Natural multimodal human robot interaction performed on a low cost robot head

DIPLOMARBEIT

Ausgeführt zum Zwecke der Erlangung des akademischen Grades eines

Diplom-Ingenieurs (Dipl.-Ing.)

unter der Leitung von

Ao. Univ.-Prof. Dr. techn. M. Vincze
Dr. techn. W. Wohlkinger

eingereicht an der

Technischen Universität Wien
Fakultät für Elektrotechnik und Informationstechnik
Institut für Automatisierungs- und Regelungstechnik

von
Markus Bajones
Schönburgstrasse 25/3
1040 Wien
Österreich

Wien, im August 2013

Vorwort

Zuerst möchte ich mich bei meiner Familie bedanken, für die Unterstützung die sie mir zukommen ließ, für die Möglichkeiten die sie mir geboten und für das Vertrauen daß sie immer in mich gesetzt hat.
Danke.

Außerdem gebührt den Menschen der Hochschülerinnen- und Hochschülerschaft, speziell natürlich der Fachschaft Elektrotechnik ein besonderer Platz in meinem Herzen. Mit vielen von euch habe ich Freundschaften geschlossen von denen ich hoffe sie bestehen ein Leben lang. Es war ein sehr schöner Weg den ich mit euch beschreiten durfte.

Weiters möchte ich mich bei Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Markus Vincze und Dipl.-Ing. Dr.techn. Michael Zillich für die Unterstützung und Hilfe bedanken die ich von ihnen in der Zielgeraden meiner Diplomarbeit erhalten habe.

Zu guter Letzt danke ich meinem Betreuer, Dipl.-Ing. Dr.techn. Walter Wohlkinger für die Unterstützung, Anregungen und Herausforderungen die ich von ihm erwarten durfte. Ohne ihn wäre der Weg zum Abschluss dieser Diplomarbeit kein so erlebnisreicher und spannender gewesen.

Vielen Dank.

Wien, im August 2013 Markus Bajones

Abstract

Within the scope of this diploma thesis we investigated the possibilities of mulitmodal human robot interaction with current technologies. For this, a pre-built low-cost robot head, called *Eva*, has been used and a software system was developed which includes state of the art algorithms from the fields of speech recognition, face detection, face recognition and object classification. Special attention was given to provide natural communication between users and the robot by using current speech recognition technology. Multiple systems were evaluated after integrating them in our implementation, before using one of them in our complete set-up. For the ability to find and identify individual people known algorithms were implemented and compared to each other. These include two variations of the *Viola-Jones* algorithm for face detection as well as *Eigenfaces*, *Fisherfaces* and *Local Binary Pattern* histograms for face recognition. These, in combination with face tracking by coupling the *Viola-Jones* algorithm with either a *Kalman filter* or a *Lucas-Kanade* optical flow estimation, provide one more part of the multimodal interaction between *Eva* and the user. Object classification provides the robot with the ability to perform further analysis in the field of interaction with objects. One method for this, using random decision forests, is explained as well.

Kurzzusammenfassung

Im Rahmen dieser Diplomarbeit wurde untersucht, welche Möglichkeiten für eine multimodale Mensch-Roboter-Interaktion mit aktueller Technologie bestehen. Dazu wurde ein bestehender Low-Cost Roboterkopf, genannt Eva, genutzt und ein Softwaresystem entwickelt welches State of the Art Algorithmen der Bereiche Spracherkennung, Personendetektion, Personenidentifikation und Objektklassifizierung integriert. Besondere Aufmerksamkeit galt der möglichst natürlichen Kommunikation zwischen Eva und den Anwendern. Dafür wurde auf die Möglichkeit eingegangen Spracherkennungssoftware der aktuellen Generation für die Kommunikation zu nutzen. Mehrere Systeme wurden dafür implementiert und deren Fähigkeiten evaluiert bevor sie im Gesamtsystem zum Einsatz kamen. Zum Auffinden und Identifizieren der Anwender wurden gängige Verfahren gegenüber gestellt. Dazu zählen *Viola-Jones* in zwei unterschiedlichen Varianten zur Gesichtsdetektion sowie *Eigenfaces*, *Fisherfaces* und *Local Binary Pattern* Histogramme zur Personenidentifizierung. Diese Methoden, sowie die Nachverfolgung von Gesichtern mit Hilfe eines *Kalman Filter* und eines *Lucas-Kanade* Trackers dienen der multimodalen Kommunikation mit den Anwendern. Außerdem wird ein System zur Klassifizierung von Objekten unter Verwendung von *random decision forests* erläutert, welcher in weiteren Folgen genutzt werden könnte um mit diesen Gegenständen sinnvoll zu interagieren.

Contents

List of Figures

1 Introduction

Human-robot interaction is a complex, difficult task. To set up even simple tasks involved sending commands with a programming device, a keyboard and joysticks or with an application on a computer. As robots are tools that should provide assistance for humans the act of commanding the robot to perform a task should be as natural as telling another person to do the same task. To make the interaction for the user this natural it is necessary for the robot to understand at least a subset of the spoken language of the user. This natural human robot interaction simplifies specifying tasks and are a prerequisite for acceptance by averages users. Further a robot has to be able to recognize a set of individuals and distinguish them from each other as well as learn new people. Another important ability for a robot is to classify objects in order to estimate its possible interaction with them and its capability to find such an object. This thesis uses a selected set of state of the art techniques to implement natural human robot interaction by providing face-detection and -recognition and speech recognition. In chapter 2 the requirements and the resulting decisions for the system are explained and the used frameworks are presented. Section 1 explains what hardware was used during this entire work and what goals it should be able to accomplish. The mathematical background of detecting and recognizing as well as tracking of a user by observing his or her face are described in chapter 3 and chapter 4. Object classification by the means of using decision forests is presented in chapter 5 and the handling of speech recognition is shown in chapter 6. In each of these chapters first an introduction to the state of the art and the methods used by our implementation as well as details about the implementation is given. Chapter 7 presents the evaluation of the implemented algorithms and show how well they operate.

Hardware

For the thesis a low-cost robot head was used, which is home to all needed sensors. The design of this head was chosen under the conditions to keep the cost of the complete implementation as low as possible while maintaining the ability to perform the following tasks.

pan and tilt
: of the head in order to track a user and to extend the range of vision during the task of classifying an object and the taks of searching a users face.

presentation
: of an artificial face to display emotions in order to provide a feedback to the user.

detection and classification of three-dimensional objects
as a way of learning items which could be grasped with an robotic arm and hand.

detection and tracking of people
to follow the users face during a conversation. This should provide a higher acceptance by users as the need to stay within a certain area is diminished for the user.

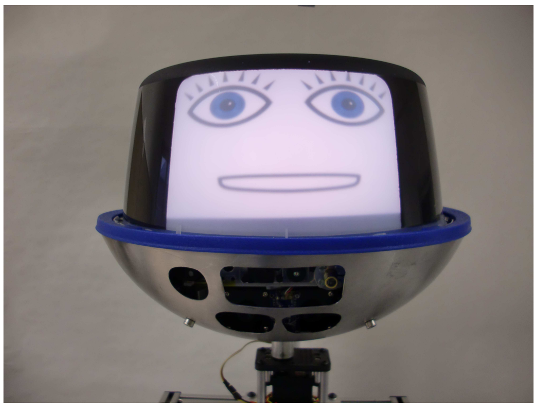


Figure 1.1: Eva - front view

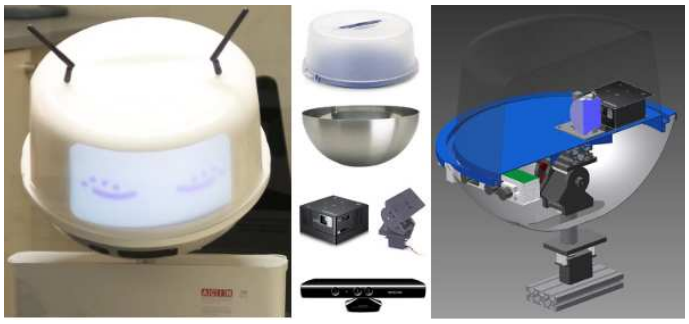


Figure 1.2: Eva - components and design. Source[1]

To accomplish these tasks the robot has two servomotors, one pico-projector, a stripped down Kinect sensor and one infra-red temperature sensor mounted inside its head. The power supply connectors as well as all connections for data transport are accessible on the main platform, which is about 50cm below the head. The complete platform is battery powered and connects to the control system on a computer using multiple USB connections. Figures showing the finished head, used components and the rendered design are figure 1.1 and figure 1.2.

[1]http://www.acin.tuwien.ac.at/?id=294

2 System decisions

As the main features of this work are people and face detection, recognition and tracking, speech recognition for voice commands and object detection and classification, the implementation requires the combination of a hardware part, which has been built in a previous work, and the software stack.

2.1 Used frameworks

For the software implementation we chose ROS as the de-facto standard framework for robotics, OpenCV and Point Cloud Library for their abilities in computer vision for 2D-, 3D- as well as depth-images and SMACH as state machine library. A short introduction to each of these components is given in this section.

Robot Operating System (ROS)

The use of the ROS[2] software framework, which can be seen as the standard software framework in the field of robotics, gives the possibility to reuse and adapt modules. This framework distributes the workload onto individual nodes, which can be on the same or on different hardware and software platforms. These include various GNU/Linux distributions on platforms which support the C++ or Python programming languages as well as Android based devices via the Java based rosjava[3] system. For any form of communication between the nodes, they have to register to the so called master node, which has to be started as the first process. ROS defines a few basic principles of communication between ROS-nodes.

Topics handle the communication in a one-way style. Every node can publish and listen to a topic. Each topic has a well defined message format, which has to be used to publish data onto this topic. This is extremely useful to send a continuous data stream of video frames from a module handling the recorded data from a camera (two-dimensional or with additional depth information) or other devices like temperature- or distance-sensors. The sensor data gets published at a fixed rate in one message at a time.

Services handle the communication in a slightly different way. Nodes which use this form of communication have to implement either a server or a client. The client sends a request message to the server, that processes the data in the request and

[2]http://www.ros.org
[3]http://code.google.com/p/rosjava/

puts the result data in the response. Afterwards this response will be handed to the client over the network.

Actionlib Due to the fact that some tasks require a long time to be executed (e.g. navigation, pick and place process) another form of communication is provided by the Actionlib package. The communication between a client and server application is done via the ROS Actionlib[4]protocol, which is built on top of the ROS message stack and reuses the publisher/subscriber infrastructure as illustrated in figure 2.1.

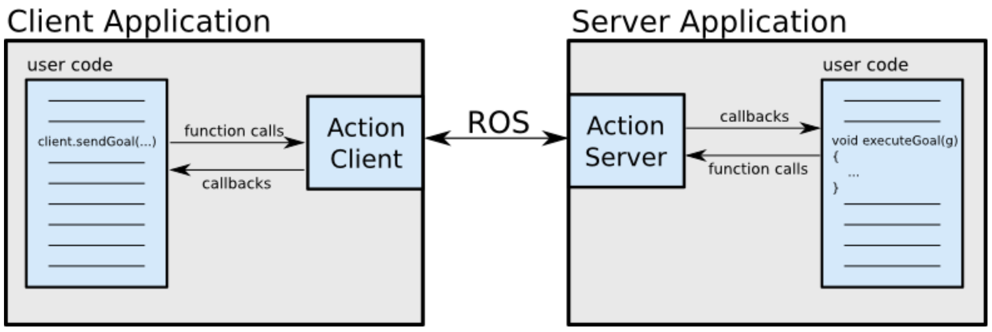


Figure 2.1: Actionlib Client-Server interaction. Source[4]

Actionlib specifies three types of messages between the Action client and the Action server.

Goal In here the client sends informations and parameters of the desired workload.

Feedback The server sends feedback messages on a regular time basis to inform the client on the progress of the desired goal.

Result The result message is only sent once as it marks the reaching of the goal.

It is possible to define and execute multiple Actionlib goals in one server, but as it is overcomplicated and not needed in this work we will only explain the Simple-ActionServer which discards a goal whenever a new or a *preempt* goal is received. The communication between client and server is best described in the way that the client wishes to inform the server it should work on a specific task, e.g. to move the head to a certain position. The client implementing the ActionClient generates the corresponding goal message and publishes it on the automatically created topic onto which the ActionServer inside the server application is subscribed. On reception of the goal it starts the corresponding code and sends the feedback messages back to the client. From here on two things can happen: First, the ActionServer receives another goal message which will trigger the execution of the user code given by the data in the new goal. Second, the task for the last goal has been reached and the final message - the result - is sent to the ActionClient. After this the server waits for any new goal messages.

[4]http://www.ros.org/wiki/actionlib

OpenCV

OpenCV[5] is a widely used open source computer vision library originally developed by the Intel Corporation and is supported by the non-profit organisation OpenCV.org since 2012. OpenCV is a comprehensive collection of the most commonly used state of the art computer vision algorithms. In this thesis it is widely used to detect, track and identify a person, but this is only a small subset of features offered by this library. An overview and documentation about the features currently offered by OpenCV are provided on http://docs.opencv.org/.

Point Cloud Library

PCL[6] is a framework dedicated to ease the processing of point clouds and 2-D/3-D images. Features used in this thesis include registration, segmentation, keypoint and feature detection and are part of the object classification. Details about their implementation, usage and further documentation is available on http://pointclouds.org/documentation.

SMACH

SMACH[7] is a Python library, designed to build hierarchical state machines. It provides an easy way to develop robust robot behaviour with maintainable and modular code. Even though it is independently usable without ROS SMACH is well equipped to be used in a ROS-aware application. It also provides a useful debugging mechanism in form of the graphical SMACH viewer, in which the state machine with the current transitions, states and shared data can easily be observed in real time.

2.2 Systems state machine

In figure 2.2 the design of the state machine is pictured. It consists of four major states. These are *main*, *learn person*, *follow face* and *classifiy object*.

Response is a helper state which gives a verbal confirmation or reply to the user, so that a feedback system is implemented. Especially for tasks which take longer than 30 seconds for the confirmation, that the command has been understood and the robot is in the process of handling it, this is a good measure.

Start handles the start up of all needed services, checks the availability of the needed devices and runs the main loop if the checks return a positive result.

The *end* state on the other hand is responsible to shut down all remaining processes and free the remaining system resources and devices. In *main* voice commands are received, checked against available features and the next state is executed with the appropriate arguments supplied to it.

[5]http://www.opencv.org
[6]http://pointclouds.org
[7]http://http://www.ros.org/wiki/smach

Follow face will try to locate a persons face in the supported field of view and if found, hold the face in the center of the recorded image with the implementations detailed in section 3.1, section 3.2 on page 13 and chapter 4 on page 23 until the state gets stopped by another command.

Learn person handles the addition of a new person into the database of known individuals and will exit after the training process is done or it gets pre-empted in which case the person is not added. The state *classifiy object* will take its input point cloud, perform classification by means explained in chapter 5 on page 31 and return the name of the objects class, as well as the certainty it achieved. If no object can be found an error message is returned instead.

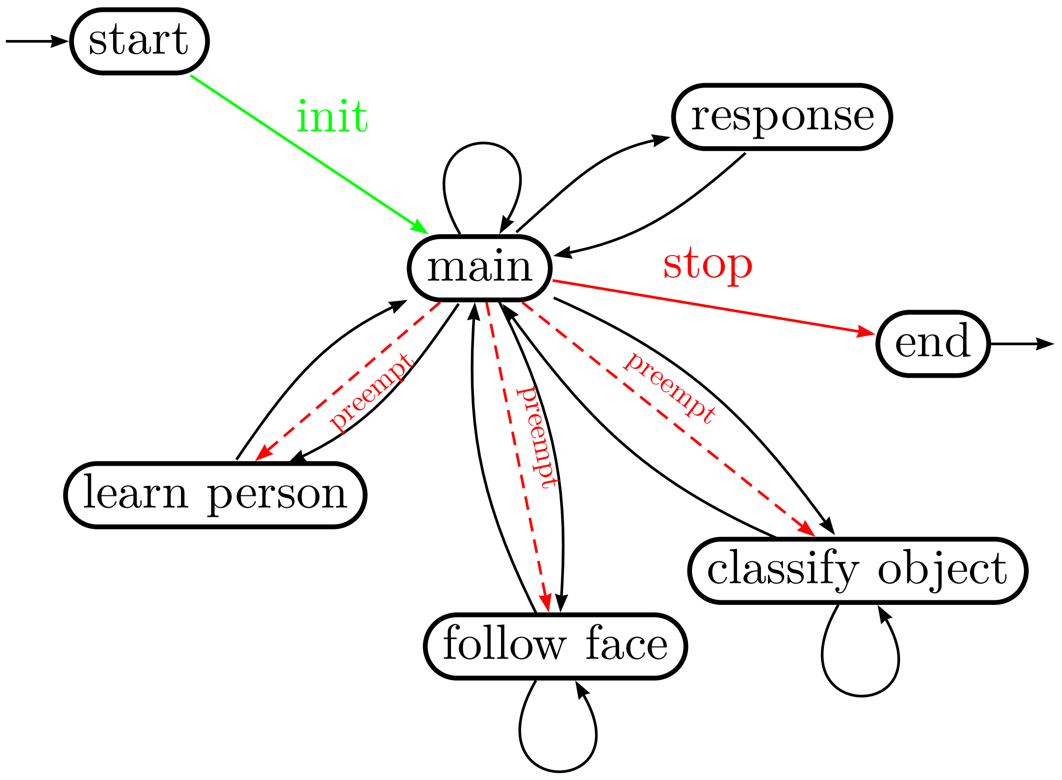


Figure 2.2: State machine design

The following chapters will detail the techniques and implementation of the above components and their interaction in the state machine.

