VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

HEAD POSE ESTIMATION AND TRACKING

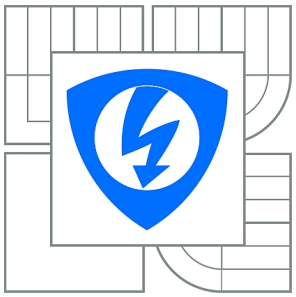
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. ALEŠ POSPÍŠIL
AUTHOR

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

HEAD POSE ESTIMATION AND TRACKING

DETEKCE A SLEDOVÁNÍ POLOHY HLAVY V OBRAZE

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

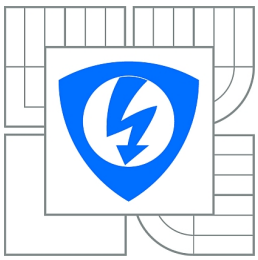
AUTOR PRÁCE — Bc. ALEŠ POSPÍŠIL

AUTHOR

VEDOUCÍ PRÁCE — Ing. JIŘÍ PŘINOSIL, Ph.D.

SUPERVISOR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Aleš Pospíšil	***ID:***	72921
Ročník:	2	***Akademický rok:***	2010/2011

NÁZEV TÉMATU:

Detekce a sledování polohy hlavy v obraze

POKYNY PRO VYPRACOVÁNÍ:

Prostudujte moderní metody číslicového zpracování obrazových signálů orientované na detekci a sledování pohybu lidské hlavy. Na základě získaných teoretických znalostí navrhněte metodu pro určení aktuální pozice hlavy včetně úhlu jejího natočení vzhledem k snímacímu zařízení. Navrženou metodu následně implementujte ve vhodném programovacím jazyku a stanovte podmínky její použitelnosti.

DOPORUČENÁ LITERATURA:

[1] Murphy-Chutorian, E.,Trivedi, M.M.: Head Pose Estimation and Augmented Reality Tracking: An Integrated System and Evaluation for Monitoring Driver Awareness, IEEE Transactions on Intelligent Transportation Systems, ISSN: 1524-9050, 2010.
[2] Nixon, M., Aguado, A.: Feature Extraction & Image Processing, Academic Press, ISBN: 978-0-1237-2538-7, 2008.

Termín zadání:	7.2.2011	***Termín odevzdání:***	26.5.2011

Vedoucí práce: Ing. Jiří Přinosil, Ph.D.

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Abstract

This thesis is focused on head pose estimation as a possibility to bridge a gap between human and computers. The main contribution of the thesis is usage of a novel hardware and software technology such as Kinect, Point Cloud Library and CImg Library. In the beginning of the thesis overview of prior works in a field of head pose estimation is provided. Own database was created in order to test and evaluate developed algorithm. The head pose estimation and tracking system is based on 3D image data acquisition and Iterative Closest Point registration algorithm. In the end of thesis performance and limitation of the system are described and further improvement is proposed.

Keywords

Head pose estimation, head pose tracking, face detection, point cloud, Iterative Closest Point, ICP, image registration, Kinect, Point Cloud Library, PCL, depth image, transformation matrix, Euler angles, 3D geometry.

Anotace

Diplomová práce je zaměřena na problematiku detekce a sledování polohy hlavy v obraze jako jednu s možností jak zlepšit možnosti interakce mezi počítačem a člověkem. Hlavním přínosem diplomové práce je využití inovativních hardwarových a softwarových technologií jakými jsou Microsoft Kinect, Point Cloud Library a CImg Library. Na úvod je představeno shrnutí předchozích prací na podobné téma. Následuje charakteristika a popis databáze, která byla vytvořena pro účely diplomové práce. Vyvinutý systém pro detekci a sledování polohy hlavy je založený na akvizici 3D obrazových dat a registračním algoritmu Iterative Closest Point. V závěru diplomové práce je nabídnuto hodnocení vzniklého systému a jsou navrženy možnosti jeho budoucího zlepšení.

Klíčová slova:

Určení polohy hlavy, sledování polohy hlavy, detekce obličeje, mračno bodů, Iterative Closest Point, PCL, registrace obrazu, Kinect, Point Cloud Library, PCL, prostorový obraz, transformační matice, Eulerovy úhly, 3D geometrie.

POSPÍŠIL, A. *Detekce a sledování polohy hlavy v obraze* . Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2011. 58 s. Vedoucí diplomové práce Ing. Jiří Přinosil, Ph.D..

Prohlášení

Prohlašuji, že jsem svoji diplomovou práci na téma *Detekce a sledování polohy hlavy v obraze* vypracoval samostatně pod vedením svého školitele s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v rámci práce a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nepovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení §11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení §152 trestního zákona č. 140/1961 Sb.

V Brně dne ....

Podpis autora

Acknowledgements

I would like to express my deep and sincere gratitude to my supervisor, Professor Alberto Albiol Ph.D., Polytechnic University of Valencia, for allowing me to work with him, his warm welcome in Valencia and his guidance throughout my thesis-writing period.

I am deeply grateful to my supervisor, Ing. Jiří Přinosil, Ph.D., Brno University of Technology, for his tolerant approach and wise advices I received from him.

I wish to express my warm and sincere thanks to Image Processing Lab team at Polytechnic University of Valencia, especially Javier Oliver, Jordi Mansanet and David Monzo, for the valuable advices, help and fun I had with them during my stay at the Lab.

I owe my loving thanks to my sister, Klára Pospíšilová, for providing a loving environment for me.

Lastly, and most importantly, I wish to thank my parents, Marie Pospíšilová and Eduard Pospíšil. They bore me, raised me, supported me, taught me, and loved me. To them I dedicate this thesis.

Contents

List of Figures

List of Tables

1 Introduction

The common human ability to estimate the head pose of another person presents a unique challenge for computer vision systems. Previous face-related vision research was mainly focused on face detection and recognition. Therefore there are fewer head pose estimation systems that are rigorously evaluated.

From an early age, people display the ability to quickly and effortlessly interpret the orientation and movement of a human head, thereby allowing one to infer the intentions of others who are nearby and to comprehend an important nonverbal form of communication. The ease with which one accomplishes this task belies the difficulty of the problem that has challenged computational systems for decades.

The head pose estimation problem can be understood by different ways. In the context of computer vision, head pose estimation is most commonly interpreted as the ability to infer the orientation of a person's head relative to the view of a camera. More generally speaking, head pose estimation is the ability to infer the orientation of a head relative to a global coordinate system. The orientation of head is based on multiple degrees of freedom (DOF). A system considering only a single DOF, usually the left to right movement is still a head pose estimator, as is the more complex approach that estimates a full 3D orientation and position of a head, while using additional DOF. In the complex approach human head is limited to the three DOF in pose, which can be characterized by pitch, roll and yaw angle as shown in Fig. 1.

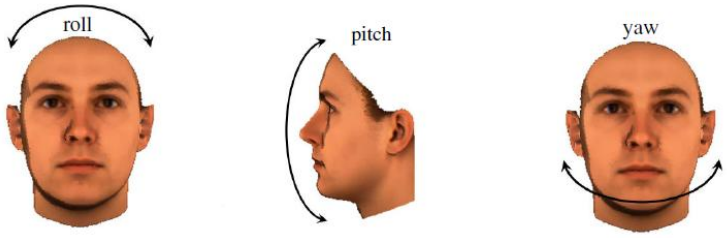


Figure 1 The three degrees of freedom of a human head can be described by the rotation angles pitch, roll and yaw

What is our motivation in the field of head pose estimation? People use the orientation of their heads to transmit rich, interpersonal information. For example, a person will point the direction of his head to indicate who is the intended target of a conversation. Similarly, in a dialogue, head direction is a nonverbal communication that cues a listener when to switch roles and start speaking. There is an important meaning in the movement of the head as a form of gesturing in a conversation. People nod to indicate that they understand what is being said and they use additional gestures to indicate dissent, confusion, consideration, and agreement. Exaggerated head movements are synonymous with pointing a finger, and they are a conventional way of directing someone to observe a particular location. Like a speech recognition, which has already became widely used in many available technologies, head pose estimation will likely become a tool to bridge the gap between humans and computers.

1.1 Goal of Thesis

This thesis investigates possible implementation of a head pose estimation and tracking system using novel hardware technologies such as Microsoft Kinect and software technologies such as Point Cloud Library. The topic of this thesis was proposed by associate professor Alberto Albiol, Image Processing Lab at the Polytechnic University of Valencia. Outcomes of this thesis will serve as a building block for future applications in the field of computer vision and image processing. The goal of the thesis is to create a general head pose estimation block that can be further modified or implemented according to specific needs. Good understanding of used technologies is crucial for successful work.

1.2 Organization of Thesis

In Chapter 2 the thesis begins with information about prior work in a field of head pose estimation. Different head pose estimation approaches together with their advantages and disadvantages are presented. Chapter 3 provides information about technologies that were used to develop the head pose estimation system itself. Chapter 4 provides information about database that had to be created to test and evaluate the system. Chapter 5 describes developed head pose estimation system and provides details about all main building blocks. Chapter 6 is focused on measurements that were done using the database and on evaluation of measurement's outcomes. Chapter 7 concludes the results of the thesis and summarize all the work that was done.

2 Prior Work

For last 15 years many researchers have been involved in investigating suitable approach and framework for head pose estimation. Many methods were examined and evaluated. There are systems that require stereo depth information and also systems that require only monocular video. Similarly, some systems require a near-field view of a person's head while other can adapt the low resolution of far-field view. To be able to evaluate prior work it would be difficult to consider all mentioned system obstacles therefore this work will be based on evolutionary taxonomy presented by Erik Murphy-Chutorian and Mohan Manubhai Triveldi in their head pose estimation survey [1].

Systems presented in the survey are arranged by their fundamental approach that underlies its implementation. There are eight categories that describe approaches that have been used to estimate head pose:

1. **Appearance template methods** compare a new image of a head to a set of exemplars (each labeled with a discrete pose) in order to find the most similar view.

2. **Detector array methods** train a series of head detectors each attuned to a specific pose and assign a discrete pose to the detector with the greatest support.

3. **Nonlinear regression** methods use nonlinear regression tools to develop a functional mapping from an image or feature data to a head pose measurement.

4. **Manifold embedding methods** seek low-dimensional manifolds that model the continuous variation in head pose. New images can be embedded into these manifolds and then used for embedded template matching or regression.

5. **Flexible models** fit a nonrigid model to the facial structure of each individual in the image plane. Head pose is estimated from feature-level comparisons or from the instantiation of the model parameters.

6. **Geometric methods** use the location of features such as the eyes, mouth, and nose tip to determine pose from their relative configuration.

7. **Tracking methods** recover the global pose change of the head from the observed movement between video frames.

8. **Hybrid methods** combine one or more of these before mentioned methods to overcome the limitations inherent in any single approach.

In the following section detail information about functional requirements, advantages and disadvantages are provided for each method.

2.1 Appearance template methods

Appearance template methods use image-based comparison metrics to be able to match an image of person's head to a set of exemplars with corresponding pose labels. In the simplest implementation, the processed image is given the same pose that is assigned to the most similar of these templates.

These methods have some advantages over more complicated methods. The templates can be expanded to a large set at any time what allows systems to adapt changing conditions. Appearance templates do not require negative training examples or facial feature points what decreases the amount of requirements for data set quality.

At the same time there are many disadvantages. Without any additional technique are these methods only capable to estimate discrete head pose locations. Methods also assume that the head region has already been detected and localized. More computationally expensive image comparisons make from appearance template methods less efficient choice. The most significant problem with appearance templates is that they operate under faulty assumption that pairwise similarity in the image space can be equated to similarity in pose. Take example of two images of the same person in slightly different poses and two images of different people in the same pose. In this scenario, identity aspects can cause more dissimilarity in the image than from different people in the same pose, and template matching would make faulty pose estimation [1][14].

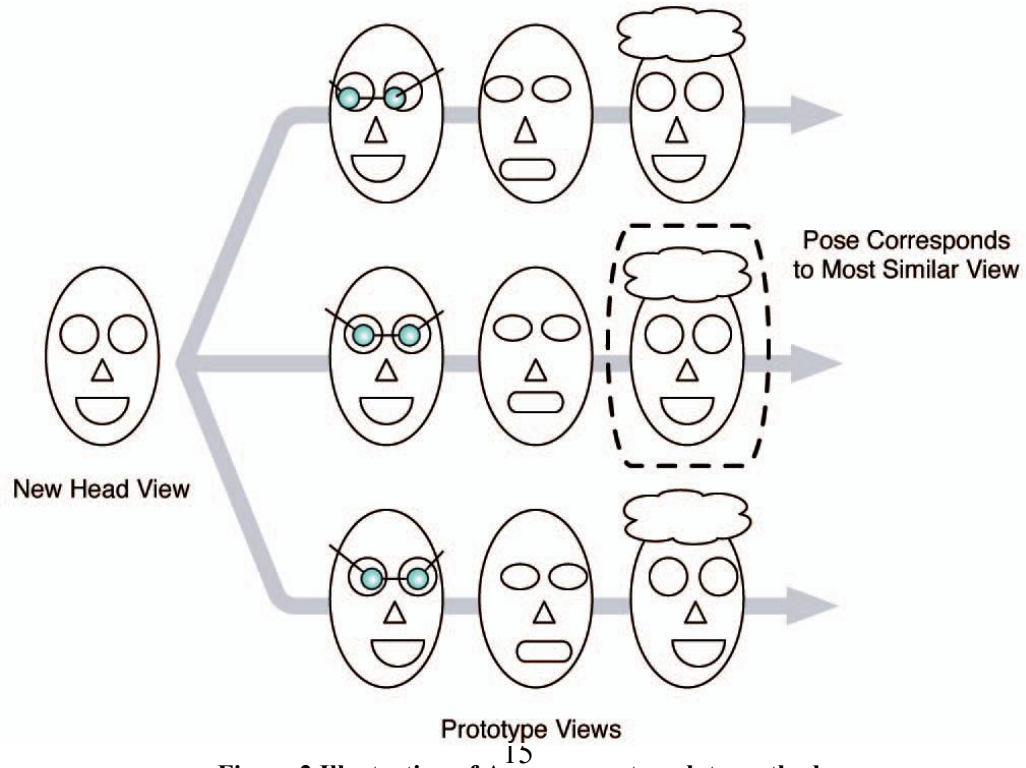


Figure 2 Illustration of Appearance template method

2.2 Detector array methods

Detector arrays are similar to appearance templates because they operate directly on an image patch. Instead of comparing an image to a large set of individual templates, the image is evaluated by a detector trained on many images with supervised learning algorithm. Examples of a successfully implemented detector array are three Support Vector Machines (SVMs) or five FloatBoost classifiers operating in a far-field multicamera setting.

