it was originally called Natal, the name being chosen after Alex Kipman (one of Kinect researchers) who comes from Brazil – Natal is a city in Brazil. There were also some rumors about Microsoft developing a new Xbox 360 console designed for Kinect, which, however proved not to be true [15]: *“It was rumored that the launch of Project Natal would be accompanied with the release of a new Xbox 360 console (as either a new retail configuration, a significant design revision and/or a modest hardware upgrade). Microsoft dismissed the reports in public, and repeatedly emphasized that Project Natal would be fully compatible with all Xbox 360 consoles. Microsoft indicated that the company considers it to be a significant initiative, as fundamental to the Xbox brand as Xbox Live, and with a launch akin to that of a new Xbox console platform. Kinect was even referred to as a "new Xbox" by Microsoft CEO Steve Ballmer at a speech for the Executives' Club of Chicago. When asked if the introduction will extend the time before the next-generation console platform is launched (historically about 5 years between platforms), Microsoft corporate vice president Shane Kim reaffirmed that the company believes that the life cycle of the Xbox 360 will last through 2015 (10 years).”*

6.3 SONY and the MOVE system

Another gesture-recognition system is the MOVE made by SONY. Sony decided to simplify the things a bit down and used something what is called a wand. What is actually a wand? It's a controller with an orb on top of it. There is a camera in the system that can track the orb within the visual field. The orb has actually two functions: it simplifies the visual detection and it provides a

visual feedback to the user. Now, how does that work? As mentioned above, Sony simplified the things down. By using the orbs, the wand can be relatively easily detected within the visual field. How so? The orb is capable of emitting light of various colors (all RGB colors) while the software running within the system decides what color is suitable for a current visual background, e.g. if the background is mostly greenish, the software might decide that the red color might be easily recognized within such a greenish background, thus the orb starts glowing in red. Should the background color change to mostly grey, the system might pick yellow or white color for the orb to glow. What is actually meant by the background here? Background is everything that's outside the orb, i.e. it might be a table with chairs, a window behind the videogame player or even the player itself. The only thing that matters to the system is the orb. This greatly simplifies the things down because when the system decides that the orb should glow in red, it can search the visual field for exactly such a color and identify it with a near 100% certainty. Like this, the orb can be easily tracked and its motion can be applied to control a game that's currently running in the console.

The other function of the orb is to provide visual feedback to the user. Thus, it can flash in red, white and yellow if the user fires a weapon, or it can indicate the color of a brush should the user use a painting program.

The wand with the orb, the actual controller, tends to be somehow despised by users and they might be right as the controller itself does not look especially attractive. However, its strength lies in the fact that it is extremely accurate, for A), the orb can be tracked very easily and accurately within the visual field as its color can be changed dynamically to contrast with the background and the system already "knows" what to look for, and B), the wand contains an accelerometer, an angular rate sensor (these are used to track rotation and overall motion of the controller) and also a magnetometer, which uses the Earth's magnetic field to determine its position against the Earth. All these

features help to accurately process the position and motion of the controller and deliver it to the software running inside the gaming console.

7. Benefits to the reader

This thesis is a study, a sort of log book that describes the creation of a hand gesture visual recognition software. It provides a starting point to those who want to experiment with hand gesture recognition. Algorithms for background subtraction, contour finding and shape recognition are brought together to produce a real-world application. In this thesis, readers will find basic guidance and principles that can be used to build more powerful applications. Principles described here can be used universally, e.g. for recognition of people and other objects and to build human-computer interfaces.

8. Conclusion

Within the framework of this bachelor's thesis, I created an application that is capable of real-time static hand gesture recognition.

Along the way, I described the whole process of its creation using the background segmentation, contour finding and Hu moments recognition techniques. Under ideal lighting, the application works quite well with its weak point lying in poor lighting because under such conditions, the contours get misinterpreted, influencing the application's performance in a negative way.

I also mentioned other recognition systems, like Microsoft Kinect, Sony Move and HandVU with Microsoft's and Sony's systems being very much developed and advanced, thus ready for commercial deployment.

Literature:

- 1) Bradski, Gary; Kahler, Adrian: *Learning OpenCV, Computer Vision with the OpenCV Library*, 2008, First Edition, O'Reilly Media, Inc., ISBN: 978-0-596-51613-0
- 2) Davies, G. R.: *Machine Vision, Theory, Algorithms, Practicalities*, Third Edition, Morgan Kaufmann Publishers, Elsevier, 2005, ISBN: 0-12-206093-8
- 3) Gonzales, Rafael C.; Woods, Richard E.: *Digital Image Processing*, Second Edition, Prentice Hall, New Jersey, 2002, ISBN / ASIN: 0201180758
- 4) Suchý, Václav: *Rozpoznávání textu v obraze (Optical Character Recognition)*, Vysoké učení technické v Brně, Fakulta informačních technologií, Bakalářská práce, Brno 2007
- 5) Novák, Ondřej; Ondřej, František; Ondřej, Jan: *Gesture recognition (a paper on gesture recognition)*,
- 6) *OpenCV description*, (as of 3 Dec 2010), Wikipedia, available from WWW:
<<http://en.wikipedia.org/wiki/OpenCV>>
- 7) *HandVU*, description and download, (as of 3 Dec 2010), available from WWW:
<<http://www.movesinstitute.org/~kolsch/HandVu/HandVu.html#download>>
- 8) *Machine vision*, (as of 3 Dec 2010), Wikipedia, available from WWW:
<http://en.wikipedia.org/wiki/Machine_vision>

9) Lesniak, Daine Richard; Hickok, Douglas, J.; Whisler, Kristopher; Rowe, Michael, C., Ph.D: *An Autonomous Missile Defense System*, (as of 3 Dec 2010), available from WWW: <http://www.uwplatt.edu/csse/courses/prev/csse411-materials/StudentConferencePublications/MICS2005_An%20Autonomous%20Missile%20Defense%20System.pdf>

10) *Computer vision*, (as of 3 Dec 2010), Wikipedia, available from WWW: <http://en.wikipedia.org/wiki/Computer_vision>

11) *Driveless car*, (as of 3 Dec 2010), Wikipedia, available from WWW: <http://en.wikipedia.org/wiki/Driverless_car>

12) Massimo Picardi, Tony Jan: *Recent advances in computer vision*, The Industrial Physicist web page, (as of 8 Dec 2010), available from WWW: <<http://www.aip.org/tip/INPHFA/vol-9/iss-1/p18.html>>

13) *Optical Character Recognition*, (as of 8 Dec 2010), Wikipedia, available from WWW: <http://en.wikipedia.org/wiki/Optical_character_recognition>

14) Seo, Naotoshi: *Tutorial: OpenCV haartraining (Rapid Object Detection With A Cascade of Boosted Classifiers Based on Haar-like Features)*, (as of 8 Dec 2010), available from WWW: <<http://note.sonots.com/SciSoftware/haartraining.html>>

15) *Kinect*, (as of 8 Dec 2010), Wikipedia, available from WWW: <<http://en.wikipedia.org/wiki/Kinect>>

16) *Project Natal*, (as of 8 Dec 2010, not available anymore), <<http://www.microsoft.com/uk/wave/hardware-projectnatal.aspx>>