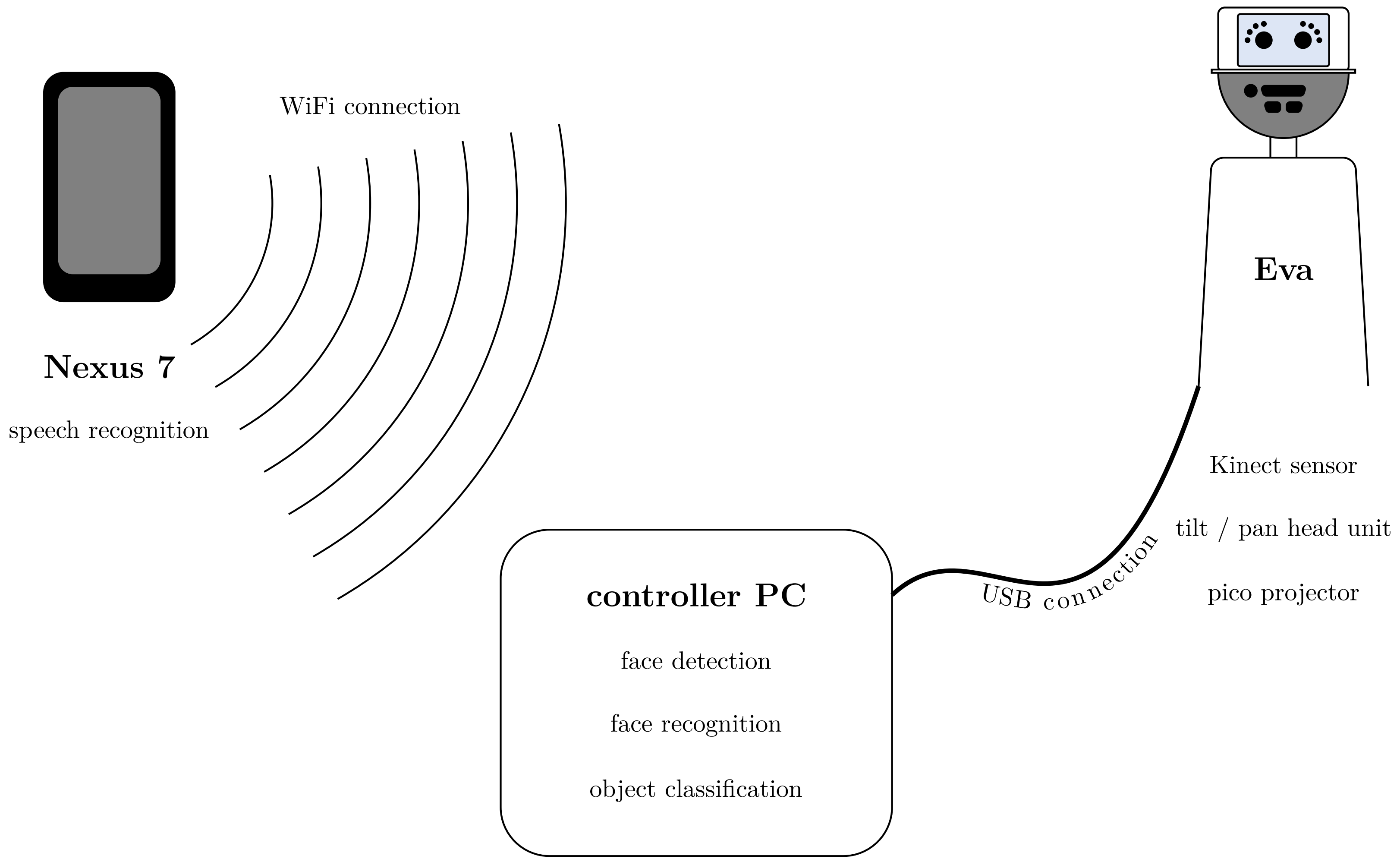


Figure 2.3: System summary

3 Face detection and recognition

In this chapter the basic methods to detect and identify a user are explained. First the detection of a face by analysing two-dimensional image data, then multiple algorithms for classification and thereby identification of single users are described.

3.1 Face detection

In any application where human machine interaction has to feel like an interaction between two individual people it is essential for the system to identify the human as quickly and reliable as possible. After a human is found the robot should address the human by looking at the persons face in the ongoing conversation. In this thesis the robot is not mobile, so there is no way of leaving the designated work space to look for the user. A face can only be searched in the view area which is limited by the pan and tilt angles of the two servomotors on which the robot head is mounted. As it is assumed that the robot is only in use when a user is present the face detection mechanism has to look for a human and his or her face in the possible area. To perform face detection, according to Yang et al. [29] and Szeliski [24], a feature-based, template-based, or appearance-based approach can be used.

feature-based
: This approach tries to find unique features, like the nose, mouth, eyes and ears, and if found calculates the plausibility of their geometrical relations to each other.

template-based
: Template faces are constructed and compared at different scales to all regions of an image. These templates can be as simple as elliptic outlines for the head, the eyes and the mouth region. If the overlay of the template matches a region this region is considered to be a face.

appearance-based
: Use training sets of face and non-face images to learn a classifier to distinguish between them.

All of these methods must find some way to overcome several barriers to detect faces on a reliable basis. Obvious challenges include:

face position and orientation
: The look of a face changes heavily under the influence of the camera's position relative to the face. May that be the fact that the camera takes its pictures in an

45° downward angle, the person rotates her head or is just too close or far away from the camera.

light conditions
Different lighting conditions, resolution changes as well as the sensor and lens properties affect the ability to perform the search with the desired result.

occlusion
A part of the face can be occluded by another object, which can disturb the detection algorithms. These obstacles can be anything from glasses, facial hair to objects passing by.

3.1.1 Viola-Jones Algorithm

The Viola and Jones [25] algorithm describes the object detection in images based on the value of simple features. Although the name suggest that this is a feature based method it is in fact appearance-based, as can be seen in this section. These simple features are similar to the Haar basis functions used by Papageorgiou et al. [21]. Viola and Jones use three distinguished features (two-, three- and four-rectangle based features). The value of a two-rectangle feature is calculated as the difference of the sum of pixel values within two rectangular areas. These areas are of same size and shape and are neighbouring each other horizontally or vertically. A three-rectangle feature is built by the sum of the pixel values from the two outside regions subtracted from the sum in the middle rectangle. The last one, the four-rectangle feature, calculates its value as the difference of the sums of the diagonal pairs of rectangles. A graphical representation of the possible features can be seen in figure 3.1 and an example of features on the Lenna sample face is in figure 3.3 on page 11.

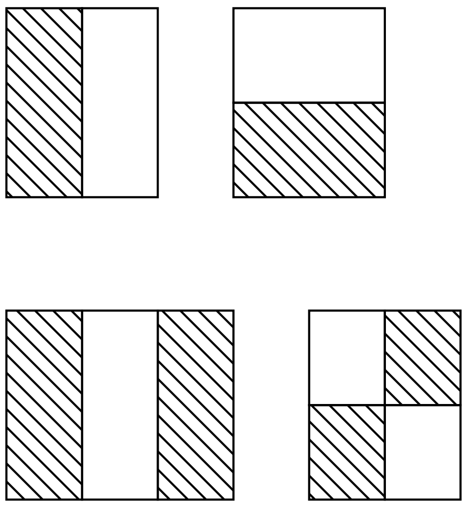


Figure 3.1: 2-, 3- and 4-rectangle based features

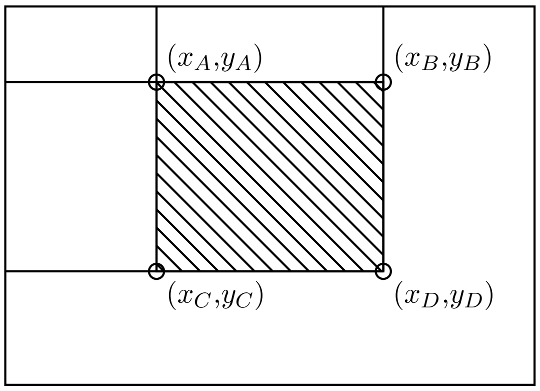


Figure 3.2: Integral image areas

Integral Images

For fast processing of the features the integral image is calculated. It is defined at the point (x, y) as the sum of all pixels to the left and above beginning at the point $(0, 0)$.

$$I(x,y) = \sum_{\substack{x' \leq x \\ y' \leq y}} i(x',y') \tag{3.1}$$

The full integral image, for every point (x, y) of the processed image, can be computed in one single pass. Afterwards it is possible to easily retrieve the sum of any rectangular region of the original image by four lookups in the integral image figure 3.2.

$$\sum_{\substack{x_A \leq x' \leq x_D \\ y_A \leq y' \leq y_D}} i(x',y') = I(x_D,y_D) + I(x_A,y_A) - I(x_B,y_B) - I(x_C,y_C) \tag{3.2}$$

To obtain the features for two-rectangles we need six lookups in the integral image, for three rectangles eight and for four-rectangles nine lookups. Within a 24×24 pixel sub-window x there are $45,396$ rectangular features which can be evaluated by weak classifiers $h_j(x)$. Each of these classifiers is described as

$$h_j(x) = \begin{cases} 1 & \text{if } p_j f_j(x) < p_j \theta_j \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

with a feature f_j, a threshold θ_j and a parity p_j indicating the direction of the inequality sign.

To overcome the need to compute each of these features a variant of the AdaBoost learning algorithm is introduced.

[8]http://ahprojects.com/projects/cv-dazzle

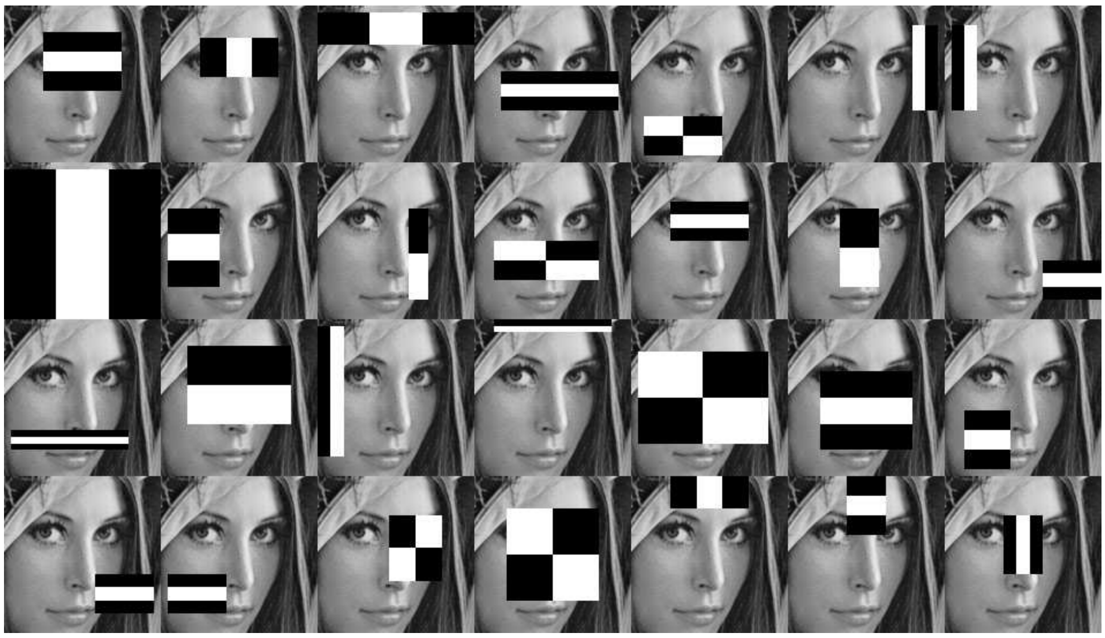


Figure 3.3: Haarcascade features visualization. Source[8]

AdaBoost

AdaBoost, which is short for Adaptive Boosting, is a learning algorithm, which combines multiple weak learners to build a stronger one. The weak learners are classifiers which can change the odds of an 0 or 1 decision in favour of one of the possibilities by a small margin, in the Viola-Jones algorithm these are the two-, three- and four-rectangle feature classifiers. The boosting of these weak classifiers can be described by the following algorithm.

- Given a set of training images $(x_1, y_1) \dots (x_n, y_n)$ with $x_i \in X$ and $y_i \in \{0, 1\}$ for negative and positive example images.
- Initialize weights:

$$w_{1,i} = \begin{cases} \frac{1}{2m} & y_i = 0 \\ \frac{1}{2l} & y_i = 1 \end{cases}$$

  m ... number of negative, l ... number of positive examples

- For $t = 1 \dots T$ (T ... number of classifiers constructed by a single feature)
  1. Normalize weights:

$$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$

2. For each feature j train a classifier h_j equation (3.3) on page 10 which is composed of one single feature and calculate the error with respect to w_t.

$$\epsilon_j = \sum_i w_i |h_j(x_i) - y_i|$$

3. The classifier with the lowest error ϵ_t gets chosen and the weights updated according to e_t.

$$w_{t+1,i} = w_{t,i}\beta_t^{1-e_i}, \quad \beta_t = \frac{\epsilon_t}{1-\epsilon_t}$$

with $e_i = 0$ if x_i was classified correctly, $e_i = 1$ if not.

- The final strong classifier is then:

$$h(x) = \begin{cases} 0 & \sum\limits_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum\limits_{t=1}^{T} \alpha_t, \quad \alpha_t = \log \frac{1}{\beta_t} \\ 1 & \text{otherwise} \end{cases}$$

The AdaBoost learning algorithm provides us with strong classifier to select only the best features in the images, as the weaker features get eliminated in one of the individual stages of the trained classifier.

Training the face cascade

It is possible to train the classifier yourself by providing positive samples, which contain the object the classifier should later be able to identify and negative samples, which bear no resemblance to a human face. The quality of the final classifier is always dependent on the training data. As this approach is widely used a collection of already trained face cascades is available for free use in OpenCV, introduced in section 2.1 on page 5.

Cascade

The classifiers extracted by AdaBoost are now placed in a cascade system with the weakest, but on the other hand fast, classifier at the beginning of it. These simpler classifiers can reject most of the sub-windows before the more complex classifiers are called. Each classifier can reject its input or trigger the evaluation of it by the next stage. Only if all stages of the cascade evaluated the input region as true it is accepted for further processing. This results in a boosted as in faster, more efficient calculation, as a majority of regions in an image do not have to be evaluated by all classifiers of the cascade chain. The schematic of this cascade design is pictured in figure 3.4 on the next page.

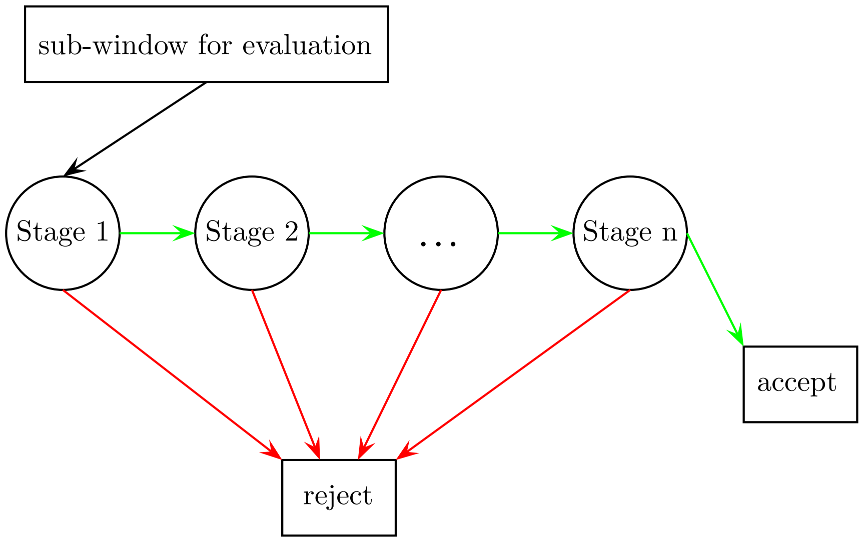


Figure 3.4: AdaBoost cascade

3.2 Face recognition

Face detection is a two-class classification problem, where the goal is to assign a region of an image either the class "face" or the class "non-face". Face recognition on the other hand aims to provide the possibility to differentiate one member of the class "face" from every other member of it and is therefore a multi-class classification.