An advantage of detector array methods is that a separate head and localization steps are not required, since each detector is also capable of making the distinction between head and nonhead. Detector arrays also work well for both high and low-resolution images.

Disadvantage of detector array method is the difficultness of trainings many detectors for each discrete pose. More training data is required due to the need for negative nonface images which are necessary for proper detector training. Finally, the computational complexity increases linearly with the number of detectors, making it difficult to implement a real-time system with large array [1][15].

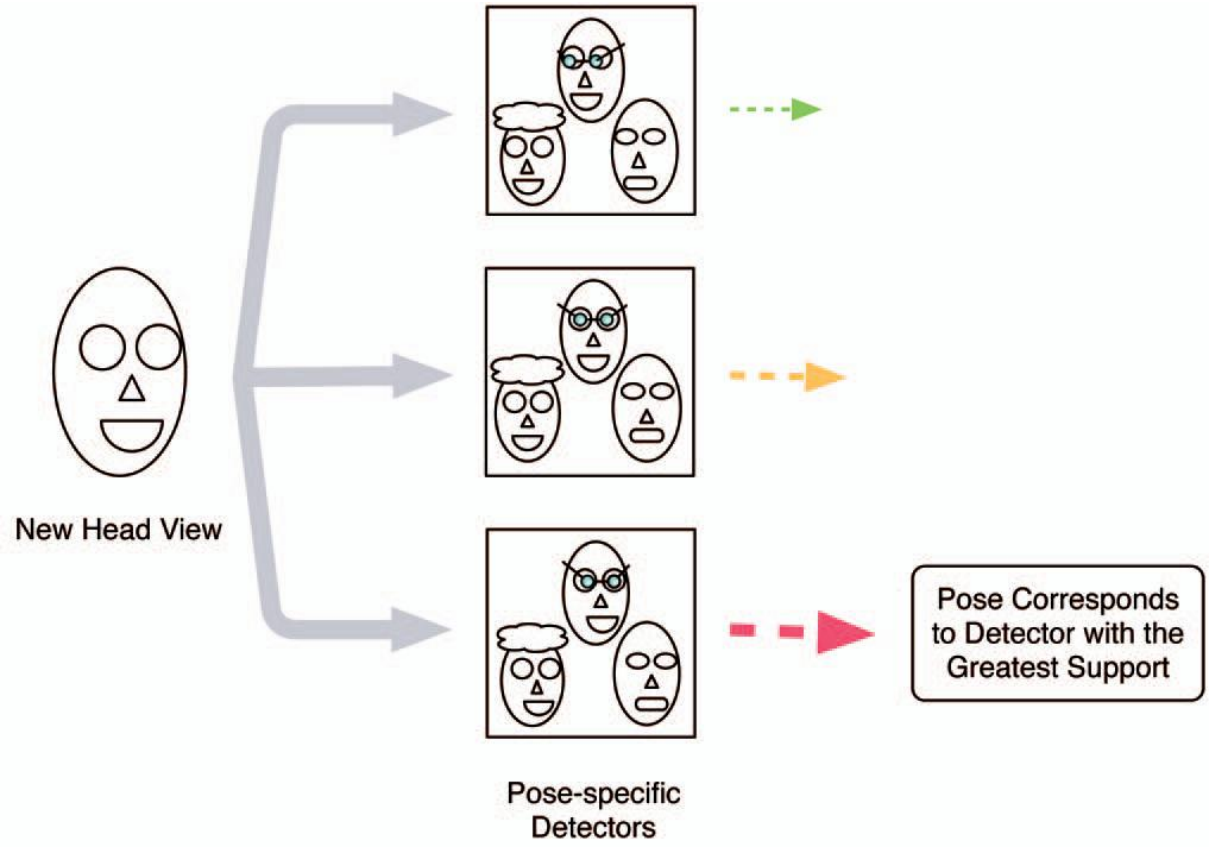


Figure 3 Illustration of Detector array method

2.3 Nonlinear regression

Nonlinear regression methods estimate pose by learning a nonlinear function mapping from the image space to one or more pose directions. The benefit of this approach is that with a set

of labeled training data, a model can be built that will provide discrete or continuous pose estimate for any new data sample. One of the requirements for this method is decrease of high dimensionality of an image. This may be achieved using principal component analysis (PCA) or localized gradient orientation (LGO) histogram.

These systems are very fast, only require cropped labeled faces for training, work well in near-field and far-field imagery, and give some of the most accurate head pose estimates in practice [1].

The main disadvantage of these methods is that they are error-prone due to poor head localization [16].

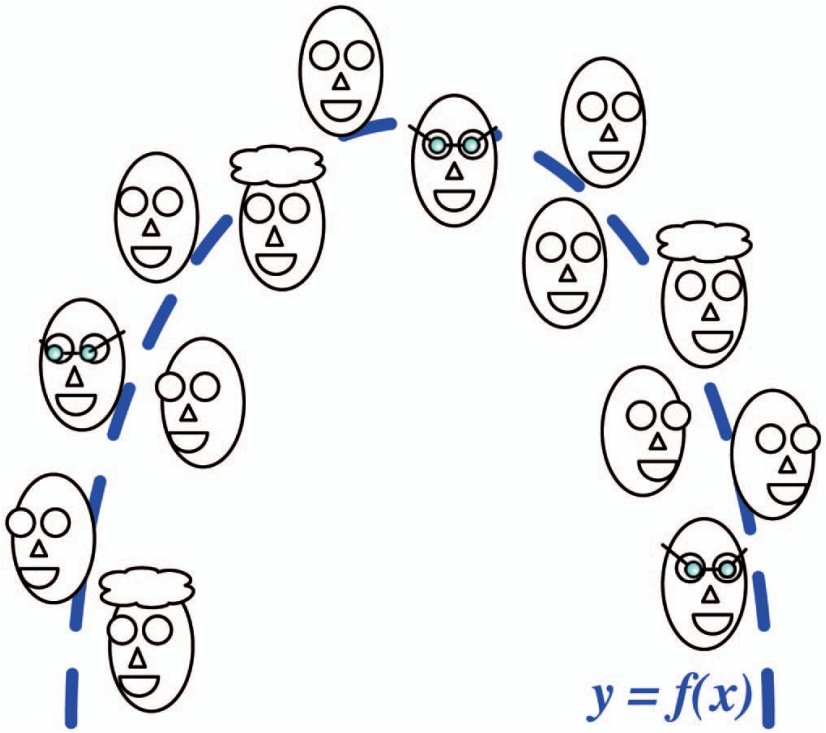


Figure 4 Illustration of Nonlinear regression method

2.4 Manifold embedding methods

Although an image of a head is considered as data sample in high-dimensional space, there are many fewer dimensions in which pose can vary. With a rigid model of head, this can be as few as three dimensions for orientation and three for position. For head pose estimation, the manifold must be modelled and the embedding technique is needed to project a new sample into the manifold. This low-dimensional embedding can then be used for head pose estimation with techniques such as regression in the embedded space or embedded template matching.

There are both linear and nonlinear approaches for manifold embedding. The linear techniques have the advantage that embedding can be performed by matrix multiplication, but they lack the representational ability of nonlinear techniques [1][17].

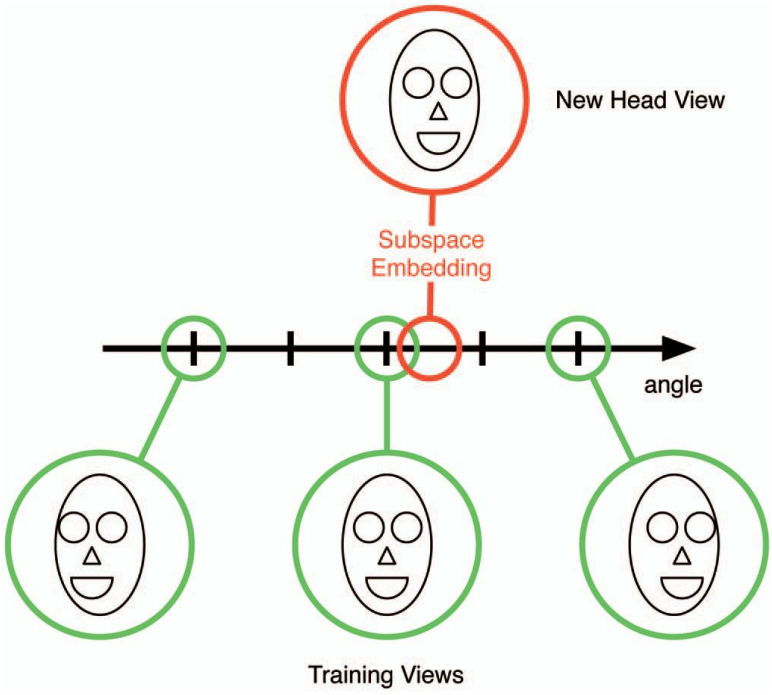


Figure 5 Illustration of Manifold embedding method

2.5 Flexible models

In contrast to previous methods that consider head pose estimation as a signal detection problem, mapping a rectangular region of image pixels to a specific pose orientation flexible models take a different approach. A nonrigid model is fitted the image in a way that conforms to the facial structure of each individual. These methods require training data with annotated facial features.

Flexible models approach offer very good invariance to head localization error, since they adapt to the image and find the exact location of facial features. This allows for precise and accurate head pose estimation.

The main limitation of flexible models is that all the facial features must be located in each image frame what create obstacles during data set creating. In practice, these approaches are limited to head pose orientation where the outer corners of both eyes are visible. There is also not evident success for far-field head pose estimation with low-resolution facial images [1][18].

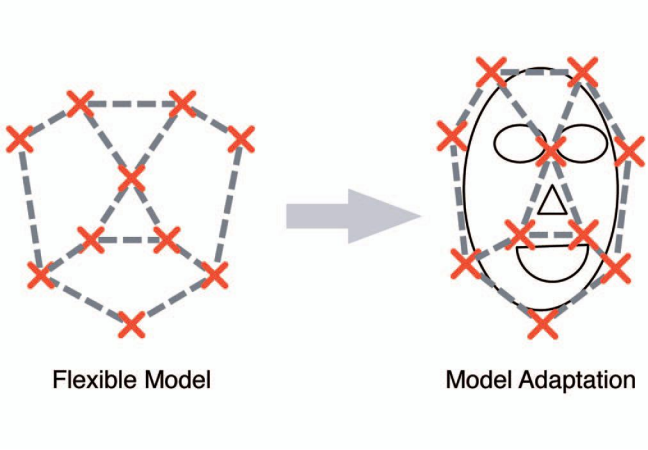


Figure 6 Illustration of Flexible models method

2.6 Geometric methods

Thanks to psychological experiments we are aware how human perceive head pose. Based on this knowledge geometric methods were developed. These methods consider the human perception of head pose to rely on cues such as the deviation of the nose angle and the deviation of the head from bilateral symmetry. Geometric approaches to head pose estimation use head shape and precise configuration of local features to estimate pose. Commonly there are five facial points used in this method (the outside corner of each eye, the outside corners of the mouth, and the tip of the nose).
Geometric methods are fast and simple. With knowledge of only few facial features, a decent estimate of head pose can be reached.

The obvious difficulty lies in detecting the features with high precision and accuracy. Far-field imagery is problematic since the resolution can make it difficult or impossible to determine the feature locations. Also situations such as when a person wears glasses can decrease likelihood of proper facial features detection [1][19].

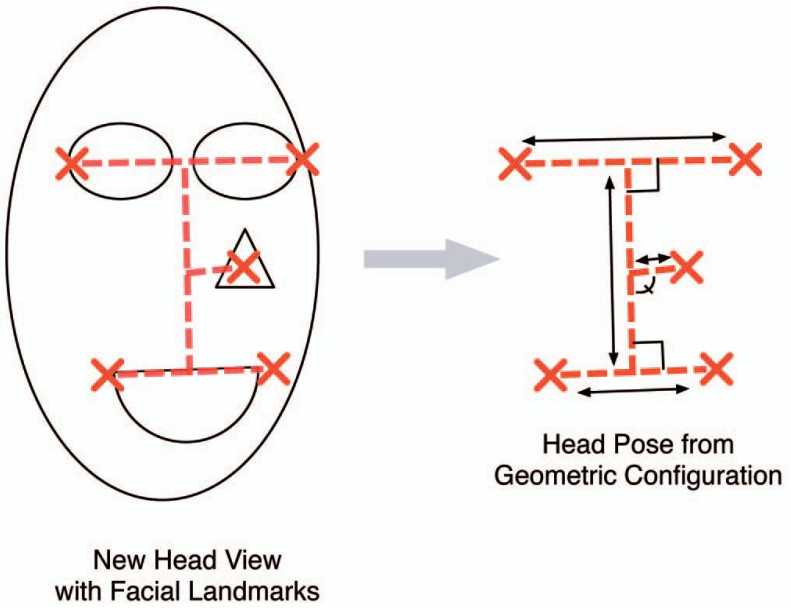


Figure 7 Illustration of Geometric methods

2.7 Tracking methods

Tracking methods are designed to follow the relative movement of the head between consecutive frames of a video sequence. Temporal continuity and smooth motion constraints are utilized to provide visual estimate of head pose over time. The systems usually offer a high level of accuracy, but initialization from a known head position is requisite. Therefore these approaches often rely on manual initialization or camera view such that the subject's neutral head pose is forward-looking and easily reinitialized with a frontal face detector.

The main advantage of tracking approaches is their ability to track the head with high accuracy by discovering the small pose changes between video frames. These methods consistently outperform other head pose estimation approaches [1].

The disadvantage of tracking methods is connected to the requisite of accurate initialization of position and poses to generate new model or adapt an existing model. Without a separate localization step, these approaches can only be used to discover the relative transformation between frames [20].