One of the first things a human learns in his life is to recognize people in their surrounding and to differ between them. It comes naturally for a human to recognize these people among a group of other humans and to memorize previously unknown people after a short period of time. The more contact the child has with these human beings, the more it learns who they are and that they can and sometimes should listen to them. For our robot, Eva, we aim to implement this behaviour, so that we are able to follow a recognized person's face. Small differences in appearance can make it difficult to get a certain recognition rate. To reduce the possible patterns a robot could verify we limit the area of recognition to the human face. This is the most prominent feature to distinguish two people from each other which can be obtained by using a 2D or 3D camera alone. In the following section I will lay down the mathematical background to three two-dimensional and one three-dimensional based face recognition approaches. Afterwards a short introduction to the implementation of this module is given and in section 7.1 on page 45 an comparison between the implemented algorithms is given.

3.2.1 Eigenfaces

The Eigenfaces algorithm uses the following approach to recognize faces in a given training set: With the definitions from Belhumeur et al. [1] this approach can be described in the following manner: First we define the image space, a high-dimensional space which elements are the pixel-values of the sample images. This high-dimensional space gets processed by principal components analysis (PCA) to reduce the dimensionality. This uses a dimensionality reducing projection which maximizes the scatter of all projected samples. Starting with N sample images $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ with pixel values from 0 to 255 (grey-scale images) in the n-dimensional image space, with each image in one of the c classes $\{X_1, X_2, \ldots, X_c\}$. We then consider a linear transformation mapping the n-dimensional image space into a m-dimensional feature space, where $m < n$. The feature vectors $\mathbf{y}_k \in \mathbb{R}^m$ are defined as:

$$\mathbf{y}_k = W^T \mathbf{x}_k \quad k = 1, 2, \ldots, N$$

where $W \in \mathbb{R}^{n \times n}$ is a matrix with orthonormal columns. The total scatter matrix is defined as

$$S_T = \sum_{k=1}^{c} (\mathbf{x}_k - bm\mu)(\mathbf{x}_k - \boldsymbol{\mu})^T$$

where $\boldsymbol{\mu} \in \mathbb{R}^n$ is the mean image of all sample images and c is the number of classes. After applying the linear transformation W^T the scatter of the transformed feature vectors $\{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N\}$ is $W^T S_T W$. During PCA, the projection W_{opt} which maximizes the determinant of the total scatter matrix of the projected samples is chosen.

$$W_{opt} = \arg\max_W |W^T S_T W| = [\mathbf{w}_1 \; \mathbf{w}_2 \; \ldots \; \mathbf{w}_m]$$

where $\{\mathbf{w}_i | i = 1, 2, \ldots, m\}$ is the set of n-dimensional eigenvectors of S_T corresponding to the m largest eigenvalues. These eigenvectors have the same dimension as the original images and are called Eigenfaces, see the example set in figure 3.5 on the next page.

3.2.2 Fisherfaces

This recognition is based on Fisher's linear discriminant (FLD) which is in close relation to linear discriminant analysis (LDA). Both of them look for linear combinations of variables which best fit to the given data points. Parts of the thorough explanation from Mika et al. [18] and Welling [27] are given bellow. We define the between-classes scatter matrix as

$$S_B = \sum_{i=1}^{c} N_i (\boldsymbol{\mu}_i - \boldsymbol{\mu})(\boldsymbol{\mu}_i - \boldsymbol{\mu})^T$$

and the within-class scatter matrix as

$$S_W = \sum_{i=1}^{c} \sum_{x_k \in X_i} (\mathbf{x}_t - \boldsymbol{\mu}_i)(\mathbf{x}_k - \boldsymbol{\mu}_i)^T$$

[9] http://docs.opencv.org/trunk/modules/contrib/doc/facerec/facerec_tutorial.html

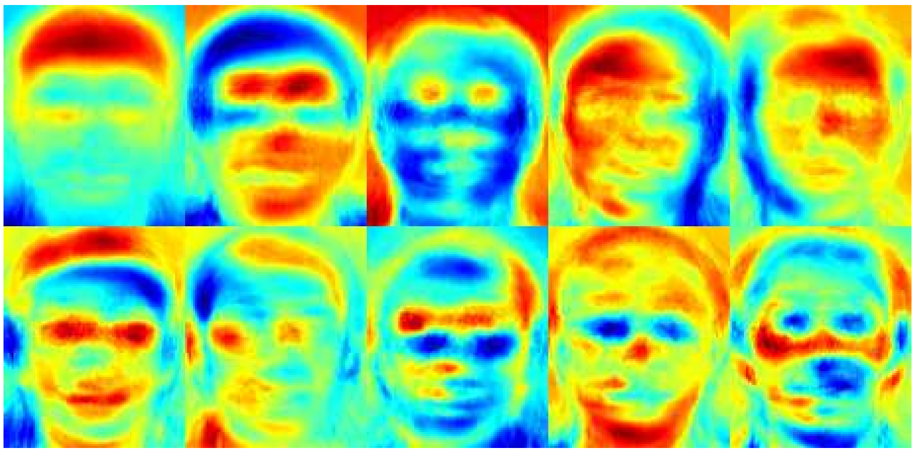


Figure 3.5: Eigenface visualized. Source[9]

where $\boldsymbol{\mu}_i$ is the mean image of class X_i, $\boldsymbol{\mu}$ is the mean image of all classes, N_i is the number of samples in class X_i and c the number of classes. If S_W is non-singular, we chose the optimal projection W_{opt} as the matrix with orthonormal columns which maximizes the ratio determinant of the between-class scatter matrix of the projected samples to the determinant of the within-class scatter matrix of the projected samples, i.e.

$$W_{opt} = \arg\max_{W} \frac{|W^T S_B W|}{|W^T S_W W|} = [\mathbf{w}_1 \; \mathbf{w}_2 \; \dots \; \mathbf{w}_m] \tag{3.4}$$

where $\{\mathbf{w}_i | i = 1, 2, \dots, m\}$ is the set of generalized eigenvectors of S_B and S_W corresponding to the m largest generalized eigenvalues $\{\lambda_i | i = 1, 2, \dots, m\}$, i.e.

$$S_B \mathbf{w}_i = \lambda S_W \mathbf{w}_i, \quad i = 1, 2, \dots, m.$$

As there are $c - 1$ non-zero generalized eigenvalues it is an upper bound on m. To overcome the problem that the within-class scatter matrix $S_W \in \mathbb{R}^{n \times n}$ is always singular, because of the face that the rank of S_W is at most $N - c$ and the number of images in the learning set is much smaller than the number of pixel in each image, we adapt FLD. We reduce the dimension of the feature space to $N - c$ with PCA. Afterwards we apply the standard FLD equation (3.4) to reduce the dimension to $c - 1$.

$$W_{opt}^T = W_{fld}^T W_{pca}^T \tag{3.5}$$

where

$$\begin{aligned} W_{pca} &= \arg\max_{W} |W^T S_B W| \\ W_{fld} &= \arg\max_{W} \frac{|W^T W_{pca}^T S_B W_{pca} W|}{|W^T W_{pca}^T S_W W_{pca} W|}. \end{aligned}$$

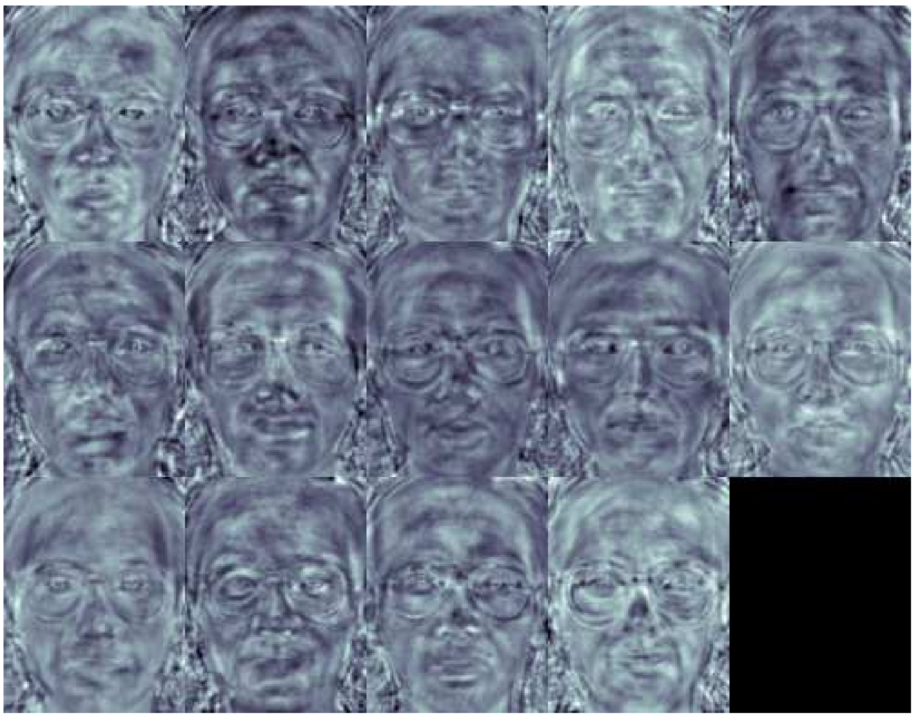


Figure 3.6: Fisherface examples. Source[10]

Visualized Fisherfaces can be observed in figure 3.6

3.2.3 Local binary patterns histograms

Another popular method used in the field of pattern recognition is called Local Binary Pattern Histogram, or LBPH. Ojala et al. [19] introduced the basic LBP operator for a pixel at position x_c, y_c, its intensity i_c and eight surrounding neighbours with their corresponding intensities i_p. The LBP operator assigns a label to every pixel by thresholding the 3×3-neighbourhood with the value of the center pixels intensity. We use both, the decimal and the binary representation of the resulting label, in further considerations.

$$LBP(x_c, y_c) = \sum_{p=0}^{p-1} 2^p sign(i_p - i_c), \quad sign(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

Extended versions of the LBP handle greater neighbourhoods than 3×3 and uniform Local Binary patterns. The first one $(LBP(P, r))$ calculates the value for P sample points at a circle of the radius r instead of the fixed position neighbours. The second extension classifies a Local Binary Pattern as uniform if no more than two transitions from 0 to 1 or vice versa can be found in the binary representation of the $LBP(P, r)$

[10]http://docs.opencv.org/trunk/modules/contrib/doc/facerec/facerec_tutorial.html

and is noted as $LBP(P,r)^{u2}$. The remaining non-uniform LBP's receive the same label (binary representative number). A histogram for the labelled image f_l is defined as

$$H_i = \sum_{x,y} I\{f_l(x,y) = i\}, \quad i = 0, 1, \ldots, n-1 \tag{3.7}$$

where n is the number of unique labels resulting from the LBP operator and

$$I(a) = \begin{cases} 1 & \text{if a is true} \\ 0 & \text{if a is false.} \end{cases}$$

This histogram contains information about the distribution of local patterns which correspond to edges, corners, lines, spots and flat areas over the whole image. For efficient face recognition the addition of spatial information is introduced Chan et al. [4]. This is done by splitting the image into m regions $R_0, R_1, \ldots R_{m-1}$ and defining the spatially enhanced histogram as

$$H_{i,j} = \sum_{x,y} I\{f_l(x,y) = i\} I\{(x,y) \in R_j\}, \quad i = 0, 1, \ldots, n-1, j = 0, 1, \ldots, m-1 \tag{3.8}$$

This process is visualized in figure 3.7 on the following page.

3.2.4 3D face recognition using local appearance-based models

Face recognition based on 3-dimensional data needs more preparations than its 2-dimensional counterparts. This results in a greater computational effort, but as the results are not influenced by pose variation and illumination changes the added work is well worth it. Koschan et al. [15] defines three stages which are necessary prior to the actual face reconstruction process. Afterwards a depth-map of the recorded face is present and subject to a recognition analysis.

data acquisition

In this stage the data from a 3-D sensor is acquired and preprocessed. This is needed as the depth data has spikes or holes, often corresponding to eyeballs and eyebrows (Bowyer et al. [2]), as well as noise from the sensor itself. To handle the spikes a median filter is applied, the holes are filled using linear interpolation. For a smoother face surface a Gaussian filter is used as well.

data registration

Wherein all faces are transformed to a common coordinate system. This transform is based on common landmarks which are placed on salient facial features such as the nose tip, the endpoints of the mouth, the lowest point of the chin as well as the endpoints of the eyes. The base mesh, onto which all faces are to be projected on, is calculated by using the generalized Procrustes algorithm (Gower [8]). Regions outside the face itself, such as the neck and ears, are removed from the depth image.

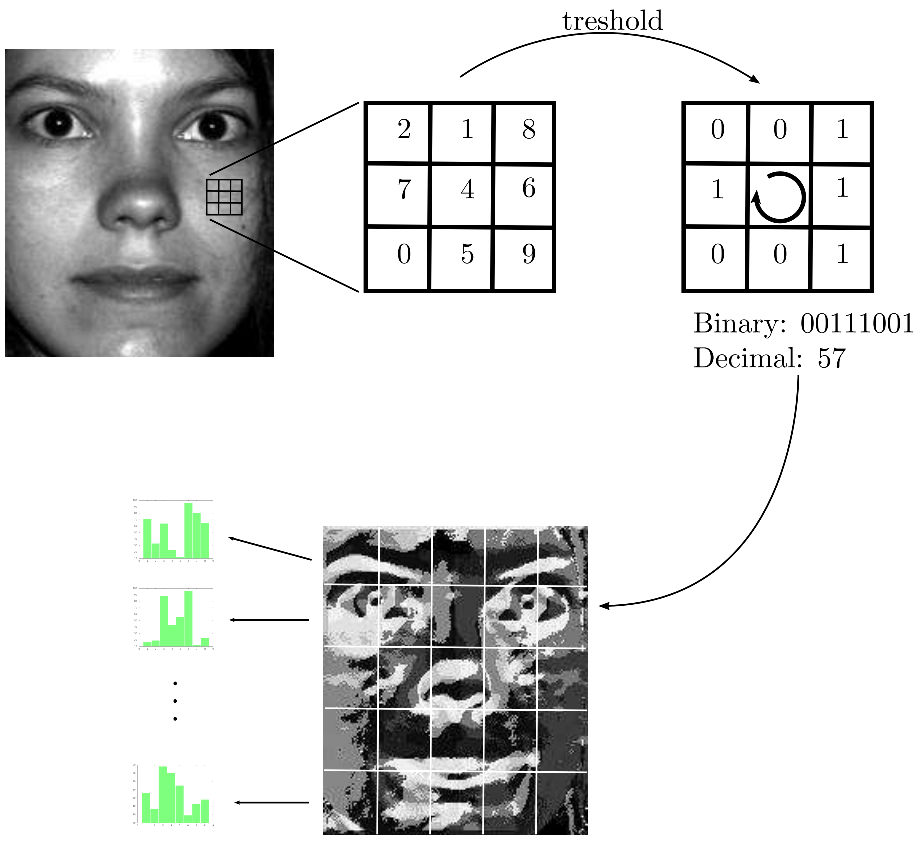


Figure 3.7: LBPH extraction sequence

integration

This stage handles the re-sampling of the face as depth map based on the projected mesh from the registration stage. Commonly used are closest-point, which result in folds and uneven sampling where the correspondence between surfaces with high curvature is not very close. These can result in points not being sampled at all, which would introduce a fold in the final mesh. Ekenel et al. [7] used ray-casting as an alternative re-sampling method, which they described as follows. A ray is cast through each vertex of the base mesh along the z-axis onto the target surface. If there is a crossing point on the target surface it is considered to be the re-sampled point. If it does not exist, most often at the border of the base mesh, the closest-point mapping is used to acquire the re-sampled point.