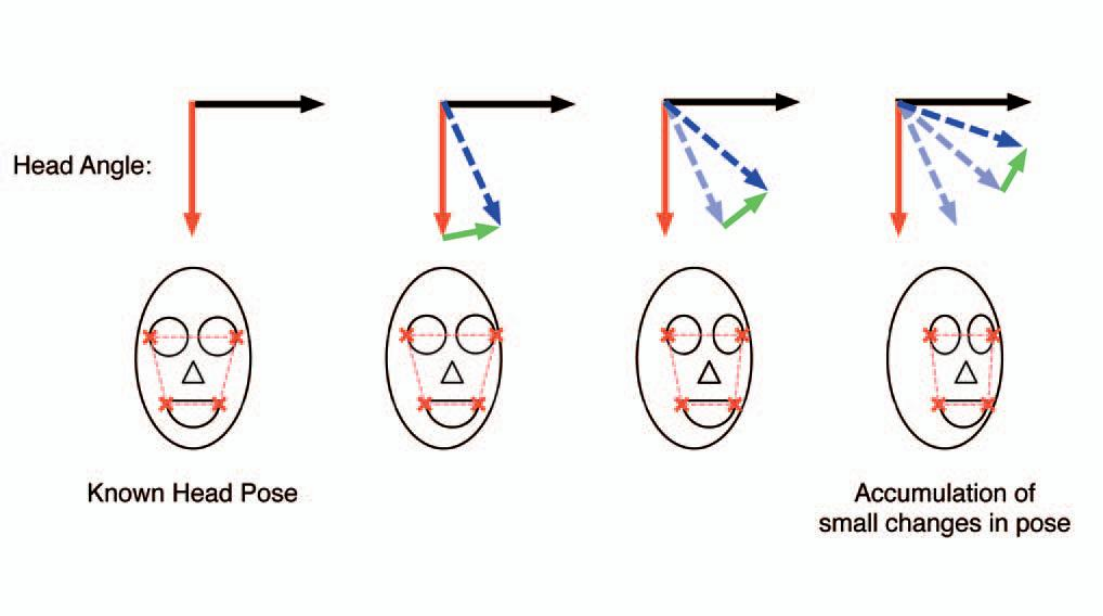


Figure 8 Illustration of Geometric methods

2.8 Hybrid methods

Hybrid approaches combine one or more of the above mentioned methods to estimate pose. These systems can be designed to overcome the limitations of any one specific head pose estimation category. Common example is to extend a static head pose estimation approach by a tracking system. The static system is in this case responsible for initialization and the tracking system is responsible for maintaining pose estimates over time [1][21][2].

There are many possible combinations and implementations and both advantages and disadvantages depend on specific approach that is used for hybrid method.

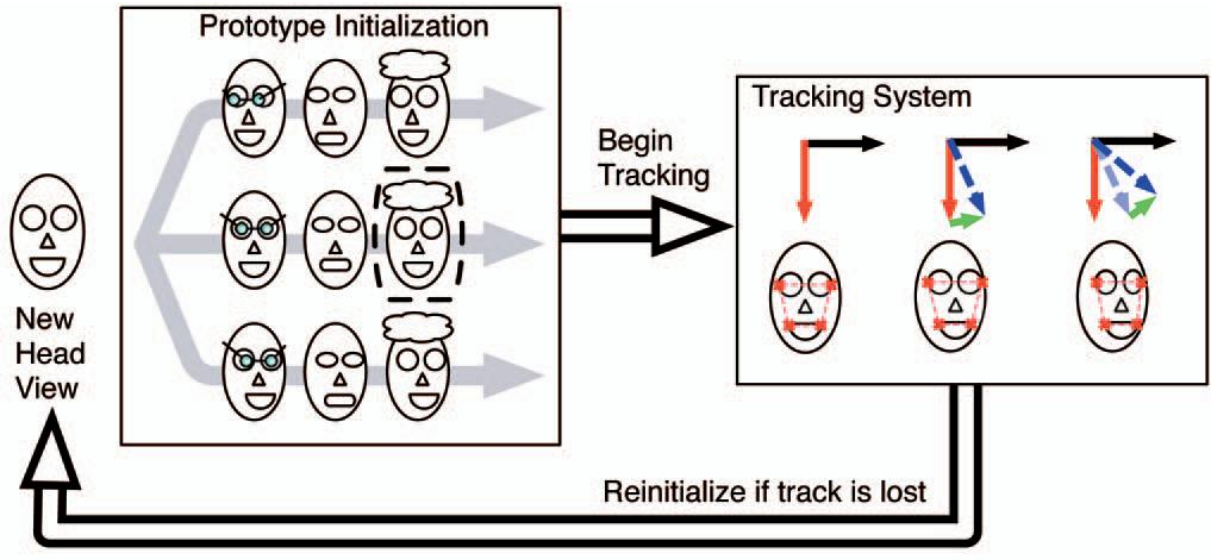


Figure 9 Illustration of Hybrid methods

3 Involved Technologies

This chapter describes the core technologies involved in solving head pose estimation problem within this work. One of possible ways how to deal with head pose estimation is usage of stereo camera to acquire 3D information from observed scene. Recently, November 2010 [22], new motion sensing input device by Microsoft for the Xbox 360 video game console called Kinect was released. Kinect in its core represents low-cost and affortable piece of hardware that is used in this work. Software support for this work represent primarily following libraries: Point Cloud Library, a **large scale, open project** for point cloud processing, and CImg, an **open source**, **C++ toolkit** for **image processing.** Brief description of mentioned technologies follows.

3.1 Kinect

Kinect is final product of Project Natal lead by Microsoft based on range camera technology developed by PrimeSense. Kinect as an extension for Xbox 360 opens new level of human computer interaction in this case with focus on game experience. Users are allowed to control running game/application just by hand or body gestures. This functionality is powered by two innovative features: depth information acquisition and body skeleton tracking. For this work first mentioned depth information acquisition is important therefore no description of body skeleton tracking will follow.

Figure 10 Kinect device from Microsoft

Kinect holds Guiness World Record of being the "fastest selling consumer electronics device" with 8 millions units sold in its first 60 days [23]. 10 million units of the Kinect sensor have been shipped as of March 9, 2011 [24]. Many units weren't bought with intention to play games but to explore possibility of 3D image processing, gesture recognition, body

tracking and other specific tasks. There are many interesting examples of "Kinect hacks" from around world available on the internet [26][27]. The main reason for such commercial success was low price (currently about 140$ per unit [25]) for a device that provides 3D information of scene in sufficient quality for research and development.

3.1.1 Depth acquisition

To understand how depth information acquisition works the hardware structure of Kinect has to be described. As can be seen in Figure 11 Kinect consists of two cameras (one RGB, second IR) and one laser-based IR projector [28]. The IR camera and the IR projector form a stereo pair with baseline of approximate 7.5 cm. The IR projector sends out a fixed pattern of light and dark areas shown in Figure 12. The pattern is generated from a set of diffraction gratings, with special care to eliminate the effect of zero-order propagation of centre bright dot. Depth is then calculated by triangulation of pattern received through IR camera against known pattern emitted by IR projector. This technology called Light Coding was developed by PrimeSense and is suitable for any indoor environment [29].

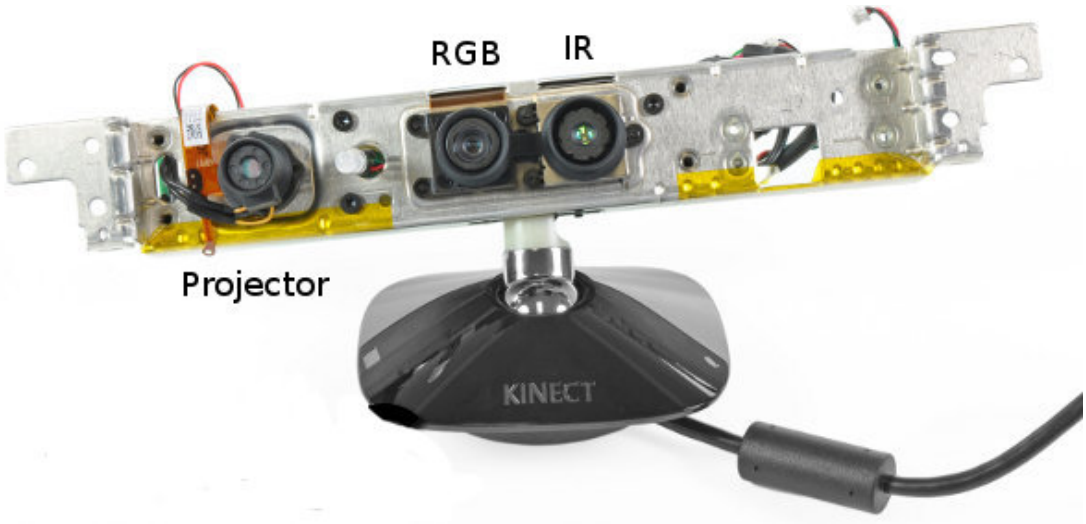


Figure 11 Uncovered Kinect device showing RGB and IR camera and IR projector

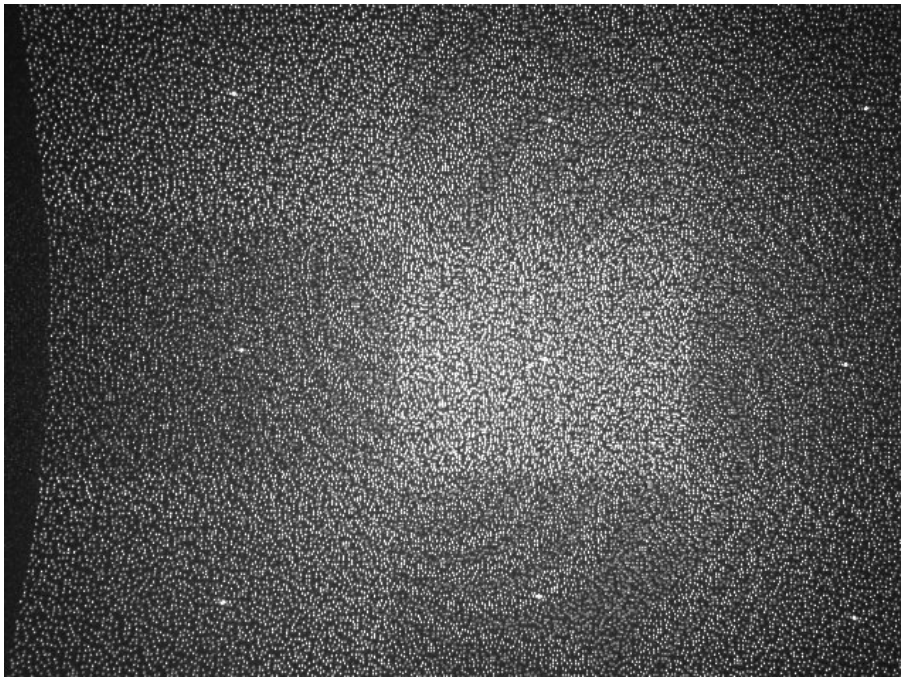


Figure 12 Fixed pattern emited by IR projector

3.1.2 Drivers

Recently (June 16, 2011) official non-commercial Kinect for Windows SDK beta was released but people were eager to explore possibilities that Kinect offers since its first release therefore many unofficial drivers were developed and Kinect was "hacked" many times. Short overview of available Kinect drivers follows.

Libfreenect

Libfreenect is software developed by OpenKinect [30], "an open community of people interested in making use of the amazing Xbox Kinect hardware wit our PCs and other devices". OpenKinect community works for free and develop open source libraries for Windows, Linux and OS X. Code contributed to OpenKinect is available under an Apache20 or GPL2 license. At the moment there are wrappers in Python, C Synchronous, Actionscript, C++, C#, Java JNI, Java JNA, Javascript. Source code of Libfreenect is publicly downloadable through repository [31].

OpenNI

OpenNI (Open Natural Interaction) [32] is a multi-language, cross-platform project that defines APIs for writing applications utilizing Natural Interaction. The main purpose of OpenNI is to form a standard API that allows communication with both vision and audio sensors and vision and audio perception middleware. OpenNI's API also enables middleware developers to write algorithm on top of raw data formats, regardless of which sensor (e.g.

Kinect) has produce them. OpenNI is an open source API that is publicly available [33].

Kinect for Windows SDK beta

The Kinect for Windows SDK beta [34] is programming toolkit for application developers. It enables easy access to the capabilities offered by the Microsoft Kinect device. The only supported operating system is Windows 7 and supported programming languages are C++, C# and Visual Basic by using Microsoft Visual Studio 2010. Implemented features are raw sensor streams, skeletal tracking and advanced audio capabilities with Windows speech recognition API. The current SDK is designed for non-commercial purposes only and commercial version is expected to be available at later date. Kinect for Windows SDK beta is available to free download through its website [34].

There are several libraries built on top of the mentioned software. **Nestk** [35] is library developed by Nicolas Burrus [36] to be easily integrated into existing cmake-based software and provide quick access to the Kinect features. The library is build on top of OpenCV and QT and partly depends on PCL. Both Libfreenect and OpenNI drivers are involved. **ofxKinect** [37] is an OpenFrameworks add-on for the Xbox Kinect that supports Linux and OS X. OpenFrameworks is a cross-platform open source toolkit for creative coding in C++ [38]. **depthJS** allows any web page to interact with the Microsoft Kinect using Javascript [39]. depthJS is developed by a team from the MIT Media Lab and is distributed as open source under the AGPL license [40]. Currently supported web browsers are Safari and Chrome.

3.2 Point Cloud Library (PCL)

PCL is a comprehensive free, BSD licensed, library for n-D Point Clouds and 3D geometry processing [41]. The PCL framework contains numerous state-of-art algorithms including filtering, feature estimation, surface reconstruction, registration, model fitting and segmentation. PCL is cross-platform, and has been successfully compiled and deployed on Linux, MacOS, Windows, and Android. To simplify development, PCL is split into a series of smaller code libraries that can be compiled separately. PCL website contains detailed documentation and tutorials demonstrating key features of library [42]. PCL is under ongoing development and currently PCL version 1.1 has been released (18 July, 2011). It is free for commercial and research use.

Figure 13 Point Cloud Library logo

A **point cloud** is a data structure used to represent a group of **multi-dimensional points** and is used to represent three-dimensional data. In a 3D point cloud, the points usually represent the X, Y, and Z geometric coordinates of an underlying sampled surface. By adding color information to each point we get 4D point cloud.

Point clouds can be acquired from various hardware sensors such as stereo cameras, 3D scanners, or time-of-flight cameras, or generated from a computer program synthetically. Common characteristic of mentioned hardware is high price that limited its massive spread before Kinect like devices were released. PCL supports natively the OpenNI 3D interfaces therefore can acquire and process data from devices such as the Microsoft Kinect or Asus XTionPRO.

3.2.1 Architecture

To simplify development the library consists of series of smaller code libraries, that can be compiled separately.

- **libpcl_filters** - implements data filters such as downsampling, outlier removal, indices extraction, projections, etc.
- **libpcl_features** - implements many 3D features such as surface normals and curvatures, boundary point estimation, PFH and FPFH descriptors, spin images, integral images, NARF descriptors, etc.
- **libpcl_io** - implements I/O operations such as writing to/reading rom PCD (Point Cloud Data) files.
- **libpcl_segmentation** - implements cluster extraction, model fitting via sample consensus methods for variety of parametric models (planes, cylinders, spheres, lines, etc.)
- **libpcl_surface** - implements surface reconstruction techniques, meshing, convex hlls, Moving Least Squares, etc.
- **libpcl_registration** - implements point cloud registration methods such as ICP.
- **libpcl_keypoints** - implements different keypoint extraction methods, that can be used as a preprocessing step to decide where to extract features descriptors
- **libpcl_range_image** - implements support for range images created from point cloud

dataset

- **libpcl_visualization** - implements visualization function for point clouds based on VTK library

3.2.2 3rd Party Libraries

- **Eigen** 3 [43], an open-source template library for linear algebra, is used for most mathematical operations in PCL
- **FLANN** (Fast Library for Approximate Nearest Neighbors) [44] is backbone for fast k-nearest neighbor search operations.
- **Boost** [45] shared pointers are used in all the modules and algorithms in PCL to avoid the need to re-copy data that is already present in the system
- **VTK** (Visualization Toolkit) [46] offers great multi-platform support for rendering 3D point clouds and surface data, including visualization support for tensors, texturing and volumetric methods.
- **CMinPack** [47] is open source library for solving nonlinear equations and nonlinear least squares problems.

3.3 CImg

CImg [48] stands for **Cool Image** and it is easy to use, efficient library that intends to be very pleasant toolbox to design image processing algorithms in C++. It was originally developed by David Tschumperlé during his PhD and later was extended by many contributors. CImg is open source product distributed under two distinct licenses (CeCILL-C and CeCILL).

The CImg library was design with several properties in mind:

- **Usefulness** - CImg defines classes and methods to manage images in your own C++ code. There are methods for loading/saving various file formats, accessing pixel values, displaying/transforming/filtering images, drawing primitives, computing statistics, etc.
- **Genericity** - CImg defines a single image class which can represent datasets having up to 4-dimensions with template pixel types. It also handles image collections and sequences.
- **Portability** - CImg is self-contained and this highly portable. It fully works on different operating systems and is compatible with various C++ compilers.
- **Simplicity** - CImg is lightweight. It is made of a single header file "CImg.h" that must be included in C++ source.
- **Extensibility** - CImg can use functionalities of external tools/libraries such as OpenCV, liftoff, libjpeg, etc. Moreover, a simple plug-in mechanism allows any user to directly enhance the library according to his needs.
- **Freedom** - CImg is free and open-source library

4 Database

This chapter describes database that had been created for purpose of this work. The developed head pose estimation system works with input from the Kinect device that is relatively new equipment. It brings big improvement in possible 3D geometry research but there is still lack of suitable databases for various computer vision fields. This work require database that provides 3D information of observed scene that can be transferred to PCD (Point Cloud Data) format used in PCL. Such database was not available therefore creating one was a key task for the thesis.

Following paragraphs describe the process of recording database, post-processing of images, labelling and all other aspects of database building.

4.1 Description

The database consists of two sequences of images that were taken in standard lightning condition in a room of Image Processing Lab at UPV, Valencia, Spain. Sequences records two different people in the same conditions. Details about parameters of database are mentioned in Table 1. The sequence scenario starts when person is directly facing camera and continues twice turning right, left, up and down. After reaching the limit position the head is returned to starting position.

Table 1 Parameters of built database

	Sequence 1		**Sequence 2**	
Name of person	Goons Mowde		Aleš Pospíšil	
Number of frame	485		462	
Time of sequence (s)	26		24	
Rotations	pitch	yaw	pitch	yaw
Angle range (deg)	-30<x<40	-60<x<60	-30<x<40	-60<x<60

During post-processing and labelling the intention is to automatically receive transformation angles of each individual frame compared to template frame. Therefore face markers are used to make the recognition of a point in the face easier. In total otal 9 markers are place in the face to guarantee enough visible markers in each proposed face rotation. Location of markers is visible on Figure 14. As the markers regular blue post-it tape with a size 2.5 x 1.5 cm was used.

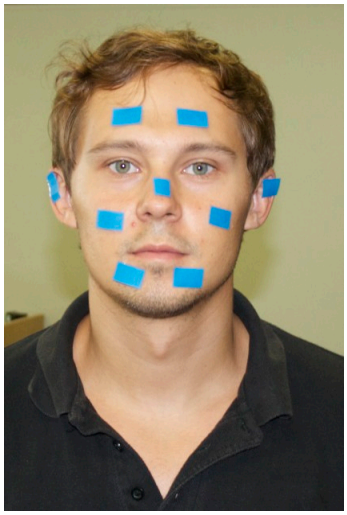


Figure 14 Face with 9 blue markers

Measurement setting is shown in Figure 15 and exact parameters of the settings are following:

- Number of markers on the face: **9**
- Distance between the Kinect device and the ground: **101 cm**
- Distance between the Kinect device and observed person: **77 cm**

Figure 15 Table with Kinect prepared for measurement

Final dataset was used for testing and evaluation of the developed head pose estimation system. Thumbnails of image sequence are placed in Figure 16 and Figure 17.

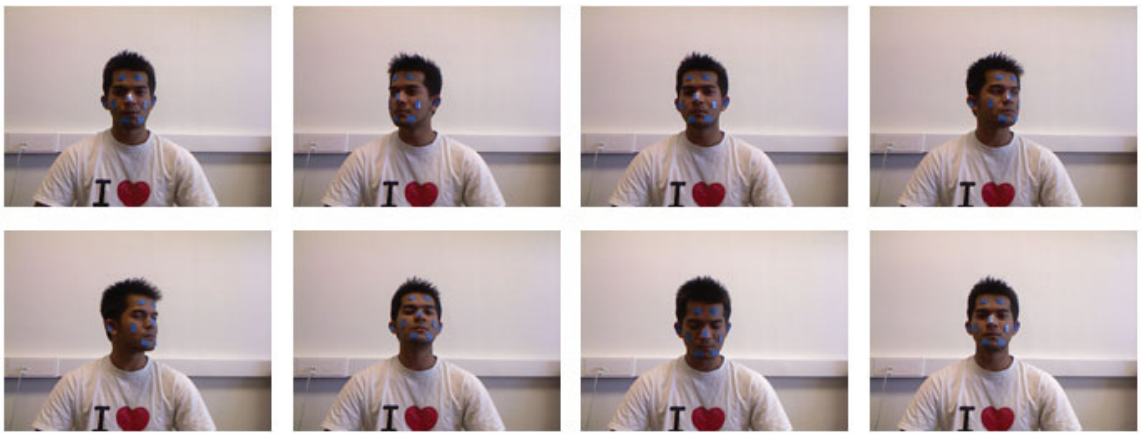


Figure 16 Thumbnails of Sequence 1

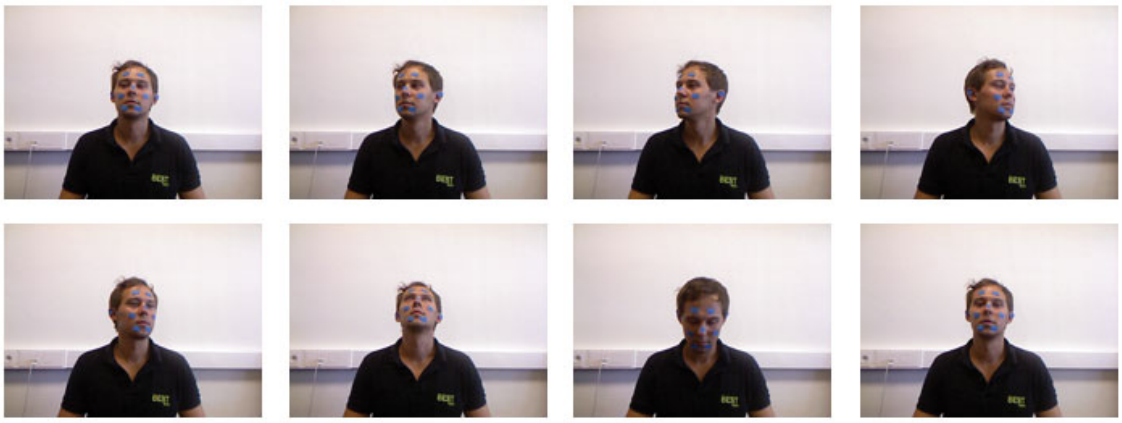


Figure 17 Thumbnails of Sequence 2

4.2 Dataset processing

The purpose of images processing is to extract useful information that is related to marker location and in the final step receive three-dimensional coordinates of each single visible marker in each frame. Image processing was implemented in C++ through the CImg library and its Segmentation Utility extension developed at UPV.

The first step was extraction of markers' (blue) areas from images. Each RGB channel was loaded separately and following equation was applied to emphasize markers areas.

```
image = -0.3*R - 0.6*G + 0.9*B;
```

Fine markers separation was achieved by filtering image by threshold equal to 35 (value of 35 was reached empirically).

Additional information about all segments were acquired via three methods implemented in Segmentation Utility CImg library:

`Segmentbin8_cimg(filtered, seg, numobj);`

- is a method to segment input image `CImg<unsigned char> filtered` and as an output there is `CImg<int> seg` that provide pixel by pixel information about assigned segment of each specific pixel and second output `int numobj` that represents number of recognized segments in the input image

`Momentos_Areas_cimg(seg, centros, covarianzas, n_points, numobj);`

- is a method that calculates from given image segments geometrical center of each segment (`CImg<float> centros`), covariance of each segment (`CImg<float> covarianzas`) and number of pixels in each segment (`CImg<int> n_points`)

`BoundingBox2_cimg(seg, numobj, bbox);`

- is a method that calculates from given image segments bounding box for each single segment (`CImg<int> bbox`)

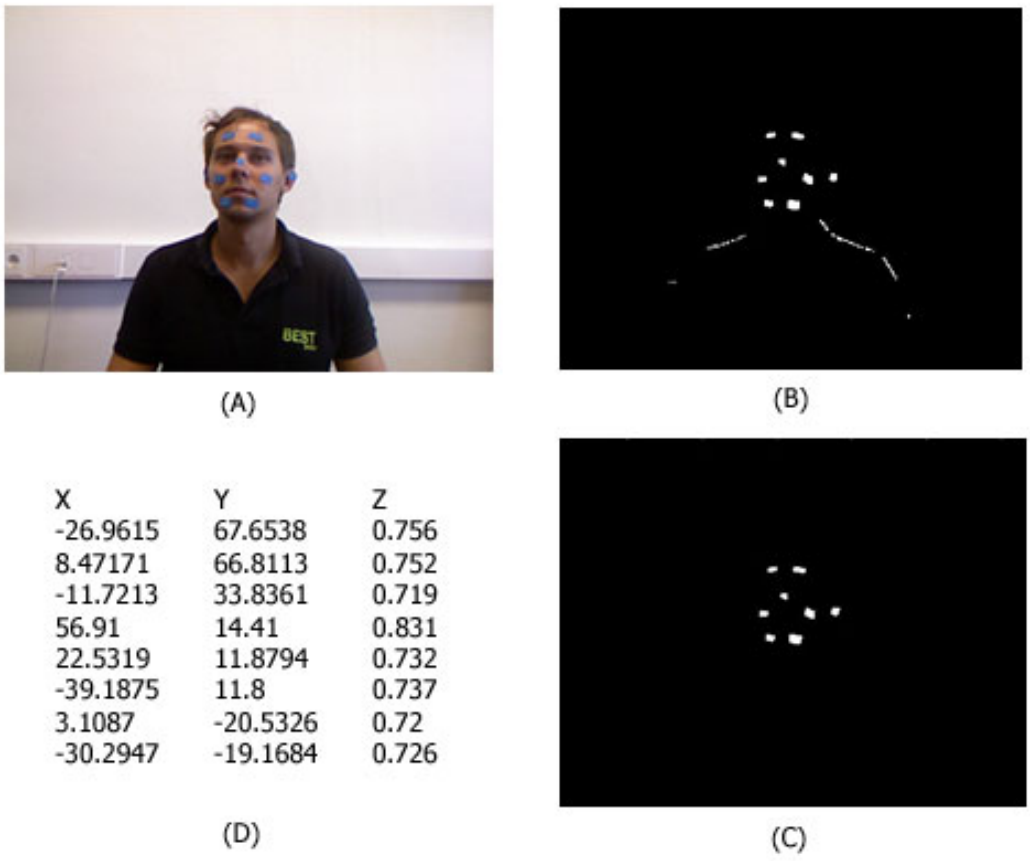


Figure 18 Filtering process: (A) Input image (B) First step of filtering with outliers (C) Precise filtering extract only markers areas (D) 3D coordinates extracted from markers

Filtering process is illustrated in Figure 18 and Figure 19 and is run over each found segment to decide if segment represents marker or doesn't. In the first step segments with number of pixels lower than threshold (used threshold equals 40) are filtered. In the second step segments whose height extracted from bounding box parameters is lower than threshold (used threshold equals 20) are filtered. In the third step segments whose ratio between Eigen values extracted from covariance is lower than threshold (used threshold equals 15) are

filtered. All these filtering steps are required because of imperfect segmentation in original step (from various reasons) and filters are designed to filter segments that do not fit markers' size or space orientation. As a result the centre of segments is drawn into image and 3D coordinates are saved for further operations.

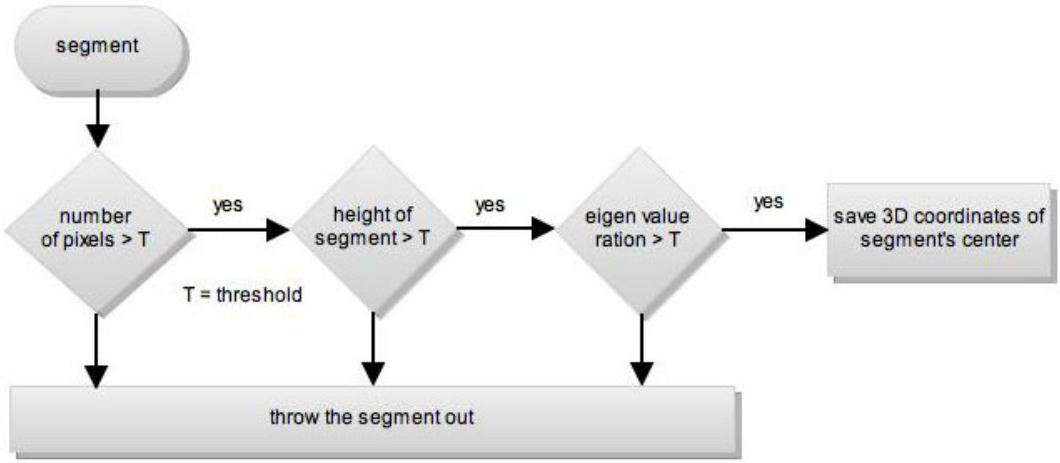


Figure 19 State flow diagram showing all filtering criteria

4.3 Labeling

The purpose of labeling is to extract transformation matrix and Euler angles between specific frame (image) and keyframe. Space coordinates are used to calculate all mentioned parameters and following algorithm is used to fulfil the task. Algorithms were implemented in MATLAB environment. Matching corresponding coordinates and calculating transformation from correspondence of set of points are two challenges that have to be solved in this part of work.