For the resulting depth-map a local face representation based on the discrete cosine transform (DCT) is generated using the following steps. Divide the input depth image into 8×8 pixel sized regions. For each block the DCT coefficients are calculated and represent the entire block with $N \times M$ being the size of the input block, $f(i, j)$ as the intensity of the input pixel at position (i, j) and $F(u, v)$ as the DCT coefficient for the

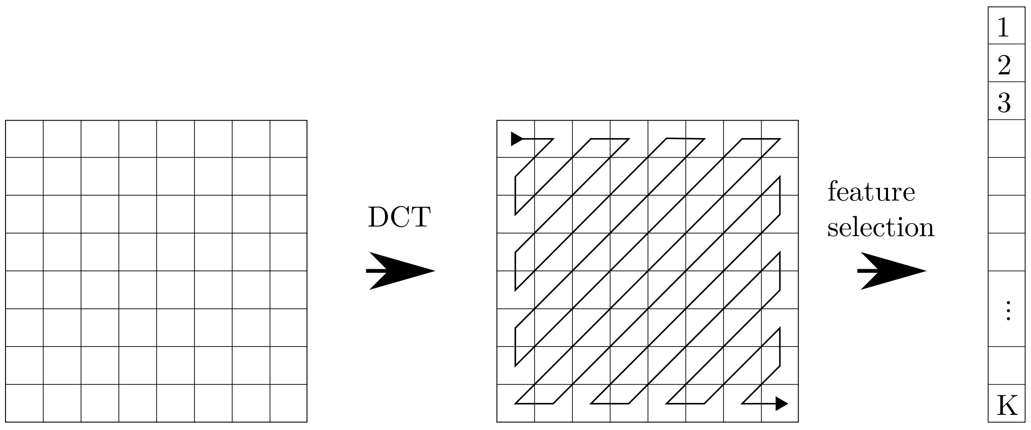


Figure 3.8: Zig-zag scan to obtain the K-dimensional feature vector

pixel at (i,j).

$$F(u,v) = \sqrt{\frac{2}{N}}\sqrt{\frac{2}{M}} \sum_{i=0}^{N-1} A(i) \cos\left(\frac{u(2i+1)\pi}{2N}\right) \sum_{j=0}^{M-1} A(j) \cos\left(\frac{u(2j+1)\pi}{2N}\right) f(i,j) \quad (3.9)$$

where

$$A(i) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } u = 0 \\ 1 & \text{otherwise} \end{cases} \qquad A(j) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } v = 0 \\ 1 & \text{otherwise} \end{cases}$$

The DCT coefficients are ordered using a zig-zag pattern as shown in figure 3.8. This is done to group low frequency coefficients together at the top of the vector. For an image of size 128×128 we obtain 16384 coefficients. From them the first $K = 128$ coefficients are selected. The K-dimensional vector from each block of the input image is then concatenated to construct the overall feature vector of the corresponding depth image.

Before a classification is possible the training phase has to be started. During this phase a set of face images is run through the classifiers and the resulting data is stored in a database in combination with a label corresponding to the class. This data differs from method to method and consists of eigenvectors for the Eigenface or Fisherface algorithms, LBP-historgram or DCT-coefficients for the local appearance-based method. To perform a classification of a recorded face the data obtained by the descriptors are fed into a decision engine, which returns the class in which the face belongs with the highest probability. One of the most common used algorithms to perform the classification is the k-nearest neighbour method, which can be extended by including a weight in form of the inverse of Euclidean distance from the test data for every k-neighbours. Another popular classification method is to use support vector machines (SVM). These work by mapping the original finite-dimensional data into a higher-dimensional space where the data is linearly separable by a hyperplane. Standard SVMs only distinguish between two classes, but SVMs handling multiple classes have emerged. This works by either dividing a multiclass problem into multiple binary classification problems or cast the multiclass problem onto a constrained optimization problem with a quadratic objective function [5].

3.3 Implementation

The implementation we chose is based on the people_perception module from the Fraunhofer Institute for Manufacturing Engineering and Automation (IPA)[11]. This module was chosen as it provided most of the functions needed for face detection and recognition for a previous release of ROS. Therefore only slight modifications were needed to run the first evaluation trials. The people_perception module uses the Eigenface method for face recognition, but thanks to the fact that the original face images and not the Eigenfaces themselves were stored the data could be reused without invasive code changes. This and the circumstance that the procedure which calculates the Eigenfaces could easily be replaced with the equivalent for Fisherfaces or Local binary pattern histograms.

face detection

This module receives its input data from the topic */camera/depth_registered/points* which contain the coloured point cloud data (RGBD) supplied by the "opennimodule" handling the Kinect-sensor. The first step in this module is to limit the frame rate to constant 25Hz and passes the coloured point cloud data to the next node in the processing pipeline. In the head detector node the RGB-D data is split into a depth- and a color-image. On the latter one a Haar-like feature cascade is applied and each found head-region is published on topic */cob_people_detection/head_detector/head_positions*. If a face has been found there are two further possibilities in the modules code execution path. The first is to learn the new face by acquiring a set of 50 pictures of the face region, calculate the corresponding Eigenfaces and save them and the name of the person in the internal database. Afterwards the database is reloaded to be able to use the new information as soon as possible. The second possible path is to compare the newly found face against the stored set of people in the database. This is done by calculating the Eigenface of the face region and run a nearest neighbour search against the training set in the database. If a person is found which matches the tested face with more than a fixed probability it will publish on topic */cob_people_detection/face_recognizer/face_recognitions* informations about the person which include the position and size of the region of interest of the head and face, the persons name and the distance from the sensor. These will be used by both the detection tracker node which aims to keep track of the faces, deletes multiple detections of the same person, evaluates the plausibility of the face's movement between two frames and publishes the confirmed head positions as well as needed meta-data on */cob_people_detection/detection_tracker/face_position_array* which is used in the people detection display node to finally display the original RGB image data as well as an overlay of information gathered through the process detailed in this section. The same data is used to follow the recognised person with the robots head by the use of the two servo motors.

face tracking

The goal of this node is to follow a persons face of which the name has been published on */tuw_acin_eva/processed_commands* and control the two servomotors in a way that the tracked face is in the middle of the picture from the Kinect sensor. For this it needs the face positions provided by */cob_people_detection/detection_tracker/face_position_array* and utilizes the Actionlib package to communicate with the node in charge of controlling the hardware of the servomotors. The Actionlib approach with only a SimpleActionServer is chosen because of the fact that the positions of the tracked face and the head move in an independent manner. So if the head moves to the desired position but the face moves away from the center of the viewed area in which it should be, the head pose can easily be changed by sending a new goal to the SimpleActionServer. With an implementation based on topics or services it would have one of the following problems: Either the first sent position would always be moved to, even if the face was already in the desired area of view. Only then a new pose could be sent to the servomotor controller, or multiple nodes could send speed or position data to the servomotors, which would result in an unknown or at least jittery movement. A big problem for the tracker node is the fact that the face detector is unable to deliver a accurate position of a head in every frame. This can lead to the annoying fact that there can be gaps between frames with confirmed face detections. As this would lead to a trajectory with jumps between the desired positions a node which implements a Kalman filter is added between the face detection and the node controlling the servo motors. This provides the possibility to position the head to a well defined position corresponding to every time frame.

As the implementation of face detection and face tracking is done in a single module we did anticipate the background details for face tracking which are explained in the next chapter.

[11] http://www.ros.org/wiki/cob_people_detection

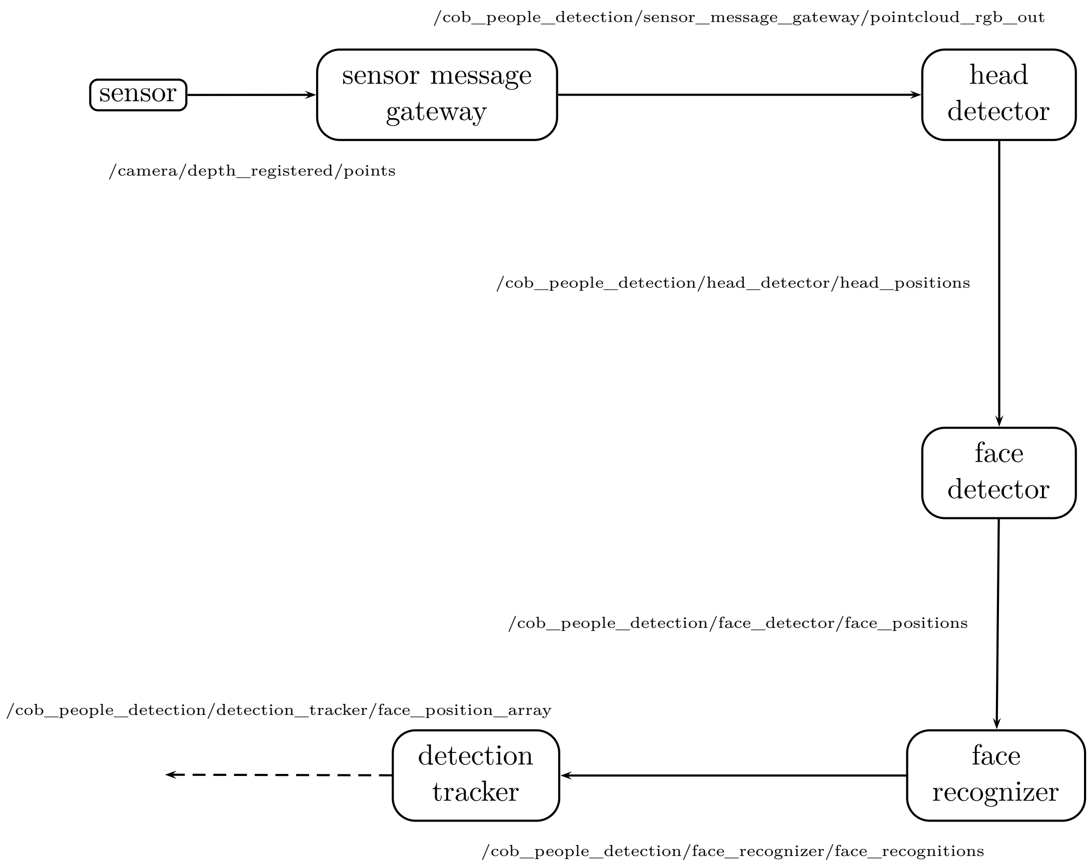

/cob_people_detection/sensor_message_gateway/pointcloud_rgb_out
sensor
sensor message gateway
head detector
/camera/depth_registered/points
/cob_people_detection/head_detector/head_positions
face detector
/cob_people_detection/face_detector/face_positions
/cob_people_detection/detection_tracker/face_position_array
detection tracker
face recognizer
/cob_people_detection/face_recognizer/face_recognitions

Figure 3.9: People detection flowchart adopted from [11]

4 Face tracking

For the human robot interaction it is vital to be able to follow the users face and in doing so give the user the feeling that the robot is aware of his or her presence and that the robot is listening and following the interaction. Face tracking suffers from one great problem, which is the inability to detect the users face in each situation, even when a face is present. This failure depends heavily on the used method for face detection and suffers typically from changes in illumination, pose, facial expressions and partial occlusion. As it is not possible to detect a face at every moment, the robot should at least be able to guess where the users face might be given the last known position, direction and velocity.

To handle these challenges two algorithms (Lucas-Kanade optical flow and Kalman filter) are explored in this chapter.

4.1 Lucas-Kanade

For the Lucas-Kanade method we first have to clarify the definition of optical flow. For this we observe every pixel of two frames from a video stream and analyse the movement that happens for a pixel between the observed frames. This associates a velocity to each pixel in an image and is called a dense optical flow. As it is computationally expensive to calculate the velocity field for every pixel in every frame of a video stream we look at sparse optical flow as an alternative. There we do not want to track every pixel, but a subset of points which have to be selected before the velocity field is calculated. Lucas-Kanade is one of the most popular sparse tracking method.

Lucas-Kanade Bradski and Kaehler [3] builds upon three assumptions:

Brightness constancy During the possible movement between two frames, a pixel in the image does not change its appearance. This means that in a greyscale image a pixel will not change its intensity during its transition from one frame to the next.

Temporal persistence The movement speed of a tracked object between two frames is low in comparison to the frame rate of the video stream.

Spatial coherence Points neighbouring each other belong to the same surface, thus also to the same object, and therefore have a similar or the same direction and velocity.

The first assumption leads us to the formulation of

$$f(x,t) \equiv I(x(t),t) = I(x(t+dt),t+dt) \qquad \frac{\partial f(x,t)}{\partial t} = 0 \tag{4.1}$$

The second assumption leads us to the idea that the motions between two consecutive frames are so small that we can approximate them by the derivative of the intensity with respect to time. With I_x and I_y as the spatial derivatives in the directions of x and y across the first frame, I_t the derivative between images over time and u, v as the unknown velocities between the frames equation equation (4.1) on the preceding page equates to

$$\underbrace{\left.\frac{\partial I}{\partial x}\right|_t}_{I_x} {}^{I_t} \underbrace{\left(\frac{\partial x}{\partial t}\right)_{I_t}}_{u} + \underbrace{\left.\frac{\partial I}{\partial y}\right|_t}_{I_y} {}^{I_t} \underbrace{\left(\frac{\partial y}{\partial t}\right)_{I_t}}_{v} + \underbrace{\left.\frac{\partial I}{\partial t}\right|_{x(t)}}_{I_t} = 0 \tag{4.2}$$

or

$$\nabla \mathbf{I}^{\mathbf{T}} \mathbf{u} = -\mathbf{I}_t \tag{4.3}$$

with

$$\nabla \mathbf{I} = \begin{bmatrix} I_x \\ I_y \end{bmatrix} \mathbf{u} = \begin{bmatrix} u \\ v \end{bmatrix}.$$

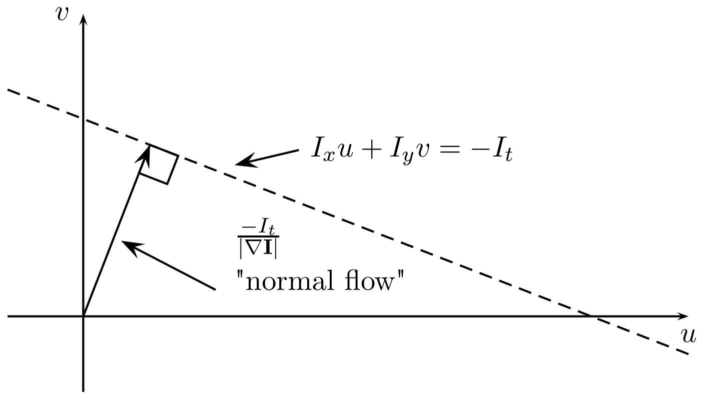


Figure 4.1: Two dimensional optical flow

As we can see this bears the problem that when we solve equation (4.3) the results for u and v from a singular pixel are not unique. Only for the normal or perpendicular flow component the equation can be solved (see figure 4.1). But this suffers from the aperture problem. This states that if the window through which motion is observed, is so small that only an edge but no corner can be detected, it is not possible to determine the exact motion of the object. This problem is shown in figure 4.2 on the next page where it is impossible to determine the exact direction and velocity by observing the small window in the middle. To overcome this fact we rely on the third assumption of Lucas-Kanade which states that if a local patch of pixels moves coherently, we can solve the equation for the motion of the center pixel by using the neighbouring pixels to set up a system of equations.