4.3.1 Transformation from correspondences

In previous steps coordinates of markers in each frame of dataset were obtained. The obvious intention is to use these coordinates to calculate transformation between frames. `TransformationFromCorrespondance` method from PCL library was implemented in MATLAB to fulfil this task [41]. Output of this function is 4x4 transformation matrix that represents both rotation and translation between considered frames.

Transformation from correspondence function is based on Singular Value Decomposition (SVD) [49] and its code is presented in `get_transformation.m`. Following code snippet represents how is the transformation calculated.

```
[u, s, v] = svd(covariance);

rotation = u*v.';
translation = mean_out - rotation*mean_in;
```

Rotation matrix (3x3) is calculated by multiplication of unitary matrixes u and v got in from SVD. Translation vector is calculated from mean values of all correspondence points involved in a step. Covariance matrix (3x3) is calculated from `alpha` what is ratio between weight and accumulated weight and `diff_in / diff_out` what represent difference between current point value on input and output and mean values.

Term transformation matrix was used in the previous paragraphs therefore deserves more detailed explanation. The transformation matrix consists of rotation matrix and translation vector.

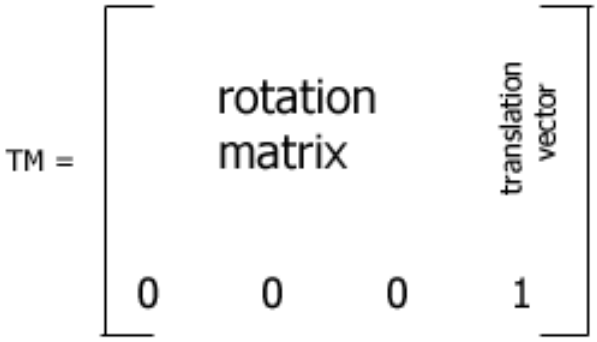


Figure 20 Transformation matrix

The rotation matrix is 3 by 3 matrix that represents vector rotation about x, y or z axis in three dimensions. Basic rotation matrices are shown in Equations 1, 2 and 3.

$$R_x(\theta) = \begin{matrix} 1 & 0 & 0 \\ 0 & cos\theta & -sin\theta \\ 0 & sin\theta & cos\theta \end{matrix} \quad (1)$$

$$R_y(\theta) = \begin{matrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{matrix} \quad (2)$$

$$R_z(\theta) = \begin{matrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{matrix} \quad (3)$$

The global rotation matrix is given by matrix multiplication as described below.

$$R = R_x(\gamma)R_y(\beta)R_z(\alpha) \quad (4)$$

$$R = \begin{matrix} cos\theta cos\varphi & -cos\phi sin\varphi + sin\phi sin\theta cos\varphi & sin\phi sin\varphi + cos\phi sin\theta cos\varphi \\ cos\theta sin\varphi & cos\phi cos\varphi + sin\phi sin\theta sin\varphi & -sin\phi cos\varphi + cos\phi sin\theta sin\varphi \\ -sin\theta & sin\phi cos\theta & cos\phi cos\theta \end{matrix} \quad (5)$$

Euler angles represent other way how interpret space rotation. Euler angles can be calculated from rotation matrix using following MATLAB code snippet where `transformation` represent rotation matrix.

```
roll  = atan2(transformation(3,2), transformation(3,3));
pitch = asin(-transformation(3,1));
yaw   = atan2(transformation(2,1), transformation(1,1));
```

Translation vector is given by vector of three members that represents translation in three dimensions as described bellow.

```
w = (wx, wy, wz);
```

4.3.2 Matching corresponding coordinates

There are two obstacles in applying transformation from correspondence algorithms for retrieve coordinates:

1. Different number of markers is visible/recognized in each individual frame
2. Marker containing segments are recognized in different order in each individual frame.

Optimization algorithms had to be developed to synchronize matrices with coordinates between consecutive frames. First calculations were done to decide which marker in input frame corresponds to which marker in output frame. Basic geometric distance between two tested points together with threshold application solved first issue. If markers were only disappearing in following frames there would be no other issue. However there are cases when markers are appearing after sequence of frames and it is difficult to match them with previous recognized marker. Our intention is to have coordinates of all nine markers in all frames of sequence another method has to be developed.

Knowledge of total number of markers gives advantage that is used in following steps. Figure 21 represent possible scenario of recognized markers (green color). There are two cases in this example, either new marker is appearing or old marker is disappearing. In the first case, if new marker appears, coordinates are saved in new position and using inverse transformation matrix is used to calculate coordinates of this marker in previous frame. Once all nine slots are takes algorithm keeps coordinates updated through the whole calculation process. In the second case, when old marker disappears, transformation matrix is applied in order to retrieve coordinates of the lost marker in the frame.

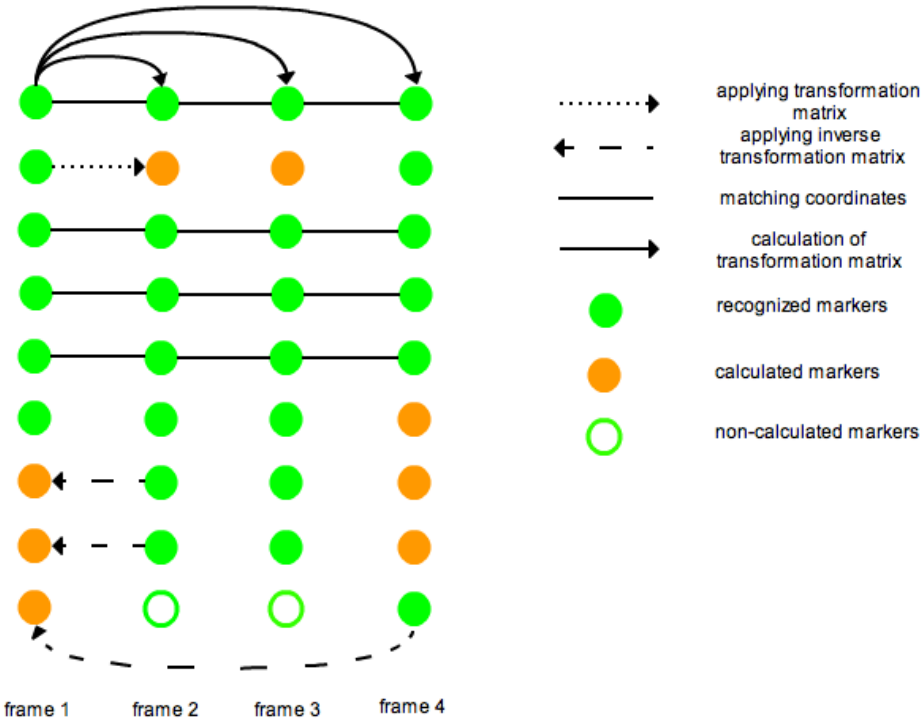


Figure 21 Matching corresponding coordinates

Described operations give opportunity to calculate final transformation matrix and Euler angles between chosen frames without any restrictions. This final step allows us to create database with ground truth to properly evaluate developed head pose estimation system.

5 Developed System

The goal of this work was to develop a system that would be able to estimate head pose and track its position in time. This problem may be approached by various ways as described in Chapter 2. As a problem solution is in this work proposed hybrid method where depth information from the Kinect device and tracking process is involved. The Kinect is low-cost equipment that makes 3D information acquisition achievable without big investment therefore is worth to investigate its possible use in this field. System was developed using C++ language under Eclipse environment. In the most cases Point Cloud Library (PCL) and CImg library is used to implement required algorithms. One of the biggest obstacles that had to be overcome was progress in work on PCL. In fact, the official release of 1.0 version of the library was announced on the 12th of May 2011. Therefore the main goal of chosen approach was not to do develop high performance system but rather explore possible way how combine these technologies to achieve competitive results.

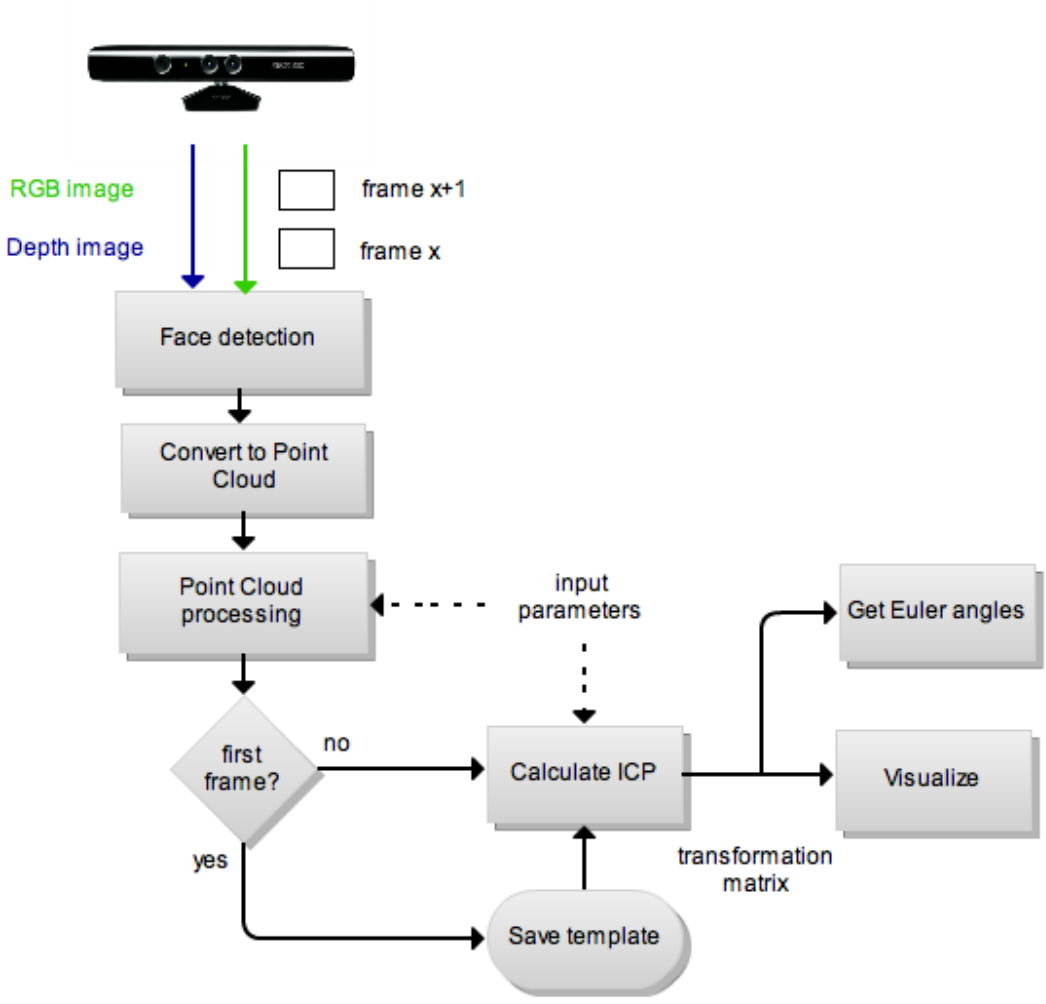


Figure 22 State flow diagram of head pose estimation system

Figure 22 represents the developed system in its high level representation. On the input there is a Kinect device that provides system with both RGB and depth image received in frame-by-frame order. System itself consists of several blocks that are described in detail later in this chapter. Only brief description of each block function follows now.

Face detection is applied on RGB image in order to extract face area from image. Convert to Point Cloud block is used to transfer face area from CImg format images to PCD, Point Cloud format used in PCL. Point Cloud processing block include primarily functions to filter uninteresting parts of Point Cloud or functions to reduce size of Point Cloud. Input is processed frame-by-frame so loop over all frames is implemented and few conditions are used. Point Cloud of face in first frame is saved as a template and is used for further calculations. Iterative Closest Point (ICP) is registration method implemented in PCL that was used to register chosen Point Clouds in order to get transformation matrix out of it. Point cloud in current frame is registered against template point cloud. In the final step visualization block and Euler angles calculation block are implemented.

5.1 Face detection

Face detection algorithm was not developed from scratch for the purpose of this work but algorithm developed by Alberto Albiol [54] based on Viola-Jones object detection framework [55] was used. This algorithm implements additional face features detection such as eyes, mouth and nose but these were not used in the work.

Face detection is implemented in this work by including `FaceObjectDetection` class and following code snippet represents the main operation related to face detection.

```
//Create object faceObjectDetector
FaceObjectDetector faceObjectDetector;
//Apply faceObjectDetector on image
faceObjectDetector.process(image);
//Save detected object into the object list - faceObjects
faceObjects=faceObjectDetector.getObjects();
//Draw detected objects into the image
faceObjects.draw(image_orig);
```

Each detected face is described by its bounding box. Bounding box is represented by X and Y coordinates of top left corner and its width and height. This information is used to crop the face area and convert to PCD later on.

Face detection block however brought into this work some obstacles and limitations. Face detection was optimized primarily for frontal view what complicated the detection in extreme limits. Detection of roll rotation of the head was under performance of pitch and yaw rotations. Even though these aspects limit the current performance of developed system, Face

detection block can be upgraded, modified or exchanged, if needed. Most importantly the performance of current system is still sufficient.

5.2 Conversion to Point Cloud

RGB and depth images are recorded using CImg library and later converted from CImg format to PCD format. PCL includes handler for communication with Kinect but there are no properties for recording and repeated loading of recorded sequences. It is most probable that in next version of PCL there will be easy solution for this problem but there wasn't in a time when this work was in progress. Class `CImg2PointCloud` was developed to provide easy transition between these two formats. The code snippet illustrates implemented conversion.

```
for(unsigned int x= 0; x< camDepth.width(); x += _step)
{
 float xcam = x - centerx;
 for(unsigned int y= 0; y< camDepth.height(); y += _step, ++depth_idx)
 {
       float ycam = alt_1_centery - y;
       float Zcam = camDepth(x,y)/1000.0;

       pcl::PointXYZ& pt = cloud->points[depth_idx];

       if(Zcam == 0)
       {
             pt.x = pt.y = pt.z = bad_point;
       }

       pt.x = xcam * iFD * Zcam; // from PCL
       pt.y = ycam * iFD * Zcam;
       pt.z = Zcam;

 }
 }
```

`Zcam`, `xcam` and `ycam` represent coordinates points in space. `iFD` represent inverse focal distance that is for Kinect 525 mm. `Centerx` and `Centery` represent center of image in both `x` and `y` axis. `Alt-1-centery` represents correction in `y` axis given by different horizontal orientation in formats.

5.3 Point Cloud processing

Point Cloud processing block provides two main functions. Firstly it decreases size of processed point cloud by downsampling and secondly it filters points that do not fulfill set criteria (primarily distance from camera). PCL provides filters module from which classes `pcl::VoxelGrid and pcl::PassThrough` are used. VoxelGrid class assembles a

local 3D grid over a given Point Cloud, downsamples and filters the data. The code snippet from VoxelGrid implementation in this work is shown below.

```
const float voxel_grid = 0.005;
pcl::VoxelGrid<pcl::PointXYZ> vox_grid;
vox_grid.setInputCloud (cloud_filtered);
vox_grid.setLeafSize (voxel_grid, voxel_grid, voxel_grid);
vox_grid.filter (*cloud_down);
```

As an input to `setLeafSize`() the size of filtering leaf is given. In this work regular leaf of common size 0.005 is chosen.

PassThrough class passes through all data that satisfies the user given constraints. Intention of this work was to get rid of points that are in background and by that extract only head points. This task is achieved by filtering point cloud according to their Z coordinate and the code snippet of PassThrough implementation for this work is shown below.

```
pcl::PassThrough<pcl::PointXYZ> pass;
pass.setInputCloud (cloud);
pass.setFilterFieldName ("z");
pass.setFilterLimits (0.0, (Z + 0.10) );
pass.filter (*cloud_filtered);
```

By setting `setFilterFieldName`() we choose according to which parameter of PCD structure filtering is done. Min and max limits are chosen by setting `setFilterLimits`() where Z represents the distance between the camera and the closest point of current Point Cloud that is got by `getClosestPoint()` function.

5.4 Iterative Closest Point (ICP)

Registration is fundamental task in image processing used to match two or more pictures taken in different time, taken from different sensors, taken from different viewpoints or under other circumstances. PCL implements three main registration algorithms in its registration module: Iterative Closest Point [49], PPF Registration and Sample Consensus Initial Alignment. Various papers propose ICP as a suitable registration algorithm for head pose estimation therefore ICP is implemented in this work [50][51][52].

Iterative Closest Point is widely used for registering the outputs of 3D scanners, which typically only scan an object from one direction at a time. ICP starts with two meshes (point clouds) and an initial guess for their relative rigid-body transform, and iteratively refines the transform by repeatedly generating pairs of corresponding points on the meshes and minimizing an error metric [49]. ICP implemented in PCL use Singular Value Decomposition (SVD) to estimate rigid transformation [53]. The algorithm has several termination criteria:

1. Number of iterations has reached the maximum imposed by user using (`setMaximumIterations`)
2. The epsilon (difference) between the previous transformation and the current estimated transformation is smaller than an user imposed value (`setTransformationEpsilon`)
3. The sum of Euclidean squared errors is smaller than a user defined threshold (`setEuclideanFitnessEpsilon`).

The code snippet of ICP implementation in this work is shown below.

```
pcl::IterativeClosestPoint<PointXYZ, PointXYZ> reg;
int maxIter = 5;
float epsilonTransformation = 0.001;

// Set the maximum number of iterations
reg.setMaximumIterations (maxIter);
// Set the transformation epsilon
reg.setTransformationEpsilon(epsilonTransformation);

// Set the input source and target
reg.setInputCloud(cloud_prev);
reg.setInputTarget(cloud);

// Perform the alignment
reg.align(*output);

// Obtain the transformation that aligned source to target
Eigen::Matrix4f transform = reg.getFinalTransformation();
```

Values for max iteration parameter and epsilon transformation parameter were chosen to balance both speed and precision of computation. Five iterations were sufficient due to limited number of points in source and target Point Cloud. Transformation matrix that is obtained in the final step represents transformation that aligned input Point Cloud to target Point Cloud.

Important information that is calculated from transformation matrix is Euler angles. For purpose of this calculation PCL function `pcl::getEulerAngles` was modified to correspond input matrices. In the measurement stage Euler Angles for each frame were recorded.

5.5 Visualization

Visualization part is important for immediate observation and evaluation of implemented algorithms. There are two different visualization techniques used in this work. First is based

on visualization module of PCL where `pcl::visualization::PCLVisualizer` class is implemented. Second technique projects each point of Point Cloud to CImg coordinates and draws points into RGB image using developed method `drawCloud()`.

`PCLVisualizer` class provides many functions to manipulate Point Cloud and modify the way how is visualize. The following code snippet represents `PCLVisualizer` implementation in this work.

```
// New PCLVisualizer object viewer is called
pcl::visualization::PCLVisualizer viewer("Head Pose");

//Color handler which set color of point cloud to RED
pcl::visualization::PointCloudColorHandlerCustom<pcl::PointXYZ>
color(cloud, 255, 0, 0);

// it's safe to remove a non-existent cloud
viewer.removePointCloud("face");

// adding point cloud to scene using color handler
viewer.addPointCloud(cloud_out, color, "face");
viewer.spinOnce();
```

`drawCloud(cloud, image, offset_x, offset_y)` requires on the input 4 parameters, where `cloud` represents Point Cloud that is visualized, `image` represents RGB image into which `cloud` is drawn, `offset_x` and `offset_y` represent offset between `image` centre and centre of the face area obtained in the face detection block. Inverse process to Conversion to Point Cloud block was applied to reach points in CImg format and coordinates. CImg function `draw_point(x_cam,y_cam,color)` was used to draw points into the image. `x_cam` and `y_cam` stand for X and Y coordinates of specific point and `color` defines color of a drawn point.

PCLVisualizer visualization is shown in Figure 23 and drawCloud visualization is shown in Figure 24.

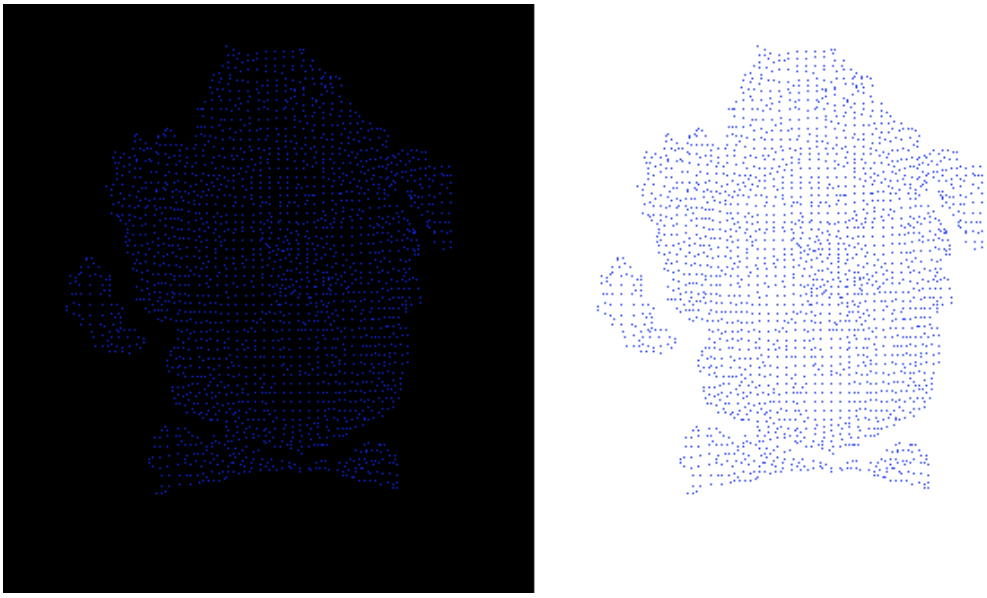


Figure 23 Visualization by PCLVisualizer class

Figure 24 Visualization by drawCloud function

5.6 System requirements and assumptions

There are several requirements and assumptions that have to be considered while performing or testing the system. Some requirements are derived from hardware specifications and some assumption has to be respected because of software implementation.

Chapter 3 describes technical aspects of Kinect device but performance limitation was not described in detail. Kinect device is design to work properly only in indoor environment what

is given by the usage of IR projector. When Kinect would be used on a daylight IR beam would not reach its source and then IR camera couldn't recognize required pattern.

Software implementation assumes several conditions that have to be fulfilled. Firstly the head has to be the first object in the image. Function for recognition of the closest pixel from the camera is used for filtering therefore no objects in front of head is allowed to keep performance of the system. System depends on its face detection block that brings additional limitations. Face detector is trained to recognize frontal face therefore system itself can recognize rotations in limited angle range.

6 Measurements and Evaluation

This chapter describes measurements of head pose estimation system done using created databases. Results of measurements are presented in graphical way and discussion over these results is provided as well. Performance analysis together with possible further improvements concludes this chapter.

6.1 Euclidean Fitness Score

Parameter that represents quick information about quality of matching during registration process is Fitness Score function. PCL implements in its `pcl::Registration` class `getFitnessScore` function that obtain the Euclidean Fitness Score. Generally speaking Fitness Score function describes how close a given design solution is to achieving the set aims. In our case the aim is perfect match between source and target Point Cloud so the Fitness Score function represents sum of squared distances from the source to the target [41].

Fitness function was recorded for both databases and for two different values of voxel grid leaf (described in Chapter 5, Point Cloud processing). Used voxel grid leaf values are 0.01 and 0.005. Results of Fitness score measurement are shown in Figure 25 and Figure 26.

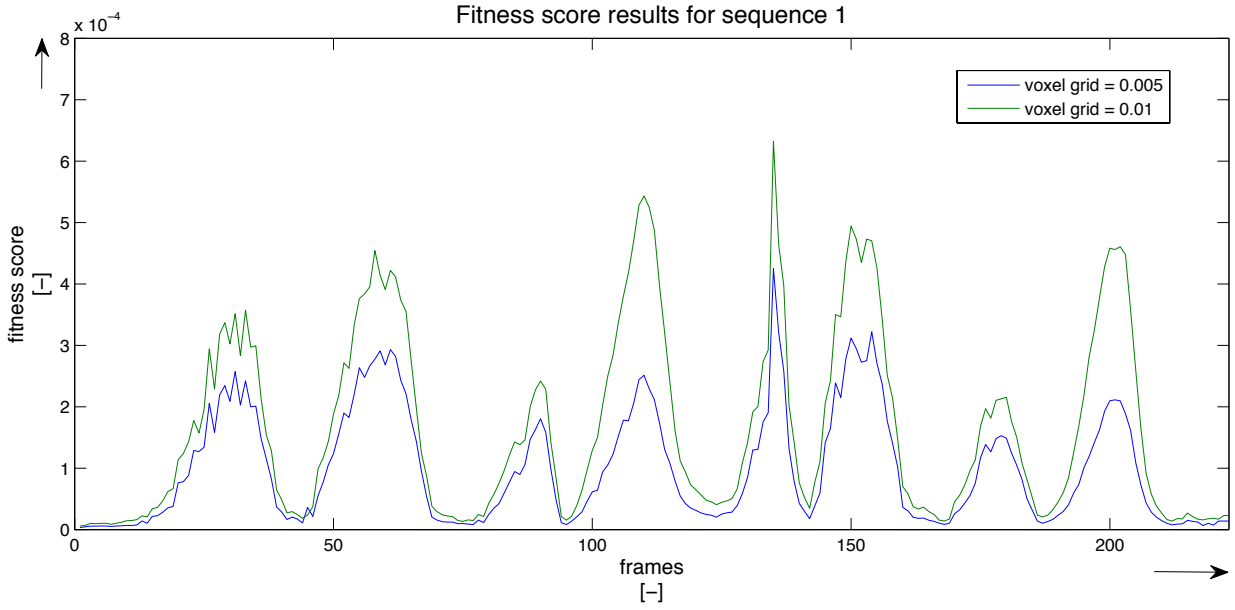


Figure 25 Fitness score results for sequence 1

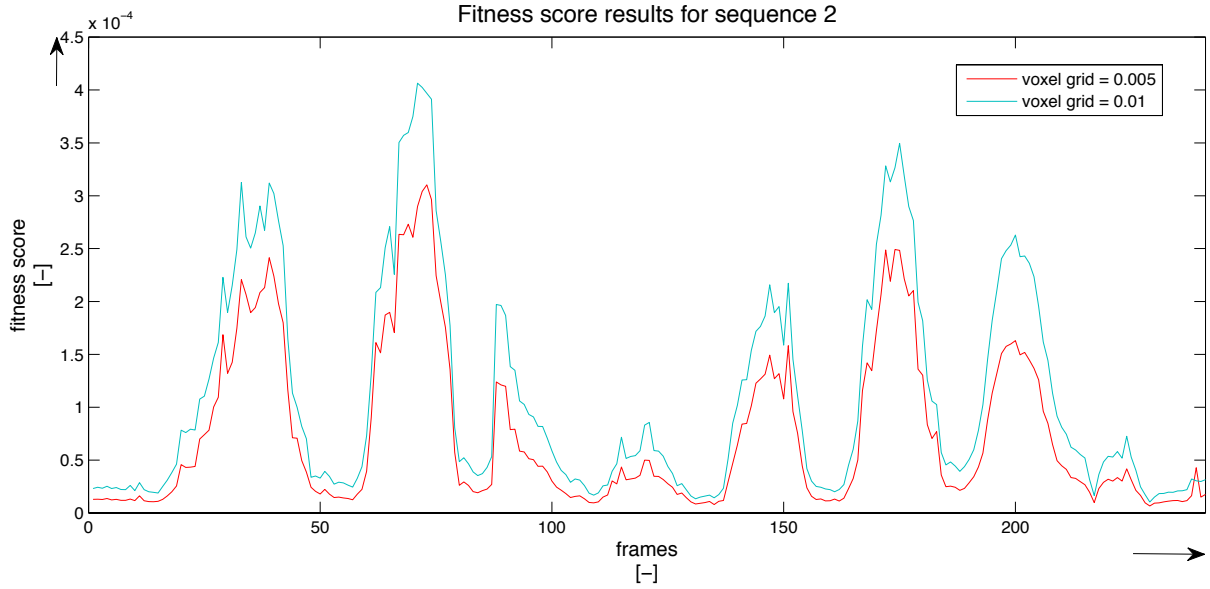


Figure 26 Fitness score results for sequence 2

Fitness score function gives following limit values that are recorded in Table 2.