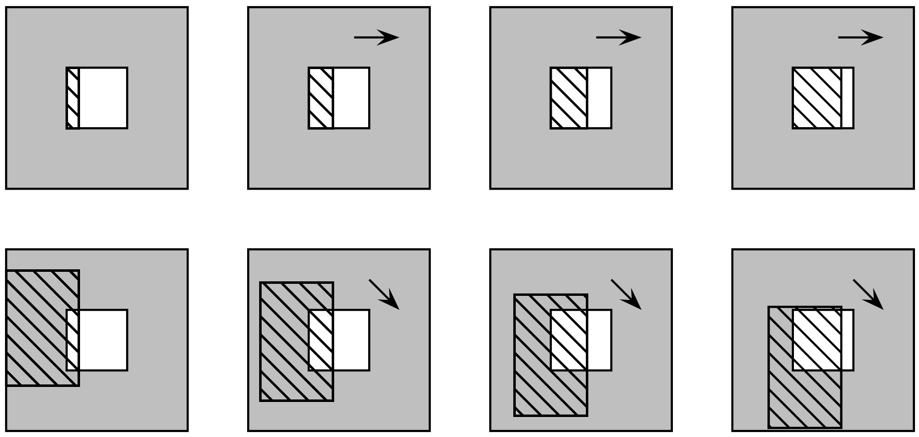


Figure 4.2: Aperture problem

By using a 5 by 5 window of intensity values for grey-scale images the set of 25 equations can be written as:

$$\underbrace{\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix}}_{\mathbf{A}\in\mathbb{R}^{25\times 2}} \underbrace{\begin{bmatrix} u \\ v \end{bmatrix}}_{\mathbf{d}\in\mathbb{R}^{2\times 1}} = - \underbrace{\begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_{25}) \end{bmatrix}}_{\mathbf{b}\in\mathbb{R}^{25\times 1}}. \tag{4.4}$$

This system is overdetermined and can be solved under the premise that it contains more than just one edge in the 5 by 5 window. We use a least-squares minimization of the equation in which $\min ||\mathbf{Ad} - \mathbf{b}||^2$ is in the standard form as

$$\underbrace{\left(\mathbf{A}^T\mathbf{A}\right)}_{2\times 2} \underbrace{\mathbf{d}}_{2\times 1} = \underbrace{\mathbf{A}^T\mathbf{b}}_{2\times 1} \tag{4.5}$$

or

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}. \tag{4.6}$$

If $\mathbf{A}^T\mathbf{A}$ is invertible the system can be solved, which corresponds to the requirement of full rank or in this case $rank\left(\mathbf{A}^T\mathbf{A}\right) = 2$. This is the case when it has two large eigenvectors which can be observed when the observation window is hovering over a corner region in the image. The resulting velocities are then solved as

$$\begin{bmatrix} u \\ v \end{bmatrix} = \left(\mathbf{A}^T\mathbf{A}\right)^{-1} \mathbf{A}^T\mathbf{b}. \tag{4.7}$$

With the results of the Lucas-Kanade method we gain the knowledge of the displacement between two frames. Using this information it is possible to first detect a face with the Viola-Jones algorithm, use pixel in the face region as initial data for the optical flow calculation. When no face is detected in the next frame the optical flow is used to follow the observed pixels in the current frame. As soon as a face has been re-detected the optical flow data is discarded and the Viola-Jones data is used to control the servo motor movement.

4.2 Discrete Kalman filter

Following the definitions of the Kalman filter from Kalman et al. [13] and Welch and Bishop [26] it describes a possible way to estimate a state $\mathbf{x}$ of a linear, time invariant, time discrete system which is described by

$$\mathbf{x}_{k+1} = \mathbf{\Phi}\mathbf{x}_k + \mathbf{\Gamma}\mathbf{u}_k + \mathbf{G}\mathbf{w}_k \qquad \mathbf{x}(0) = \mathbf{x}_0 \tag{4.8a}$$

$$\mathbf{y}_k = \mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k + \mathbf{H}\mathbf{w}_k + \mathbf{v}_k, \tag{4.8b}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the state of the system, $\mathbf{u} \in \mathbb{R}^p$ is the p-dimensional deterministic input, $\mathbf{y} \in \mathbb{R}^q$ is the q-dimensional measurement, $\mathbf{w} \in \mathbb{R}^r$ is the r-dimensional process noise and $\mathbf{v}$ is the measurement noise. In absence of either a driving function $\mathbf{u}$ or process noise the matrix $\mathbf{\Phi} \in \mathbb{R}^{n\times n}$ relates the states from step k to the next step $k+1$. $\mathbf{\Gamma} \in \mathbb{R}^{n\times p}$ is the optional control-input and $\mathbf{C} \in \mathbb{R}^{q\times r}$ is the observation model which maps the measured state $\mathbf{x}_k$ into the observed space. For the process noise and the measurement noise we assume that they are independent of each other and with normal probability distributions

$$E(\mathbf{v}_k) = \mathbf{0} \qquad E(\mathbf{v}_k\mathbf{v}_j^T) = \mathbf{R}\delta_{kj} \tag{4.9}$$

$$E(\mathbf{w}_k) = \mathbf{0} \qquad E(\mathbf{w}_k\mathbf{w}_j^T) = \mathbf{Q}\delta_{kj} \tag{4.10}$$

$$E(\mathbf{w}_k\mathbf{v}_j^T) = \mathbf{0}, \tag{4.11}$$

where the indices k and j mean two different discrete points in time. With the positive-semidefinite matrices $\mathbf{Q} \geq \mathbf{0}$ and $\mathbf{R} \geq \mathbf{0}$, the positive-definite matrix $\mathbf{H}\mathbf{Q}\mathbf{H}^T + \mathbf{R} > \mathbf{0}$ and the Kronecker delta δ_{kj}. The expected value of the state at the beginning $\mathbf{x}_0$ and the covariance matrix of the error at $k = 0$ are

$$E(\mathbf{x}_0) = \mathbf{m}_0 \qquad E([\mathbf{x}_0 - \hat{\mathbf{x}}_0][\mathbf{x}_0 - \hat{\mathbf{x}}_0]^T) = \mathbf{P}_0 \geq \mathbf{0}. \tag{4.12}$$

Furthermore we assume that the process noise $\mathbf{w}_k$ for $k \geq 0$ and the measurement noise $\mathbf{v}_l$ for $l \geq 0$ do not correlate with the start value $\mathbf{x}_0$.

$$E(\mathbf{w}_k\mathbf{x}_0^T) = \mathbf{0} \qquad E(\mathbf{v}_l\mathbf{x}_0^T) = \mathbf{0} \tag{4.13}$$

and therefore with equation (4.8a) on the preceding page

$$E(\mathbf{w}_k\mathbf{x}_j^T) = \mathbf{0} \quad \text{if } k \geq j \tag{4.14}$$

$$E(\mathbf{v}_l\mathbf{x}_j^T) = \mathbf{0} \quad \forall l, j \tag{4.15}$$

To derive the Kalman filter we define $\hat{\mathbf{x}}_k^-$ to be the a priori state estimate at step k using all the knowledge of the process until the step $k-1$ and $\hat{\mathbf{x}}_k$ to be the posteriori state estimate given the measurement $\mathbf{y}_k$. The first step in the Kalman cycle is to use the initial values for the state $\hat{\mathbf{x}}_k$ and the error covariance $\mathbf{P}_k$ to predict $\hat{\mathbf{x}}_{k+1}^-$ and $\mathbf{P}_{k+1}^-$. This process is called the Time Update or Prediction,

$$\hat{\mathbf{x}}_{k+1}^- = \mathbf{\Phi}_k\hat{\mathbf{x}}_k + \mathbf{\Gamma}_k\mathbf{u}_k \tag{4.16}$$

$$\mathbf{P}_{k+1}^- = \mathbf{\Phi}_k\mathbf{P}_k\mathbf{\Phi}_k^T + \mathbf{G}_k\mathbf{Q}_k\mathbf{G}_k^T. \tag{4.17}$$

The second step is the Measurement Update or Correction. First the Kalman gain $\mathbf{K}_k$, which is used to introduce a weight for the difference between the measurement $\mathbf{y}_k$ and the predicted state $\hat{\mathbf{x}}_k^-$, is updated. With the new $\mathbf{K}_k$ and the identity matrix $\mathbf{I}$ the error covariance matrix is recalculated:

$$\mathbf{K}_k = \mathbf{P}_k^-\mathbf{C}_k^T(\mathbf{C}_k\mathbf{P}_k^-\mathbf{C}_k^T + \mathbf{H}_k\mathbf{Q}_k\mathbf{H}_k^T + \mathbf{R}_k)^{-1} \tag{4.18}$$

$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + \mathbf{K}_k(\mathbf{y}_k - \mathbf{C}_k\hat{\mathbf{x}}_k^- - \mathbf{D}_k\mathbf{u}_k) \tag{4.19}$$

$$\mathbf{P}_k^+ = (\mathbf{I} - \mathbf{K}_k\mathbf{C}_k)\mathbf{P}_k^-. \tag{4.20}$$

In figure 4.3 on the next page the Kalman cycle is pictured, the changes in certainty of position during this cycle of prediction, measurement and update is shown in figure 4.4 on page 29. This clearly illustrates that the knowledge of the position gets more uncertain as long as no measurement occurs. After a new measurement is done the state vector $\hat{\mathbf{x}}_k$ gets updated according to equation (4.19) and resembles a higher certainty.

The Kalman filter can now be used to predict the corners of the bounding box around a detected face and predict the most probable positions of them in the next frame. For this the mathematical steps between the state variable at time k and $k+1$ have to be formulated and known.

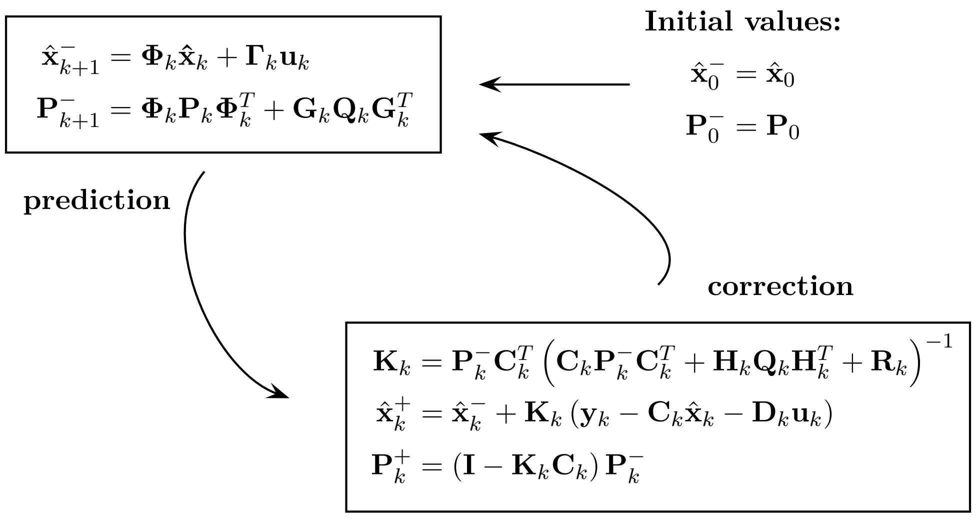

$\hat{\mathbf{x}}_{k+1}^- = \mathbf{\Phi}_k \hat{\mathbf{x}}_k + \mathbf{\Gamma}_k \mathbf{u}_k$
$\mathbf{P}_{k+1}^- = \mathbf{\Phi}_k \mathbf{P}_k \mathbf{\Phi}_k^T + \mathbf{G}_k \mathbf{Q}_k \mathbf{G}_k^T$
Initial values:
$\hat{\mathbf{x}}_0^- = \hat{\mathbf{x}}_0$
$\mathbf{P}_0^- = \mathbf{P}_0$
prediction
correction
$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{C}_k^T \left(\mathbf{C}_k \mathbf{P}_k^- \mathbf{C}_k^T + \mathbf{H}_k \mathbf{Q}_k \mathbf{H}_k^T + \mathbf{R}_k\right)^{-1}$
$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + \mathbf{K}_k \left(\mathbf{y}_k - \mathbf{C}_k \hat{\mathbf{x}}_k - \mathbf{D}_k \mathbf{u}_k\right)$
$\mathbf{P}_k^+ = \left(\mathbf{I} - \mathbf{K}_k \mathbf{C}_k\right) \mathbf{P}_k^-$

Figure 4.3: Kalman cycle

4.3 Implementation with Kalman filter

We use a Kalman-filter to predict the location of the face and to correct this prediction based on actual detections by the face detector. To correctly implement a Kalman filter the knowledge of the transition matrix $\mathbf{\Phi}$, the measurement vector $\mathbf{y}_k$, the noise matrices for the measurement $\mathbf{R}$ and the state transition $\mathbf{Q}$ are required. Equation (4.8) is adapted for our implementation. The input $\mathbf{u}$ is zero as there is no way to control the position of the face with any input from our system. The matrices $\mathbf{H}$ and $\mathbf{G}$ are set to the identity matrix. Starting with the accordingly simplified state-space model

$$\mathbf{x}_{k+1} = \mathbf{\Phi}\mathbf{x}_k + \mathbf{w}_k \qquad \mathbf{x}(0) = \mathbf{x}_0 \tag{4.21a}$$

$$\mathbf{y}_k = \mathbf{C}\mathbf{x}_k + \mathbf{v}_k \tag{4.21b}$$

under the constraints of

$$E(\mathbf{w}_k \mathbf{w}_j^T) = \mathbf{Q}\delta_{kj} \tag{4.22}$$

$$E(\mathbf{v}_k \mathbf{v}_j^T) = \mathbf{R}\delta_{kj} \tag{4.23}$$

the system is defined by the state vector
$\mathbf{x}_k = [x_{1,k}, y_{1,k}, x_{2,k}, y_{2,k}, v_{x_1,k}, v_{y_1,k}, v_{x_2,k}, v_{y_2,k}]^T$ and the measurement vector $\mathbf{y}_k = [x_{1,k}, y_{1,k}, x_{2,k}, y_{2,k}]^T$, which are the four corner points of the bounding box returned by Viola-Jones detector. In order to determine the transition matrix $\mathbf{\Phi}$ the simplified

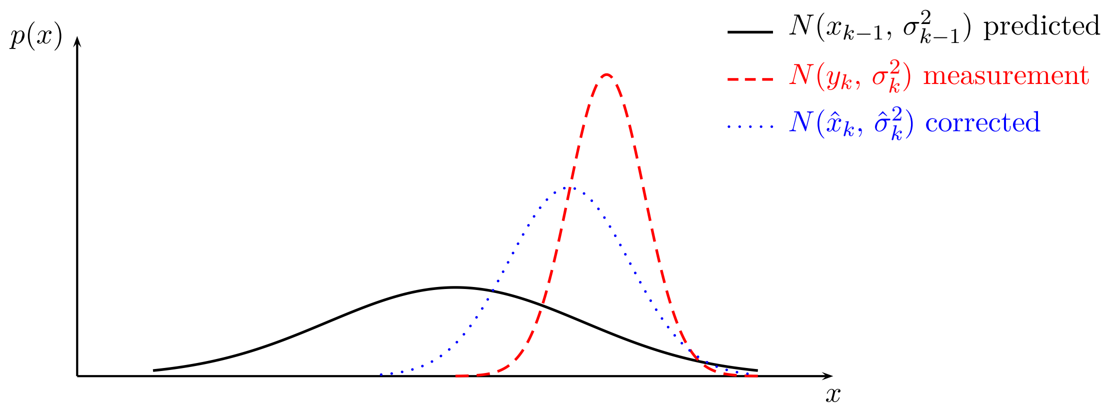


Figure 4.4: Combining prediction and measurement with the Kalman filter

equations of motion, with v as velocity and the point in space x are

$$\begin{aligned} x &= x_0 + v_x t \\ v_x &= v_{x0}, \end{aligned} \tag{4.24}$$

For y the same equations apply. With the the next step in the recursive form

$$\begin{aligned} x_{k+1} &= x_k + v_{x,k} t \\ v_{x,k+1} &= v_{x,k} \end{aligned} \tag{4.25}$$

the model is

$$\begin{bmatrix} x_{1,k+1} \\ y_{1,k+1} \\ x_{2,k+1} \\ y_{2,k+1} \\ v_{x1,k+1} \\ v_{y1,k+1} \\ v_{x2,k+1} \\ v_{y2,k+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & \triangle t & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & \triangle t & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & \triangle t & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & \triangle t \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{1,k} \\ y_{1,k} \\ x_{2,k} \\ y_{2,k} \\ v_{x1,k} \\ v_{y1,k} \\ v_{x2,k} \\ v_{y2,k} \end{bmatrix}, \tag{4.26}$$

where $\triangle t = 40\text{ms}$, limited by the cameras frame rate. The process noise matrix $\mathbf{Q}$ is chosen as

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \times 0.01. \tag{4.27}$$

The measurement matrix **C** represents the link between the measured coordinates in $\mathbf{y_k}$ and the state vector $\mathbf{x_k}$ as defined by the system in equation (4.21a) on page 28:

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{4.28}$$

The matrix **R**, the covariance of the measurement is determined empirically. From the observation that the Viola-Jones face detector detects faces within 10 pixels of the face position 95% of the time and the condition that the error is Gaussian distributed the variance in each direction is 6.5 pixels. This is represented in $\mathbf{v}_k = \begin{bmatrix} 6.5 & 6.5 & 6.5 & 6.5 \end{bmatrix}^T$ and therefore also in

$$\mathbf{R} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times 42.25. \tag{4.29}$$

The implementation is designed in the way that we first detect a face in the video stream at frame k and gain knowledge of the corner points of the bounding box around the face. These points are the measurements y_k, which are used with the initial values of the state $\hat{\boldsymbol{x}}_0^-$ and the error covariance matrix $\boldsymbol{P}_0^-$ to predict the state according to the system equation (4.8a) on page 26. This state information contains the positions of the corners for the bounding box in the next frame and can be already sent to the servo motor controller. The Kalman cycle is repeated whether an update occurs or not. If an update occurs through the means of a new measurement, the Kalman filter incorporates the new information into its calculation and returns the state vector with a heightened probability and lower variance figure 4.4 on the preceding page. If no update happens the Kalman filter still provides a state vector, but this has a high variance in which the actual corner positions should be. Whatever case occurs, the positions are sent to the servo motor control node where the movement for the servo motors is calculated and handed to the actual hardware.