Table 2 Maximal and minimal fitness score values for different voxel grid parameters

	voxel grid = 0.005		voxel grid = 0.01	
	min	max	min	max
Sequence 1	3.4114e-06	4.2504e-04	5.7726e-06	6.3197e-04
Sequence 2	6.5892e-06	3.1030e-04	1.0342e-05	4.0650e-04

Both graphs and tables indicate that size of voxel grid leaf influence the process of Point Cloud alignment and that higher voxel grid leaf (higher down sampling) implicate lower quality of registration alignment and vice versa. Speed and quality of algorithm are crucial aspects for algorithm design and development therefore proper voxel grid filter setting is important task.

Knowledge of sequence's scenarios gives opportunity to make additional conclusions. Peaks in both graphs correspond to limit head rotations and troughs represent return back to frontal position. First two peaks show fitness score for yaw rotation and second two show fitness score for pitch rotation. When comparing results for different rotation graph analysis reveals that Point Cloud matching during pitch rotation achieves better results. On other hand it is clearly visible that records in database offer higher range of rotation angles for yaw than for pitch.

6.2 Euler Angles

Euler angles acquisition is discussed in the Chapter 4 where database building is explained. Important outcome of the Head Pose Estimation system is rotation represented by Euler angles. There are 3 rotations in total, however the database is build to evaluate only yaw and pitch rotation so roll rotation is omitted. Euler angles in radians are shown in Figure 27 and Figure 28. Euler angles converted to degrees and displayed separately for each sequence and each rotation are shown in Figure 29. Deviation between system output and database ground truth is shown in Figure 30.

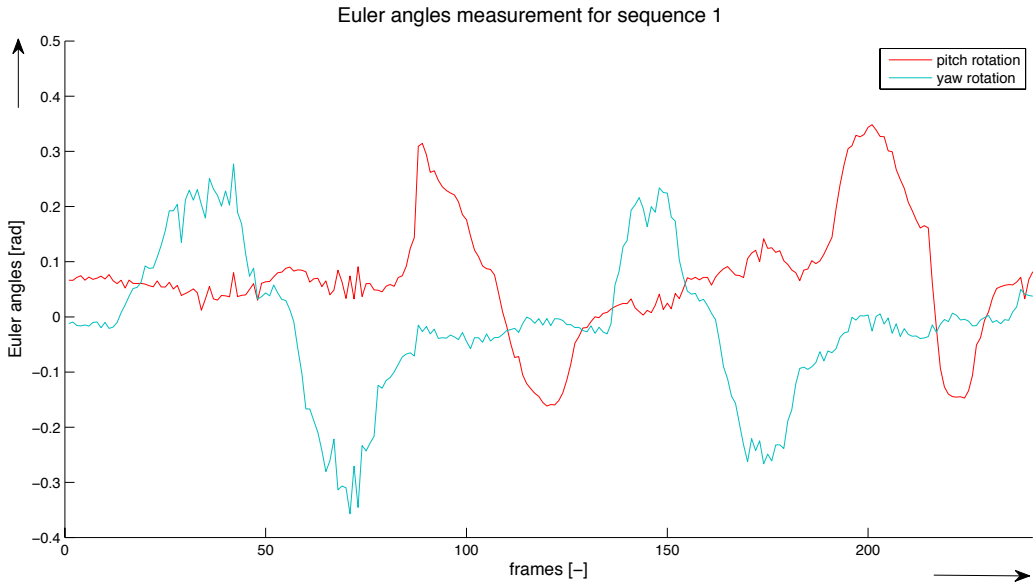


Figure 27 Euler angles measurement for sequence 1

Euler angles converted to degrees can be observed in Figure 29. Comparison between sequences shows nearly the same result for yaw rotation but pitch rotation shows considerable differences. Sequence 1 results range from – 30° to 50 and Sequence 2 range from -100° to 40°. One possible reason for such diverse result is inaccurate database record in which required angle range was not fulfilled. Second issue can be found in Figure 30 (b) where deviation for pitch rotation of Sequence 1 shows highest values what is further described in next paragraph.

As Figure 30 shows deviation between system output and database ground truth some additional information may be extracted from the data. Interesting information is the percentage of cases when deviation is higher than 5 degrees or 10 degrees.

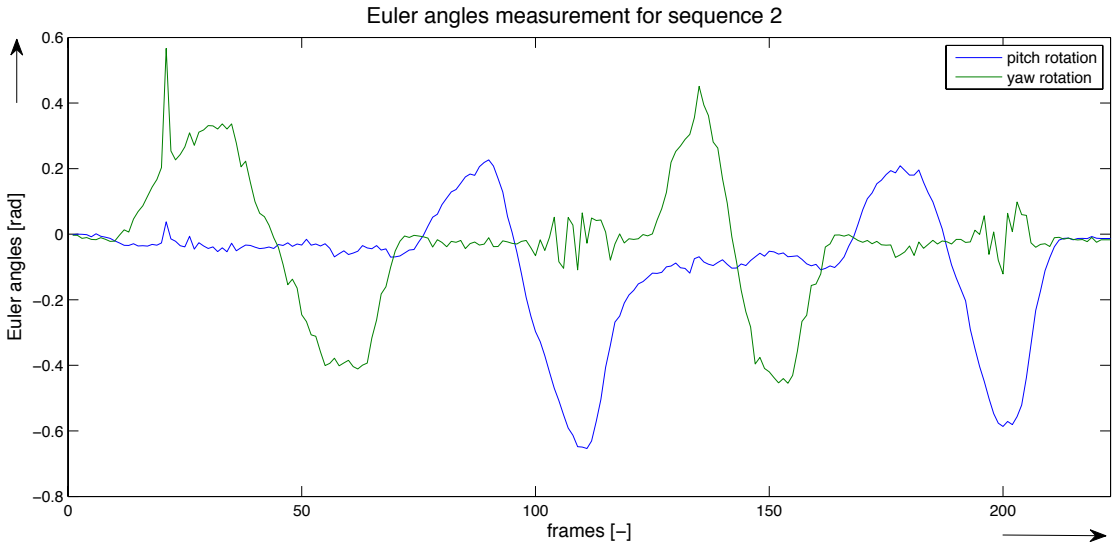


Figure 28 Euler angles measurement for sequences 2

Cases with deviation higher than 5 degrees:

- Sequence 2 – pitch: 13,4 %
- Sequence 2 – yaw: 3,6%
- Sequence 1 – pitch: 9,1%
- Sequence 1 – yaw: 11,2%

Cases with deviation higher than 10 degrees:

- Sequence 2 – pitch: 3,2 %
- Sequence 2 – yaw: 0,45%
- Sequence 1 – pitch: 0,4%
- Sequence 1 – yaw: 0 %

Pitch rotation of Sequence 2 clearly underperform the rest of cases what is obviously one of the reason for problems described above.

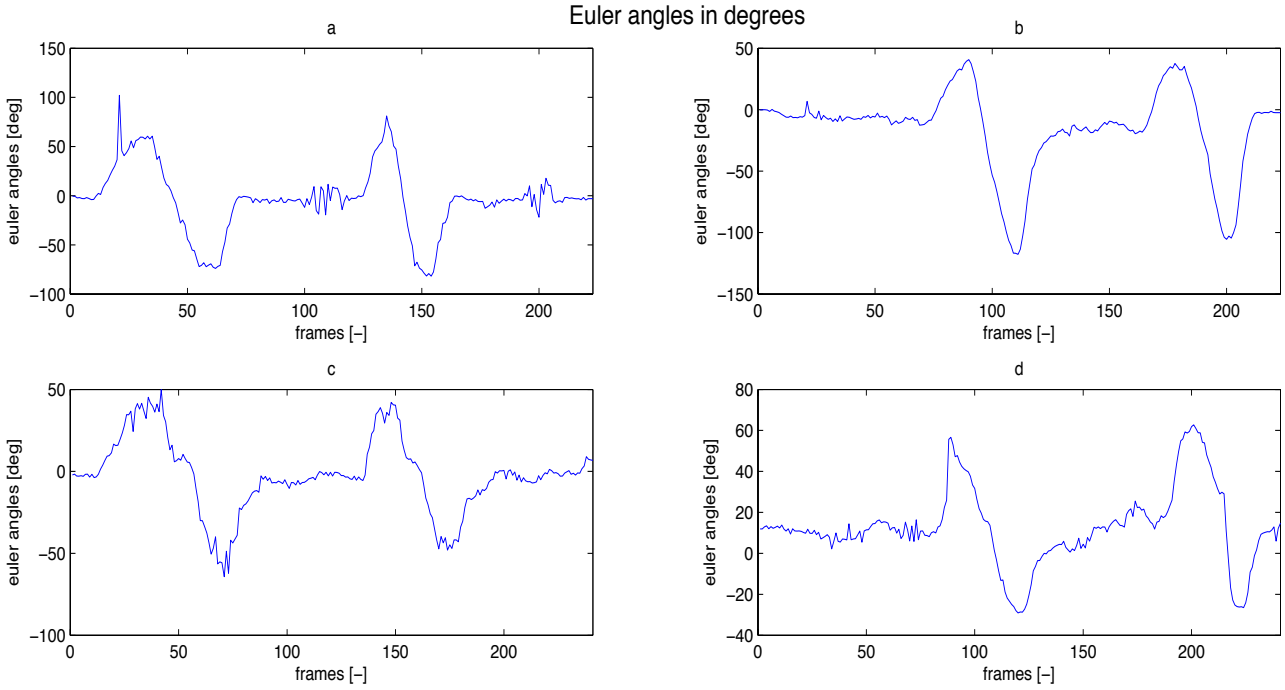


Figure 29 Euler angles in degrees

6.3 Results and discussion

Two main parameters were measured by system to describe its performance. Euclidean Fitness Score shows how well registration matching was done and Euler angles comparison shows how well rotation angles were calculated. There were no speed characteristics measured in testing process however subjective evaluation will be provided to describe this issue.

Euclidean Fitness Score was measured for two values of Voxel Grid Leaf and results are shown in Figure 25, Figure 26 and Table 2. Maximal measured value reached `6,3197e-04` and minimal measured value reached `3,4114e-06`. System using Voxel Grid Leaf equals `0,005` over perform system using the second value. Average difference between these two cases is 51%.

Euler angles were measured for both sequences and different measurement results are shown in Figure 27, Figure 28 and Figure 29. Deviation between angles captured by developed system and grand truth angles from database is shown in Figure 30. In average only 9,3% frames varies from truth data more than 5 degrees and only 1,1% frames varies from truth data by more than 10 degrees. The worst results were achieved for pitch rotation of Sequence 2.

System performance is important parameter for all systems that aspirating to be applied on real time data. System developed in this work didn't reached real time performance however there are parameters that influence speed of algorithm. The value of Voxel Grid Leaf was already discussed but it is crucial parameter that influences performance since it increases or decreases number of processed points. Another parameter is number of iteration in ICP algorithm. This value needs to be rather low in real time applications, for developed system values around 5 were used. Additional improvements that can positively affect the speed performance will be explained in next paragraphs.

6.4 Further improvement

There are several further improvements that are worth describing. System improvements are related to speed or accuracy of algorithm and are based on current block modification or combination with additional techniques.

Convert to Point Cloud block in the system is expensive in the matter of time and computational power because it requires loading, recalculating and saving of every single point. Possible improvement may be optimisation of currently used algorithm by using parallel programming or the data from Kinect may be stored directly in Point Cloud Data format what requires some development on the side of Point Cloud Library.

Face detection block limits the performance and accuracy of the system because its focus on frontal face detection. Face detection algorithm may be improved by adding trained face data with left and right face profile orientation. Other option may be including of other important face marks detection (nose, mouth, eyes). Next improvement option may be use of face detection block just for key frames and for the rest frames use frame-to-frame tracking.

Combination with another algorithms may lead to performance improvement. There are two approaches that are being developed in the Image Processing Lab at UPV. First approach focus on face corners detection and its tracking over the time. This approach limits points that have to be calculated and creates faster tracking algorithm. Second approach detects nose tips as highly important face mark [56][57]. This approach may improve detection and tracking mainly in limit head positions where nose still remains recognisable.

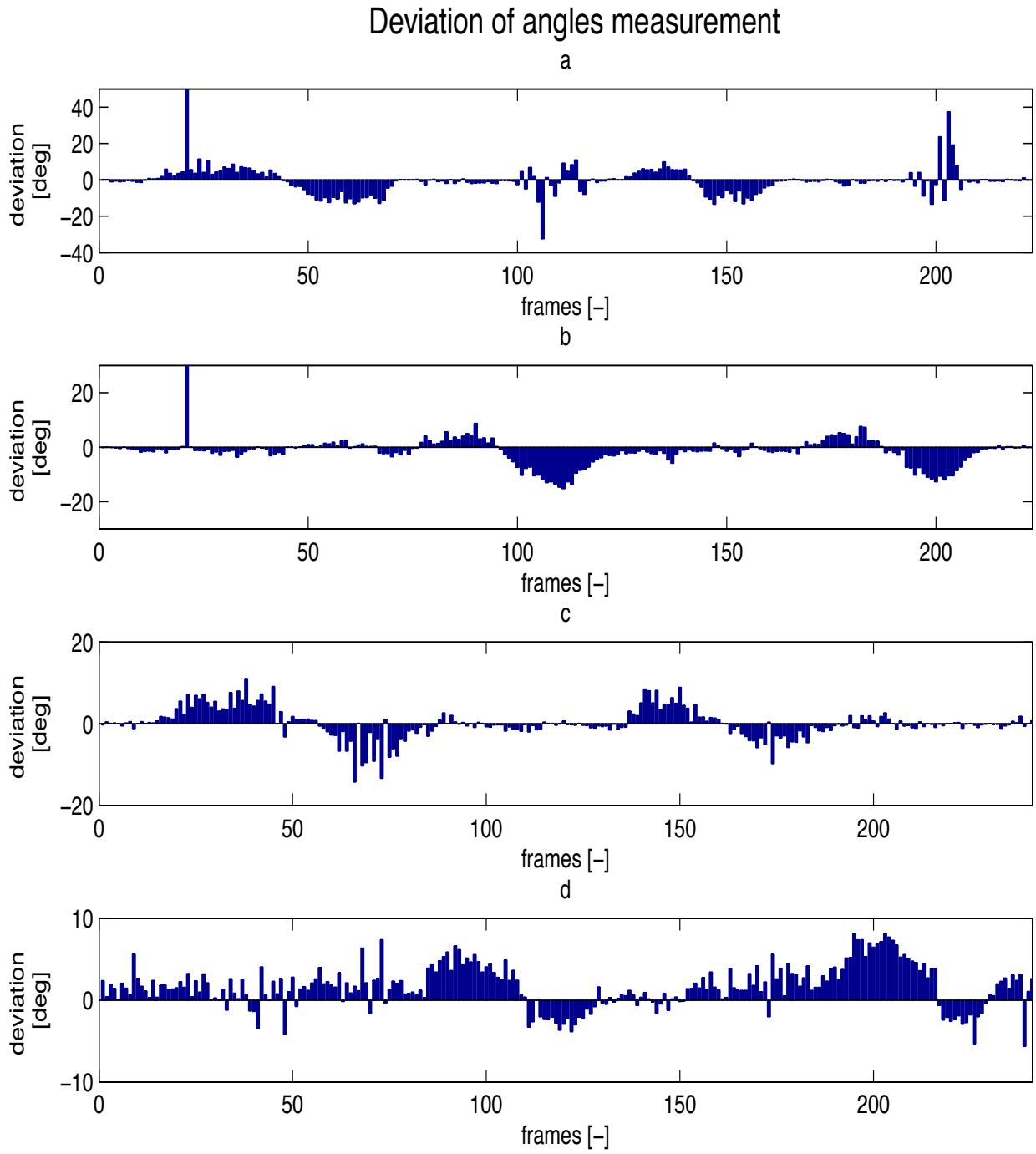


Figure 30 Deviation of angles measurement

7 Conclusions

This thesis provides view into the field of head pose estimation starting with analysis of state of art algorithms, continuing with used technology description and 3D database building and ending with explanation of developed head pose estimation and tracking system and its evaluation.