5 Object classification

This process handles the task of linking an object in a 3D-scene to the object descriptor in a database. For each known object the database contains a three-dimensional model and a description (e.g. the name). Details of the object's size are not stored, as the 3D model is scalable without loss of information.

5.1 Ensemble of shape functions (ESF) descriptor

The ESF descriptor is defined as the combination of three angle, three area, three distance and one distance ratio shape functions, each of them described by a 64-bin histogram.

D2

The D2 shape function proposed by Osada et al. [20], extended by Ip et al. [10] to work on three-dimensional structures and further enhanced by Wohlkinger and Vincze [28] to work on partial point clouds is used. Osada formed the D2 shape function by sampling randomly chosen point-pairs and calculating their distances into a histogram. Ip classified the line between the points as either inside a 3D model, outside of it or a mixture of both. Wohlkinger adapted this to categorize the lines as on or off the surface spanned by the point cloud. This is done by tracing the lines with the 3D Bresenham algorithm in a voxel grid with a side length of 64. After the tracing the lines are represented in three histograms which display distances in the possible ON, OFF or MIXED states.

Line ratio

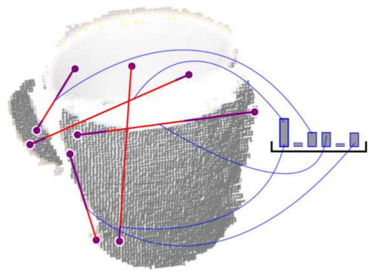
The lines described as MIXED are further used as information source by calculating the ratio from the ON to the OFF part and form an independent histogram for the ESF descriptor as seen in figure 5.1.

A3

The A3 angle shape function chooses three random points, draws two lines between them and encodes the angle between them by classification of the third line which is opposing the angle (figure 5.2a). This line can have the three states ON, OFF or MIXED.

D3

D3 area shape function is calculated by increasing the dimensionality of the D2 shape function and evaluating the area enclosed by the three randomly selected points (figure 5.2b). The classification in to ON, OFF or MIXED of the area uses the same methodology as for the D2 shape functions.


(a)

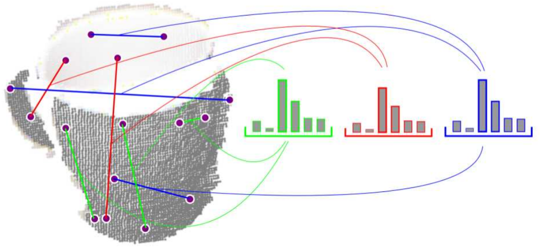
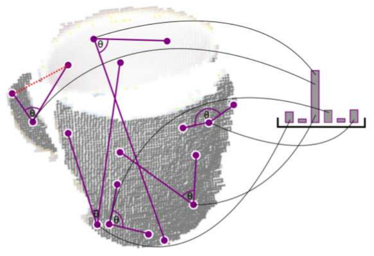

(b) D2 shape function

Figure 5.1: D2 shape and line ratio functions. Source[12]


(a) A3 shape function

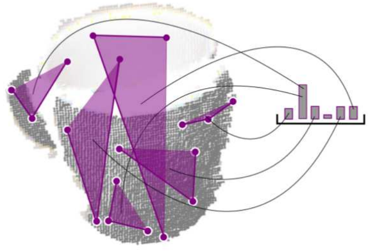

(b) D3 shape function

Figure 5.2: A3 and D3 shape functions. Source[12]

5.2 Decision forests for classification

The actual implementation uses decision forests as a method to combine weak classifiers into a strong method to predict a certain information. A basic introduction of binary decision trees and forests is derived from Criminisi et al. [6]. The requested classification of two objects, a banana and a mug, against the database is shown in figure 5.3. The green histograms are the stored values of a mug, which has been trained against the classifier earlier. The red lines denote the object which is to be classified. In the next sections the details of this tasks, training and classification, are explained in detail.

Decision tree

A decision tree consists of a set of nodes and edges connecting them together. The nodes have exactly one incoming edge and can be either split nodes or leaf nodes. Split nodes have two outgoing edges, hence the name binary decision tree, while leaf nodes are at the end of the tree and therefore only have one incoming and no outgoing edge at all. At every split node a basic decision is made, and the contestant is sent along the left or

[12]Wohlkinger and Vincze [28]

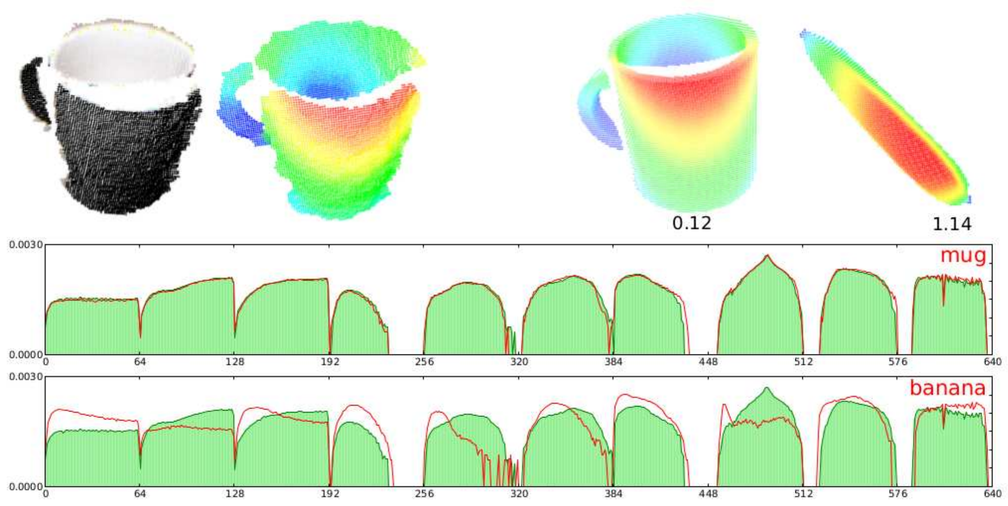


Figure 5.3: Object query against the classification. Source[12]

right outgoing edge to one of its child nodes. The principal structure of a decision tree is illustrated in figure 5.4 on the next page.

The usage of a decision tree consists of the following two parts.

Tree training

The parameters $\boldsymbol{\theta}$ of the split functions for each node are optimized so that the information gain I is maximized.

$$I = H(\mathcal{S}) - \sum_{i \in 1,2} \frac{|\mathcal{S}^i|}{|\mathcal{S}|} H(\mathcal{S}^i) \tag{5.1}$$

with the Shannon entropy defined as $H(\mathcal{S}) = -\sum_{c \in \mathcal{C}} p(c) \log p(c)$ and $\mathcal{S}$ is the set of data which is passed into the split node. Besides, stopping criteria are applied to let the tree stop growing further branches.

Tree testing

Data points $\boldsymbol{v}$ are fed into the root node of the tree, which starts applying the predefined split function $h(\boldsymbol{v}, \boldsymbol{\theta}) \in \{0, 1\}$ and sends the data to the left or right child node. This process is repeated until the data reaches a leaf node. In this node a so called leaf predictor associates the data input $\boldsymbol{v}$ with a class label.

Decision forest

An ensemble of decision trees combined together form the decision forest. The forest is adapted to the actual goal to achieve by setting up the type of split functions, the leaf predictor, stopping criteria, the ensemble model as well as the randomness model. To understand how these parameters influence the behaviour of the decision forest a short description is given.

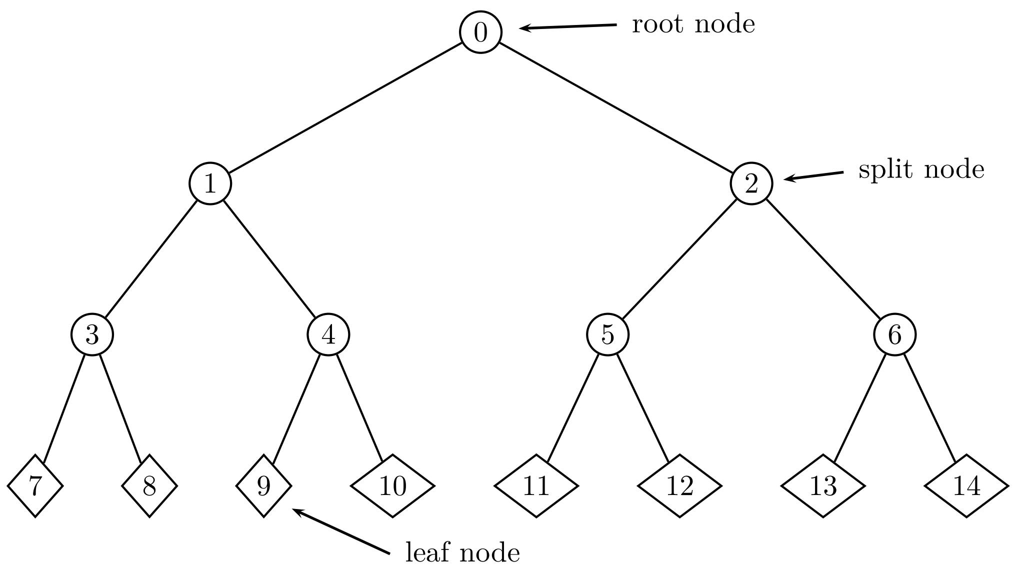


Figure 5.4: General decision tree structure

Weak learner model

is described by its parameters $\boldsymbol{\theta} = (\phi, \psi, \boldsymbol{\tau})$ where ψ is the geometric primitive used to separate the given data (a general surface in our case). The parameter vector $\boldsymbol{\tau}$ contains the thresholds for the inequalities used in the binary test and the filter function ϕ chooses a subset of features from the complete data vector $\boldsymbol{v}$. Together these parameters are used in the binary split function defined for node j as

$$h(\boldsymbol{v}, \boldsymbol{\theta}_j) \in \{0, 1\} \tag{5.2}$$

Training objective function

After choosing the split functions, the optimal parameters for them have to be obtained during training. Therefore we seek the optimal parameters $\boldsymbol{\theta}_j^*$ which maximize the information gain objective function

$$I_j = I(\mathcal{S}_j, \mathcal{S}_j^{\mathcal{L}}, \mathcal{S}_j^{\mathcal{R}}, \boldsymbol{\theta}_j) \tag{5.3}$$

$\mathcal{S}_j, \mathcal{S}_j^{\mathcal{L}}, \mathcal{S}_j^{\mathcal{R}}$ represent the training data points before the split and after split corresponding to the left and right edge respectively:

$$\boldsymbol{\theta}_j^* = \arg\max_{\boldsymbol{\theta}_j} I_j \tag{5.4}$$

Randomness model

Randomness of the individual trees is a key aspect of decision forests. It states that all decision trees in the forest are randomly different from each other. This leads to improved generalization and is in our case achieved by randomized node optimization during the training phase. This means that if $\mathcal{T}$ is the set of all possible parameters $\boldsymbol{\theta}$ we only make the subset $\mathcal{T}_j \subset \mathcal{T}$ available during the training of the j^{th} node. Therefore the training of a tree is achieved by optimizing each split node j by

$$\boldsymbol{\theta}_j^* = \arg \max_{\boldsymbol{\theta}_j \in \mathcal{T}_j} I_j. \tag{5.5}$$

The ratio $|\mathcal{T}_j|/|\mathcal{T}|$ controls the amount of randomness in the decision tree. It is common to introduce the parameter $\rho = |\mathcal{T}_j| = 1, \ldots, |\mathcal{T}|$. If $\rho = 1$ we get the maximum randomness and uncorrelated trees in the forest. For $\rho = |\mathcal{T}|$ all trees are identical and act as if only one decision tree is used.

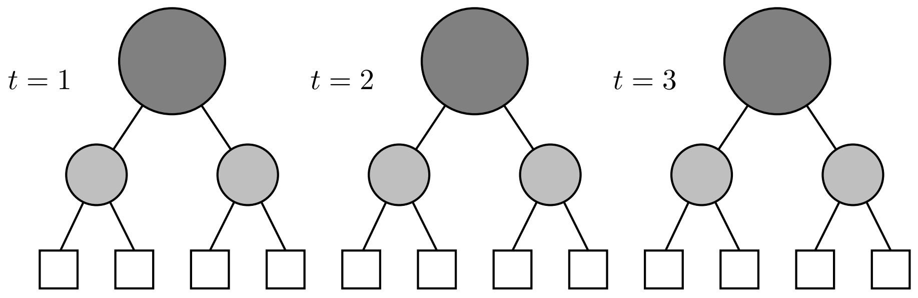


Figure 5.5: Random forest with $\rho = |\mathcal{T}|$

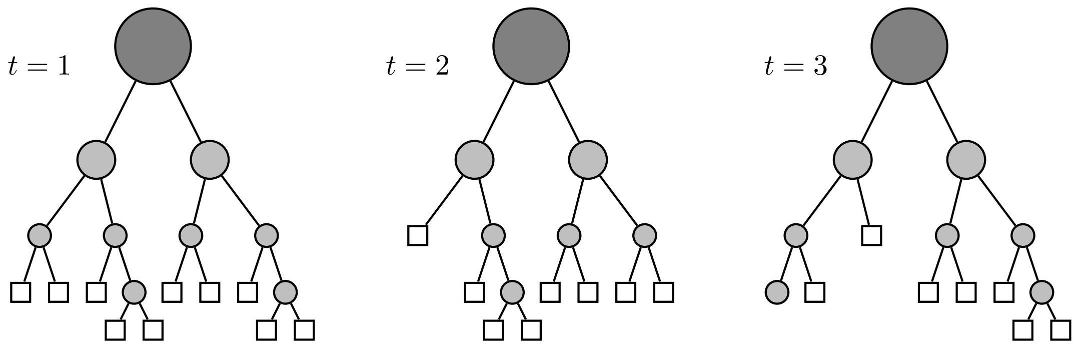


Figure 5.6: Random forest with $\rho = 1$

Leaf prediction model

Each leaf stores the empirical distribution over the classes associated to the subset of training data that reached that leaf. For the t^{th} decision tree the probabilistic leaf predictor is

$$p_t(c|\boldsymbol{v}) \quad c \in \{c_k\}. \tag{5.6}$$

Ensemble model

The ensemble model describes the method of how to combine the predictions from all the trees in the forest into a single prediction output. In classification the two most used models are the average operation or simple multiplication

$$p(c|\boldsymbol{v}) = \frac{1}{T}\sum_{t=1}^{T} p_t(c|\boldsymbol{v}) \tag{5.7}$$

$$p(c|\boldsymbol{v}) = \frac{1}{Z}\prod_{t=1}^{T} p_t(c|\boldsymbol{v}) \tag{5.8}$$

where Z is a partition function used to ensure probabilistic normalization. Both methods are heavily influenced by the most confident trees. Averaging has the effect of reducing noisy tree contributions and are therefore more robust to noise.

Stopping criteria

Stopping criteria decide when a tree should stop growing individual branches. One common way to stop the growing is after a maximum number of levels D has been reached, another is to stop after a node that contains less then a defined number of training points. Alternatively a minimum information gain can act as the requirement to further grow the branches.

5.3 Implementation

The implementation of the classification uses the point clouds obtained from the Kinect sensor and removes the supporting surface (e.g. table) under the object which has to be classified. Then it tries to build one cluster by connecting the remaining data points, then the ESF descriptor is calculated. The histograms of the descriptor are stored in the data vector $\boldsymbol{v}$ which is afterwards fed into the decision forest. During the training stage the label of the object is known and assigned to the leaf nodes which are reached after $\boldsymbol{v}$ has been inserted at the root node. The forest then returns the label of the class with the highest probability and the probability itself.

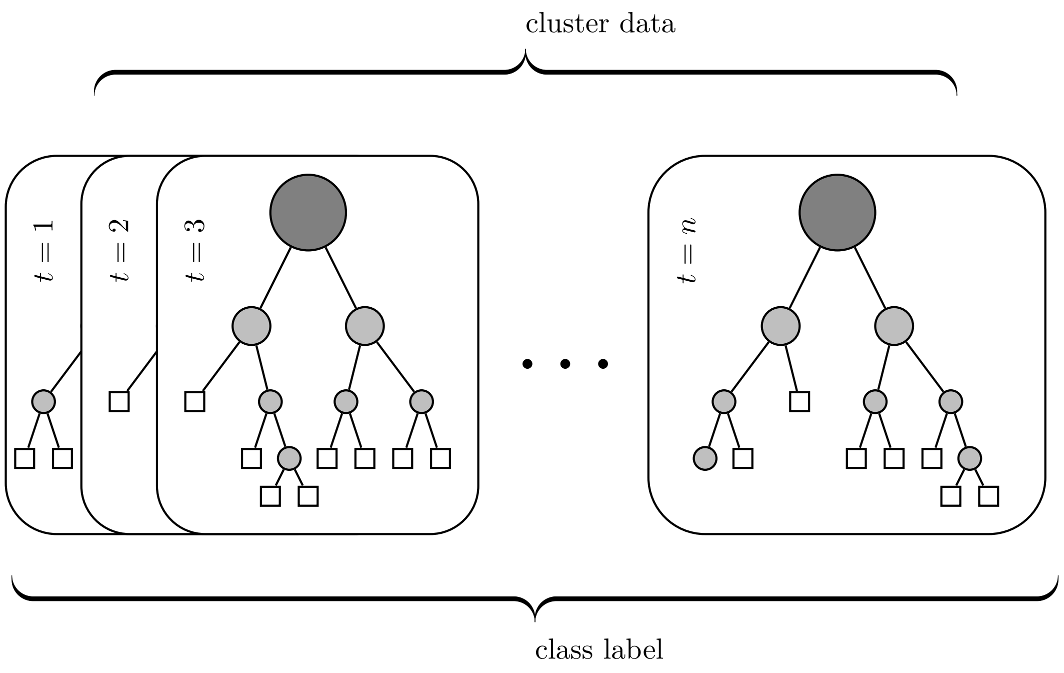


Figure 5.7: Usage of ESF and a decision forest

6 Speech recognition

Humans easily learn the concept of speech and words during the early years of their life. They learn their native language by training over the period of years while growing up as part of their childhood. As one of our goals is to establish a natural human robot interaction, Eva needs the ability to understand at least parts of natural human speech. The english language was chosen as main requirement for the recognition, with German as a secondary goal. The technical learning of speech recognition differs a lot from how a human obtains this ability. The mathematical methods to translate spoken words into machine readable text are covered in the next section. At the end of this chapter details about the chosen method and details from the implementation are further described.

6.1 Speech recognition

Prior to the implementation of this essential module we had to consider the following necessities:

Recognized languages
The minimum requirement of supported languages is English and German. English as it is the universal language in the scientific community and german as this software is likely to be used in demonstrations and applications for German native speakers without the knowledge of english language. More supported language are no requirement, but act as a bonus for the chosen system.

Programming language
As the ROS software stack gives the possibility to use either Python, C++ or to some extent Java only programs or development tools with the ability to work with one of these programming languages could be used.

Offline vs. online recognition
In the speech recognition part one of the major design decisions is whether the recognition should be processed online, which moves the computational workload to remote servers, or offline, where the device doing the speech to text processing does not need to connect to an external source.

Avoiding teach-in
Classical speech to text software modules heavily relies on a process called teach-in. During this a user who wishes to be recognized at a later point in time has to read a text passage to the speech recognition system until either the system is trained to recognize the voice of the talking person or, the speaker has adapted

his or her way to talk while speaking to the recognition system. As this process is often slow, not very accurate and not suitable if multiple persons should be able to talk to the robot without much preparation the selected process should not need any kind of teach-in.

Cost

As one of the major goals of the complete robotic system is to be cost effective it should limit the amount of money spent on the speech recognition software. If possible free or open source software is preferred.

Operating system

The system should not require the installation of a proprietary operating system. This helps to reduce costs as well as to filter out text to speech software which has been mainly designed as a semi-automated dictation system.

6.2 Mathematical background

The task of translating a spoken word into computer readable text is a statistical problem. To calculate it we follow the explanations from Mathew [17], Ittichaichareon et al. [11], Jurafsky and Martin [12] and Logan [16].

$$\begin{aligned}\hat{W} &= \arg\max_{W} P(W|Y) \\ &= \arg\max_{W} \frac{P(Y|W)P(W)}{P(Y)}\end{aligned}$$

with $W = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m\}$ is a sequence of m words and $Y = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n\}$ is a sequence of m acoustic vocal observation vectors. $P(Y|W)$ is the probability of the acoustic vector Y when the word W is given, $P(W)$ denotes the probability with which the word sequence W occurs in the written language and $P(Y)$ is the probability with which the acoustic vector sequence Y occurs in the spoken language. Therefore $\hat{W}$ is the particular word sequence W which has maximum a posteriori probability given the observation sequence Y. As $P(Y)$ is independent of the spoken word sequence $\hat{W}$ can be calculated without $P(Y)$. This leads us to

$$\hat{W} = \arg\max_{W} P(Y|W)P(W). \tag{6.1}$$

For the evaluation of equation (6.1) we need $P(W)$ which we generate from a language model and evaluate an acoustic model for $P(Y|W)$.

Preprocessing

This process consists of five main steps to compute the Mel-Frequency Cepstral Coefficients (MFCCs) [European Telecommunications Standards Institute][9], Logan [16] Mathew [17] and is shown in figure 6.2 on page 41. These steps are:

Divide the audio signal into frames

This is done by splitting a 1 second time period into 100 overlapping timeslots of typically 25 ms. It consists of a 10 ms long time frame with 7.5 ms from both the previous and the next time slot. A first-order filter i s applied afterwards, which corresponds to the transformation as $Y_0[n] = x[n] - \alpha x[n-1], \quad 0.9 \leq \alpha \leq 1, \quad 0 < n < samples\,per\,frame$. A Hamming window is used to reduce FFT leakage in the next step.
$Y_1[n] = x[n] \times H[n]$, $0 < n < framesize$ with $H[n] = c_1 - c_2 \times cos(\frac{2\pi n}{framesize-1})$ and c_1, c_2 chosen appropriately (e.g. $c_1 = 0.54$ and $c_2 = 0.46$ as used in [European Telecommunications Standards Institute][9] and Mathew [17])

Calculate the Fourier transform

Pad the frame with $zeros$ so that the frame size N is a power of two. Calculate $Y_2 = DFT(Y_1)$ and $Y_3[n] = real(Y_2[n])^2 + imag(Y_2[n])^2$, $0 < n \leq N/2$ so that the results are real numbers.

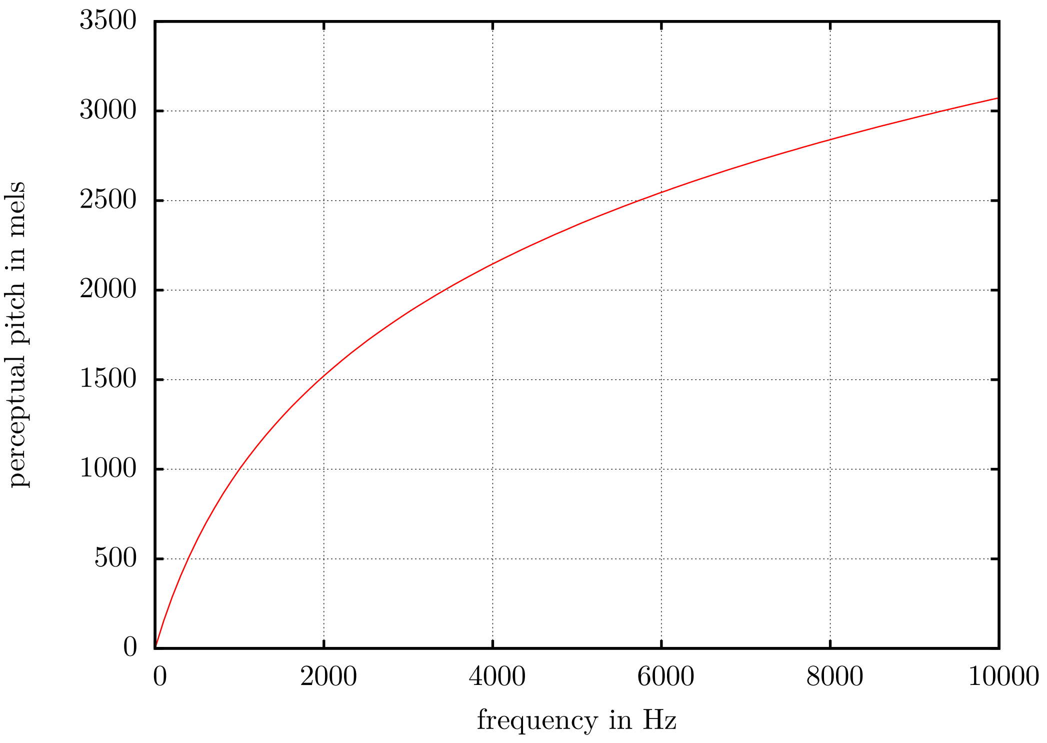


Figure 6.1: Plot of the Mel-scale

Map the spectrum onto the Mel-scale

With a set of usually 40 triangular filters the frequency spectrum after the FFT (Fast Fourier Transform) is smoothed and perceptually meaningful frequencies are emphasized. This is done by projecting the spectrum onto the Mel-frequency scale which is shown in figure 6.1 on the preceding page. The Mel-scale is used as the human interpretation of pitch rises with the frequency and was experimentally derived in the 1940's by Stevens et al. [23].

$Y_4[n] = \sum_{i=0}^{N/2} Y_3[i] \times MelWeight[n][i],\ 0 < n < number\,of\,filters$ The Mel-scale is chosen as studies have shown that low frequencies are perceptually more important in speech than high frequencies. Therefore the center frequencies of the triangular filters are placed so that the distance between the bins in the spectrum are equidistant on the Mel-scale.

Calculate the Log Compression

As the analysis against the acoustic model requires a near Gaussian statistical distribution we calculate the natural logarithm value after the projection onto the Mel-scale. $Y_5[n] = \ln Y_4[n],\ 0 < n < Number\,of\,filters$

Compute the Discrete Cosine Transform (DCT)

The DCT is used to compress the results into a set of low order coefficients which we call the Mel-cepstrum and consists of 13 coefficients (MFCCs), $Y_6 = DCT(Y_5)$. For further analysis we also compute the first and second derivatives of the MFCCs.

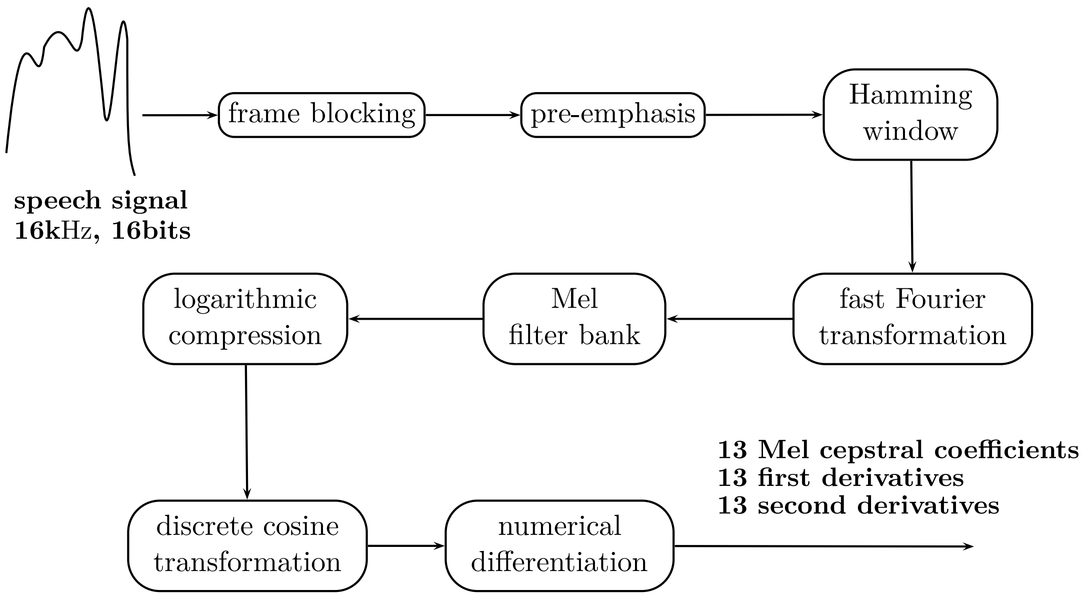

frame blocking
pre-emphasis
Hamming window
speech signal 16kHz, 16bits
logarithmic compression
Mel filter bank
fast Fourier transformation
13 Mel cepstral coefficients
13 first derivatives
13 second derivatives
discrete cosine transformation
numerical differentiation

Figure 6.2: Speech signal processing

Acoustic model

To calculate the probability of an acoustic vector sequence Y for a given word W basic sounds which combined together build words, so called phonemes, are used. Approxi-

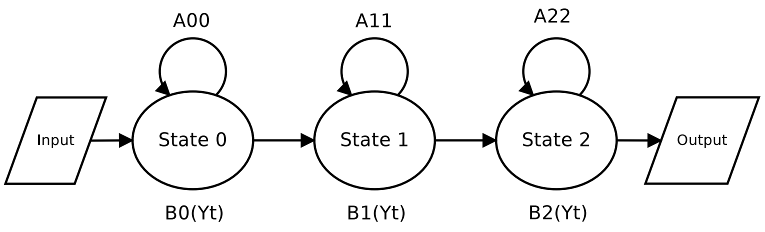


Figure 6.3: Triphone Hidden Markov model

mately 50 can be used to build every word of the English language. Spoken phonemes are dependant on the previous and the following phonemes. According to Jurafsky and Martin [12] this leads us to triphones which are used to represent the context in which a phoneme can occur. At the end of a word a phoneme for silence has to be added. With about 50 phonemes there can be 50^3 triphones, from which only 26.000 actually occur in the English language. The probability that an acoustic vector sequence corresponds to a particular triphone may be estimated using a Hidden Markov Model (HMM). Current speech recognizers use HMM models with three internal states and one entry and one exit state, as shown in figure 6.3.

A HMM model is a probabilistic finite state machine that generates observation sequences. If the model is in state S_i at timestep t, then it has a probability $B_i(Y_t)$ of producing the acoustic vector Y_t and it switches to state S_j with probability A_{ij}. The problem of computing $P(Y|W)$ now becomes what is known as the evaluation problem for HMMs - the problem of estimating the probability with which a given HMM could have generated the observation sequence Y. The evaluation problem can be solved using the Forward/Backward algorithm for HMMs, but since the optimal state sequence is needed at a later stage, it is common to do a more expensive Viterbi search which can compute the probability and uncover the optimal state sequence simultaneously (Mathew [17] and Rabiner [22].)

Language model

A language model assigns a probability to each word in a given sequence. Models which calculate the probability of a word with respect to its prior context are called N-gram models and predict the probability of a word based on the last $N-1$ words. Commonly used are bigram ($N=2$) and trigram ($N=3$) models. In order to train such an N-gram model the probability is estimated as

$$P(w_n|w_{i-N-1}^{i-1}) = \frac{C(w_{i-N+1}^{i-1}w_n)}{C(w_{i-N+1}^{i-1})} \tag{6.2}$$

where C is the relative frequency or count in which the word sequence w^k occurs. To train the language model equation (6.2) is used on a large vocabulary. Given the fact, that there is a high chance that not all possible trigrams are in the training data the probabilities of bigrams or unigrams ($N = 1$) are used in these cases instead. This approach is called *back-off* and recalculates the probabilities by multiplying with *back-off weights* α to account for the fact that higher N-grams have not been found by the training algorithm. Katz [14] extended the language model to:

$$P(w_i|w_{i-N+1}\cdots w_{i-1}) = \begin{cases} d_{w_{i-N+1}\cdots w_i}\frac{C(w_{i-N+1}...w_{i-1}w_i)}{C(w_{i-N+1}\cdots w_{i-1})} & \text{if } C(w_{i-N+1}\cdots w_i) > k \\ \alpha_{w_{i-N+1}\cdots w_{i-1}}P(w_i|w_{i-N+2}\cdots w_{i-1}) & \text{otherwise} \end{cases} \tag{6.3}$$

where $d = \frac{r'}{r}$ is the discount coefficient found through Turing's discounting in which we set $r' = r^* = (r+1)\frac{n_{r+1}}{n_r}$ with n_r as the number of words which occured exactly r times in the sample text. k is chosen as $k = 5$ by Katz.