Chapter 1 opens the thesis with short overview of the topic and goal and structure of this work. Chapter 2 presents different approaches for solving head pose estimation problem. Eight methods are presented and evaluated regarding their implementation's advantages and disadvantages. Chapter 3 describes technologies that were used to complete this work. Novel technical devices and software products such as Kinect, Point Cloud Library and CImg are explained. Chapter 4 talks about 3D database of data acquired by Kinect that was built for purpose of this work. All steps in building of database are described and illustrated by taken pictures. Chapter 5 presents developed head pose estimation and tracking system itself. Complex description of developed head estimation algorithm as well as detailed explanation of every block of algorithm is provided. Chapter 6 explain all measurements that were taken and discuss and evaluate results and outcomes. Further improvement ideas are proposed in the end.

Head pose estimation and tracking system that was developed in this work represents building block on top of which upper layer can be placed. Head pose estimation system is based on 3D data acquired by Kinect device that are processed through PCL and CImg API's. Further development in these technologies may improve quality of the system and increase its performance. Currently developed system can be used for pitch rotations ranging from -30° to 40° and for yaw rotation ranging from -60° to 60°. The system doesn't perform in the real time but improvements that can change it are proposed. Thesis fulfilled its goal and generated outcomes respect goals that were set in the beginning.

Information about head pose is useful for many human-machine interaction applications or human behaviour analysis applications. Human-machine interaction goes nowadays far beyond classical input devices such as computer mouse or keyboard. A novel device such as Kinect brings revolution in the field of human-machine interaction. Applications can be built to interact with its users by recognizing their gestures, face expressions or head poses. Interactive advertisement, augmented reality, robotics are only few fields where knowledge of head pose provides additional value. Machines will not wait for our instructions any more but rather analyse our behaviour and act accordingly. I believe that head pose estimation and tracking is important field to study and its better knowledge and application will improve our daily lives.

Bibliography

[1] E. Murphy – Chutorian, M.M. Trivedi, „Head Pose Estimation in Computer Vision: A Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 31, No. 4. (25 April 2009), pp. 607-626.

[2] E. Murphy – Chutorian, M.M. Trivedi, „Head Pose Estimation and Augmented Reality Tracking: An Integrated System and Evaluation for Monitoring Driver Awareness," *IEEE Transactions on Intelligent Transportation Systems*, Vol. 11, No. 2. (2 June 2010), pp. 607-626.

[3] J. Přinosil, „Přesné sledování pohybů tváří v reálném čase," *Elektrorevue* [online], 2009.

[4] S. Theodoridis, A. Pikrakis, K. Koutroumbas, D. Cavouras, „Introduction to Pattern Recognition: A MATLAB Approach", Academic Press, 2009.

[5] OpenCV Wiki [online]. 2006 [cit 2010-12-13]. Available at <http://opencv.willowgarage.com/wiki/>.

[6] N. Gourier, D. Hall, and J. Crowley, "Estimating Face Orientation from Robust Detection of Salient Facial Structures," *Proc. ICPR Workshop Visual Observation of Deictic Gestures*, pp. 17-25, 2004.

[7] M. Voit, CLEAR 2007 Evaluation Plan: Head Pose Estimation, http://isl.ira.uka.de/~mvoit/clear07/CLEAR07_HEADPOSE_2007-03-26.doc, 2007.

[8] S. Ba and J.-M. Odobez, "A Probabilistic Framework for Joint Head Tracking and Pose Estimation," *Proc. 17th Int'l Conf. Pattern Recognition*, pp. 264-267, 2004.

[9] T. Sim, S. Baker, and M. Bsat, "The CMU Pose, Illumination, and Expression Database," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 25, no. 12, pp. 1615-1618, Dec. 2003.

[10] J. Wu, M. Trivedi, "A Two-Stage Head Pose Estimation Framework and Evaluation," *Pattern Recognition*, vol. 41, no. 3, pp. 1138-1158, 2008.

[11] M.L. Cascia, S. Sclaroff, and V. Athitsos, "Fast, Reliable Head Tracking Under Varying Illumination: An Approach Based on Registration of Texture-Mapped 3D Models," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 22, no. 4, pp. 322-336, Apr. 2000.

[12] W. Gao, B. Cao, S. Shan, X. Zhang, and D. Zhou, "The CAS-PEAL Large-Scale Chinese Face Database and Baseline Evaluations," *Technical Report JDL-TR-04-FR-001*, Joint Research and Development Laboratory, 2004.

[13] D. Little, S. Krishna, J. Black, and S. Panchanathan, "A Methodology for Evaluating Robustness of Face Recognition Algorithms with Respect to Variations in Pose Angle and Illumination Angle," *Proc. IEEE Int'l Conf. Acoustics, Speech, and Signal Processing*, vol. 2, pp. 89-92, 2005.

[14] S. Niyogi and W. Freeman, "Example-Based Head Tracking," Proc. *IEEE Int'l Conf. Automatic Face and Gesture Recognition*, pp. 374-378, 1996.

[15] J. Huang, X. Shao, and H. Wechsler, "Face Pose Discrimination Using Support Vector Machines (SVM)," *Proc. 14th Int'l Conf. Pattern Recognition*, pp. 154-156, 1998.

[16] H. Moon and M. Miller, "Estimating Facial Pose from a Sparse Representation," *Proc. IEEE Int'l Conf. Image Processing*, pp. 75-78, 2004.

[17] J. Sherrah, S. Gong, and E.-J. Ong, "Face Distributions in Similarity Space under Varying Head Pose," *Image and Vision Computing*, vol. 19, no. 12, pp. 807-819, 2001.

[18] A. Lanitis, C. Taylor, and T. Cootes, "Automatic Interpretation of Human Faces and Hand Gestures Using Flexible Models," *Proc. IEEE Int'l Conf. Automatic Face and Gesture Recognition*, pp. 98-103, 1995.

[19] A. Gee and R. Cipolla, "Determining the Gaze of Faces in Images," *Image and Vision Computing*, vol. 12, no. 10, pp. 639-647, 1994.

[20] A. Gee and R. Cipolla, "Fast Visual Tracking by Temporal Consensus," *Image and Vision Computing*, vol. 14, no. 2, pp. 105-114, 1996.

[21] E. Murphy-Chutorian and M. Trivedi, "Hybrid Head Orientation and Position Estimation (HyHOPE): A System and Evaluation for Driver Support," *Proc. IEEE Intelligent Vehicles Symp*., 2008.

[22] Kinect gets UK release date [online]. 2010 [2011-08-17]. Available at <http://www.bbc.co.uk/newsbeat/10996389>.

[23] Kinect Confirmed As Fastest-Selling Consumer Electronics Device [online]. 2011 [2011-08-17]. Available at <http://community.guinnessworldrecords.com/_Kinect-Confirmed-As-Fastest-Selling-Consumer-Electronics-Device/blog/3376939/7691.html>.

[24] Microsoft sells 10 million Kinects, 10 million Kinect games [online]. 2011 [2011-08-17]. Available at <http://www.engadget.com/2011/03/09/microsoft-sells-10-million-kinects-10-million-kinect-games/>.

[25] Kinect Sensor with Kinect Adventures [online]. 2010 [cit 2011-08-17]. Available at <http://www.amazon.com>

[26] Kinect Hacks [online]. 2010 [cit 2011-08-17]. Available at <http://www.kinecthacks.com/>.

[27] Kinect Hacks [online]. 2011 [cit 2011-08-17]. Available at <http://kinect.dashhacks.com/>.

[28] Kinect Operation [online]. 2010 [cit 2011-08-17]. Available at <http://www.ros.org/wiki/kinect_calibration/technical>.

[29] PrimeSense [online]. 2010 [cit 2011-08-17]. Available at <http://www.primesense.com/>.

[30] OpenKinect Wiki [online]. 2010 [cit 2011-08-17]. Available at <http://www.openkinect.org>.

[31] OpenKinect/libfreenect at Github [online]. 2010 [cit 2011-08-17]. Available at <https://github.com/OpenKinect/libfreenect>.

[32] OpenNI User Guide [online]. 2011 [cit 2011-08-17]. Available at <http://www.openni.org/documentation>.

[33] Introducing OpenNI [online]. 2011 [cit 2011-08-17]. Available at <http://www.openni.org/>.

[34] Kinect for Windows SDK from Microsoft Research [online]. 2011 [cit 2011-08-17]. Available at <http://research.microsoft.com/en-us/um/redmond/projects/kinectsdk/>.

[35] Using the nestk library [online]. 2011 [cit 2011-08-17]. Available at <http://nicolas.burrus.name/index.php/Research/KinectUseNestk>.

[36] Nicolas Burrus homepage [online]. 2011 [cit 2011-08-17]. Available at <http://nicolas.burrus.name/>.

[37] ofTheo/ofxKinect at Github [online]. 2011 [cit 2011-08-17]. Available at <https://github.com/ofTheo/ofxKinect/>.

[38] openFrameworks [online]. 2010 [cit 2011-08-17]. Available at <http://www.openframeworks.cc/>.

[39] DEPTHJS – MIT Media Lab [online]. 2010 [cit 2011-08-17]. Available at <http://depthjs.media.mit.edu/>.

[40] Doug/depthjs at Github [online]. 2011 [cit 2011-08-17]. Available at <https://github.com/doug/depthjs>.

[41] R.B. Rusu; S.Cousins; “3D is here: Point Cloud Library,” *IEEE International Conference on Robotics and Automation (ICRA),* 2011.

[42] PCL – Point Cloud Library [online]. 2011 [cit 2011-08-17]. Available at <http://www.pointclouds.org/>

[43] G. Guennebaud, B. Jacob, et al., “Eigen v3,” <http://eigen.tuxfamily.org>, 2010.

[44] M. Muja and D. G. Lowe, “Fast approximate nearest neighbors with automatic algorithm configuration,” in International Conference on Computer Vision Theory and Application VISSAPP’09). INSTICC Press, 2009, pp. 331–340.

[45] Boost C++ Libraries [online]. 2005 [cit 2011-08-17]. Available at <http://www.boost.org/>.

[46] W. Schroeder, K. Martin, and B. Lorensen, Visualization Toolkit: An Object-Oriented Approach to 3D Graphics, 4th Edition. Kitware, December 2006.

[47] Documentation for MINPACK subroutine HYBRD1, Double precision version, Argonne National Laboratory, Burton S. Garbow, Kenneth E. Hillstrom, Jorge J. More, March 1980

[48] The CImg Library – C++ Template Image Processing Toolkit [online]. 2000 [cit 2011-08-17]. Available at <http://cimg.sourceforge.net/>.

[49] Lisa Gottesfeld Brown; “A survey of image registration techniques,” *ACM Comput. Surv*.,1992.

[50] Mahdi Ben Ghorbel; Malek Baklouti; and Serge Couvet; “3D head pose estimation and tracking using particle filtering and ICP algorithm,” *In Proceedings of the 6th international conference on Articulated motion and deformable objects* (AMDO'10), 2010.

[51] LP. Morency; “Stereo-based Head Pose Tracking using Iterative Closest Point and Normal Flow Constraint,” Master Thesis, Massachusetts Institute of Technology, 2002.

[52] Breitenstein, M. D; Kuettel, D.; Weise, T.; van Gool, L.; Pfister, H , “Real-Time Face Pose Estimation from Single Range Images,” 2008.

[53] Trefethen, Lloyd N.; Bau III, David (1997). Numerical linear algebra. Philadelphia: Society for Industrial and Applied Mathematics. ISBN 978-0-89871-361-9.

[54] Alberto_Albiol. Video indexing using multimodal information. PhD thesis, Universidad Politecnica de Valencia, Valencia, Spain, April 2003.

[55] P.Viola and M.Jones; “Robust Real-time Object Detection,” *International Journal of Computer Vision*, 2001.

[56] Chenghua Xu; Yunhong Wang; Tieniu Tan; and Long Quan; “Robust nose detection in 3D facial data using local characteristics,” *International Conference on Image Processing*, 2004. ICIP '04. 2004.

[57] Anuar L.H.; Mashohor S.; Mokhtar M; and Wan Adnan W.A., “Nose Tip Region Detection in 3D Facial Model across Large Pose Variation and Facial Expression,” *IJCSI International Journal of Computer Science Issues*, Vol. 7, Issue 4, No 4, July 2010.

List of Abbreviations

API	application programming interface
CImg	cool image c++ library
DOF	degrees of freedom
FPFH	fast point feature histogram
ICP	iterative closest point
IR	infrared
LGO	localized gradient orientation
NARF	normal aligned radial feature
PCA	principal component aanalysis
PCD	point cloud data
PCL	point cloud library
PFH	point feature histogram
SDK	software development kit
SVD	singular value decomposition
SVM	support vector machine

List of Appendix

A Content of CD

Attached CD contains:

1. **Thesis** – final version of the thesis in PDF format and all used pictures

2. **Code**
    a. C++: contains developed code in Eclipse (demo_4 is final product)
    b. Matlab: contains developer code in Matlab

3. **Database**
    a. Images: contains rgb and depth images
    b. Coordinates: contains extracted marker coordinates for each frame
    c. Angles: contains calculated Euler angles for each frame

4. **Multimedia** – multimedia material from demos and testing