α is then calculated with auxiliary variable

$$\beta_{w_{i-N+1}\cdots w_{i-1}} = 1 - \sum_{\{w_i:C(w_{i-N+1}\cdots w_i)>k\}} d_{w_{i-N+1}\cdots w_i}\frac{C(w_{i-N+1}...w_{i-1}w_i)}{C(w_{i-N+1}\cdots w_{i-1})}$$

so that

$$\alpha_{w_{i-N+1}\cdots w_{i-1}} = \frac{\beta_{w_{i-N+1}\cdots w_{i-1}}}{\sum_{\{w_i:C(w_{i-N+1}\cdots w_i)\leq k\}} P(w_i|w_{i-n+2}\cdots w_{i-1})}. \tag{6.4}$$

Equation (6.3) states that if the N-gram occurred more than k times during the model training the conditional probability of a word given its previous context is proportional to the maximum likelihood estimation of that N-gram. Otherwise it is equal to the back-off probability of the (N-1)-gram. After the training of both the acoustic model and the language model it is possible to compare a given vocal input against them and gain the word with the highest probability to be the textual representation of the spoken word in the language of the language model.

6.3 Implementation

As it would extend the boundaries and time for this thesis to write a speech-recognition module the decision was made to evaluate multiple vendor products. At the end of the search for a speech recognition software that could possibly fit the requirements a list of three possible contestants was found. These were evaluated against each other with regard to their accuracy, repeatability of results and handling of multiple users. The three systems are PocketSphinx [13], Nuance Dragon Mobile SDK [14] and Google Voice[15]. All three of them can run on Google's Android platform, but not necessarily on another operating system with ROS support like GNU/Linux. Therefore the next step was to implement an Android based ROS node which handles the voice input, processes the recognition with one of the libraries and evaluate the outcome of this. Details and results for the evaluation of the three programs can be found in section 7.3 on page 58.

[13]Carnegie Mellon University http://cmusphinx.sourceforge.net
[14]http://www.nuancemobiledeveloper.com
[15]https://code.google.com/p/google-voice-typing-integration/

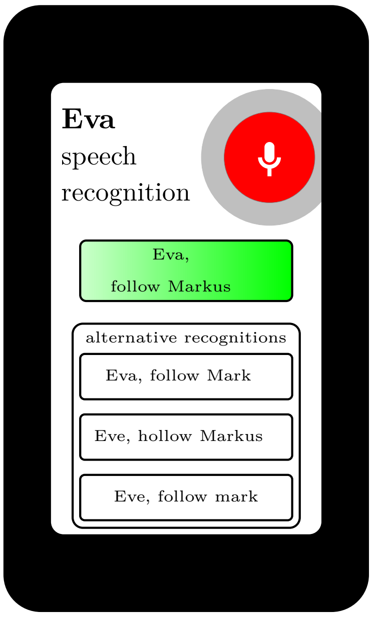


Figure 6.4: Speech recognition application

The speech recognition runs on any Android based phone or tablet from version 4.0 (Codename: Ice Cream Sandwich) and upwards, lower versions may be compatible, but have not been tested. The application we developed is split into three parts. On the first screen you have the choice of the languages 'English' and 'German' and to which ROS master node you wish to connect. On the next screen you see the last recognized sentence or a default text that has been sent as a ROS topic message to the rest of the software running on the robot in a green highlighted text box. Below the alternative recognition results are shown and can be chosen by the user if one of them is a better match for his or her spoken sentence. At this moment this is a one way communication from the tablet to the robot Eva. It could be adapted to provide a text-to-speech (TTS) feedback to the user in front of the tablet by publishing the text which the robot speaks onto a ROS topic. The application on the tablet would listen to this message and perform a TTS to play back the response to the user on its speakers. This would be useful in a scenario where the user with the input tablet is not in the same room as Eva or out of hearing range. The recognition process is started on a screen, which concept design is shown in figure 6.4. When this process is started it will try to automatically detect the start and end of a sentence and present the user with the four most likely recognition results. The result with the highest probability will be automatically sent to the robot, but the user can select one of the other recognition results to be sent to Eva. There it will be processed and if a command is recognized the SMACH state machine will act accordingly.

7 Experiments

7.1 Evaluation of face relevant implementations

To evaluate the different methods of face detection, recognition and tracking the following approach was chosen.

- A group of ten individuals was recorded in a room of the local students union of the faculty of electrical engineering and information technology. This room was chosen to evaluate the problems of background images that can mislead the Viola-Jones face detector. By avoiding a clean room, without any background other than a consistent coloured wall, the evaluation is closer to real life applications, as a user can not be expected to remove their decorations or furniture from a room just to be found by a robot.

- The recordings, which include two-dimensional image data and point clouds of the scene, were recorded with the rosbag utility. During the evaluation we observed with the rosbag application that the images and the point clouds were not synchronized, as the OpenNI Kinect driver and the rosbag utility did not manage to record them in a synchronous fashion.

- The two-dimensional image stream was then exposed to the traditional Viola-Jones face detector as described in section 3.1.1. The result of the detector, the lower left and the upper right corner of the largest detected face is then pushed into the Kalman Filter. Anticipating the results from this section we had to acknowledge that the Kalman filter did provide the needed accuracy to track a face and follow it with Eva's head, but not for a robust extraction of the face for the face recognition. Especially during the database training every false detection would lead to the fact that the face classes would overlap with data from the same false positive measurements from the face detector. Such data can be observed in frame 448 of figure 7.1a, frame 211 of figure 7.3a and frame 007 of figure 7.6a. The resulting x and y positions of this approach for test subjects Andreas, Davor and Manuel are in figure 7.7, figure 7.8 and figure 7.9. In figure 7.1a, figure 7.2a and figure 7.3a chosen frames for this method are shown.

- The point cloud and the two-dimensional images are fed into the *people_perception* module by IPA. This module have been partially introduced in section 3.2.4 and the results are compared to the two-dimensional only detection and tracking in figure 7.1, figure 7.2, figure 7.3, figure 7.4, figure 7.5 and figure 7.6. It reduces the search area for the face detector by using a Haarcascade, trained to detect heads instead of faces, on coloured point clouds. This provided a stable detecttion of a head, which could then be used for tracking, face-detection and recognition. The problems that we experienced with the 2D tracking during the training phase did not occur, as the detection on the smaller regions yielded a far lower false positive detection rate. Therefore the face recognition was only continued with this mixed 3D and 2D approach.

The approach with two-dimensional images led to the expected results. The Viola-Jones face detector had problems detecting a face in some frames, often jumped to random spots in the image, like a picture on the wall, and could not handle rotation of the head. The applied Kalman filter did a fairly good job at keeping up with the actual face position, but was mislead through the false positive measurements from the face detector. As these measurements are used to update the Kalman filter the results drifted away from the actual face position with every frame in which a false positive was reported. This can be observed in figure 7.4a, frame 287. The green rectangle is the result from the face detection, the red one is the position derived from the Kalman filter. As a false positive was delivered for a long period of time, about 50 frames in this case, the Kalman filtered result obviously drifted into this wrong position. Other occurrences of wrong positives by the Viola-Jones face detector are shown in frame 135 of figure 7.5a, frame 141 in figure 7.2a and 422 of figure 7.6a. To a human the region of the images have no resemblance to a persons face, but for the Haar filter cascade the proportions match. Good examples of the evaluation of the Haar like features at different sizes are in frame 448 of figure 7.1a, frame 31 of figure 7.2a and frames 135 and 493 of figure 7.5a. Plots of the face position over time are split in x position over time and y position over time. The positions are from the calculated center point of the detected faces.

The point cloud enabled tracking is pictured in the right handed subfigures (b) of the figures 7.1 up to 7.6. In contrast to the 2D only method multiple regions are marked by rectangles by the detector chain. The first stage, the head detector, marks any region in a light blue rectangle in which a head is discovered by the Haar cascade filter appropriately. This regions are now subject to the next stage, an implementation of the Viola-Jones face detector. If this stage found a face in one of the head regions it is highlighted by a surrounding yellow rectangle and handed to the third stage in which the face recognition is performed.

If the face recognition stage recognizes a face and assigns a label to its region, the surrounding rectangle will change the color to green to indicate the successful identification.

From now on the face recognition and the face detection stage can fail to provide a positive result in the next frames, as the label of the recognized person will be assigned to the head region in which the face was detected in the first place. Furthermore the head regions rectangle will change its colour to a dark blue.

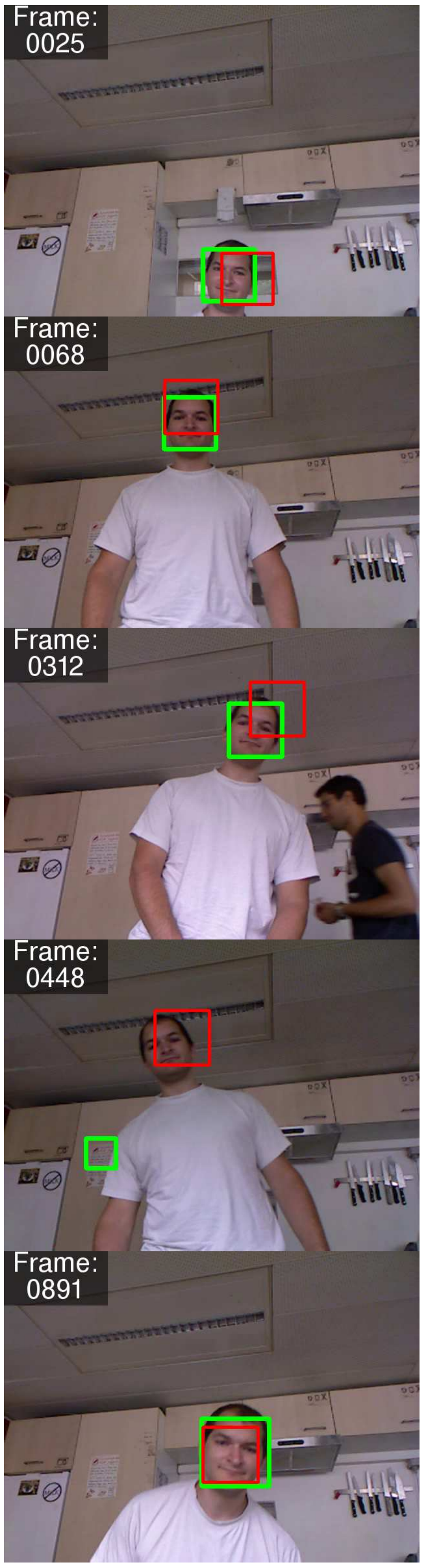
Whenever the face recognition is not able to classify a face into one specific class it will show the label 'Unknown'. This is the most likely outcome when the database has only one users data stored and another individual should get recognized.

The last possibility from the face recognition stage is that the detected face is outside of the mean face class. This will yield the result that no face recognition is possible and that the label 'No face' gets assigned to the region.

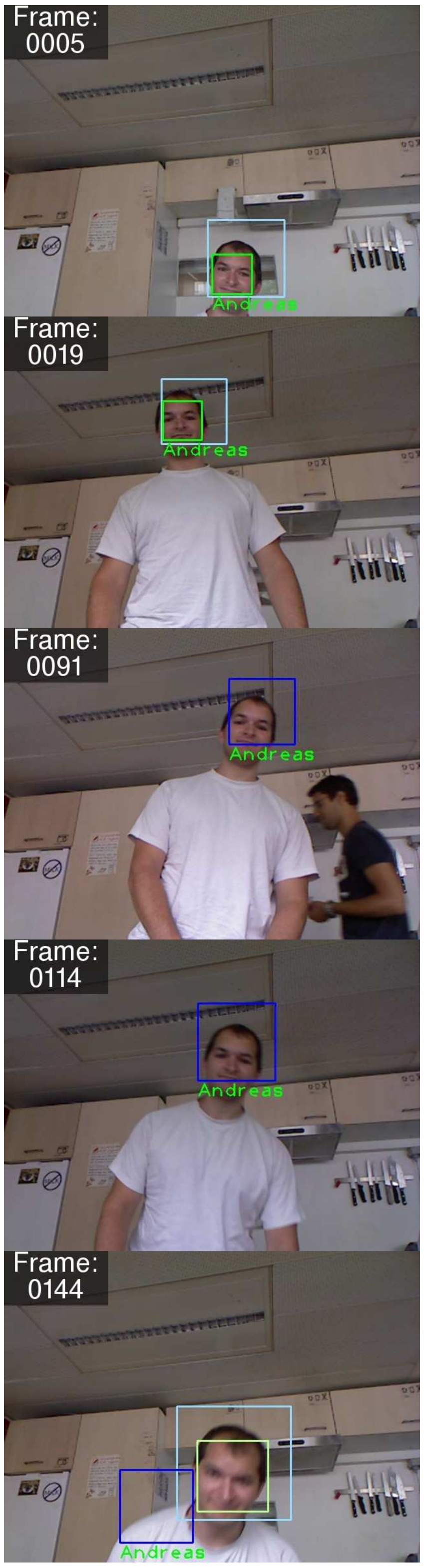
The evaluation of the two-dimensional face tracking algorithms showed that the Kalman filter provided us with a result within a small region of the actual face position as long as the velocity and direction do not change. If they do change the Kalman filter would constantly loose accuracy as long as no measurement update occurs. The weakness of the Kalman filter is that it relies solely on the measurements of the face detection. Whenever false positive detections occur the Kalman filter is mislead.

For the face tracking part of this work the Kalman filter provided better results as the Lucas-Kanade optical flow algorithm. As we recall the optical flow rely on three assumptions, which could not be met at all the time. A good example for this is the fact that during a head rotation, in a fashion that the user first looks frontal into the camera and then to the side in one smooth motion. During this movement the lighting condition will most likely change, because the primary light source in a room is at a fixed point in space during such an act. Therefore assumption is not fulfilled. The same applies to the assumption that neighbouring points move with the same speed, which can be assumed in a two-dimensional space, but not in the three-dimensional one. This can be thought through for a few significant points of a head (e.g. nose, ears) which will at different speeds. Therefore this method was not further evaluated, as the alternatives were not limited in such ways.

During training of the face recognition we observed that two out of the ten recorded rosbag data files did not contain enough information to learn their faces into the classification database. The data of these test subjects were therefore only used for the evaluation of face tracking. The standard implementation using the Eigenface method in the *people_perception* module provided a positive recognition at more than 90% of the trials, the two failed candidates not included, without further modifications.

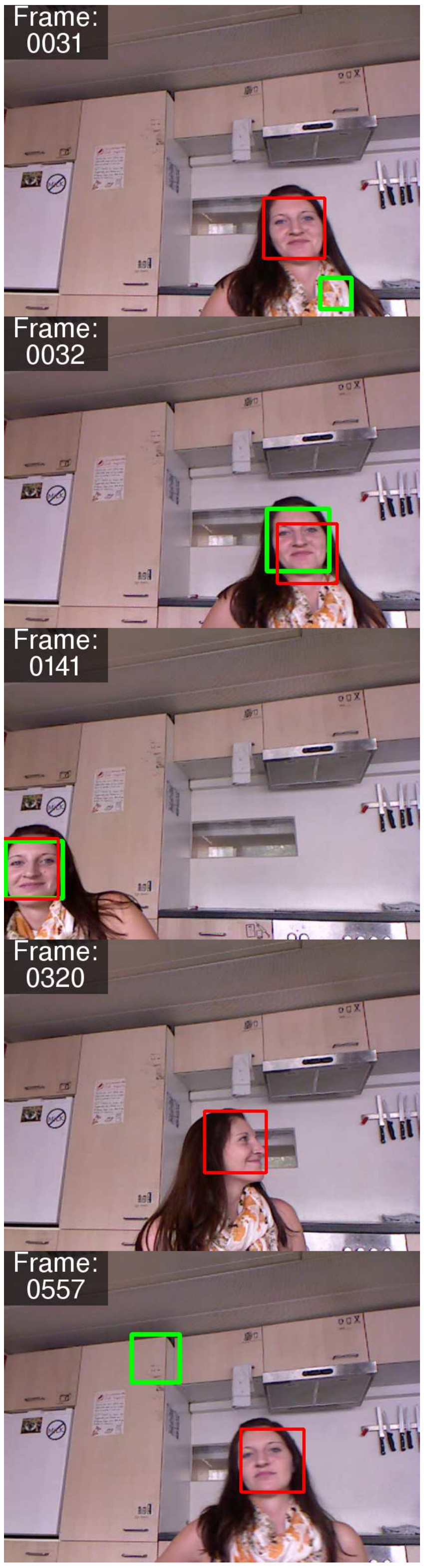


(a) 2D detection and Kalman filtered tracking

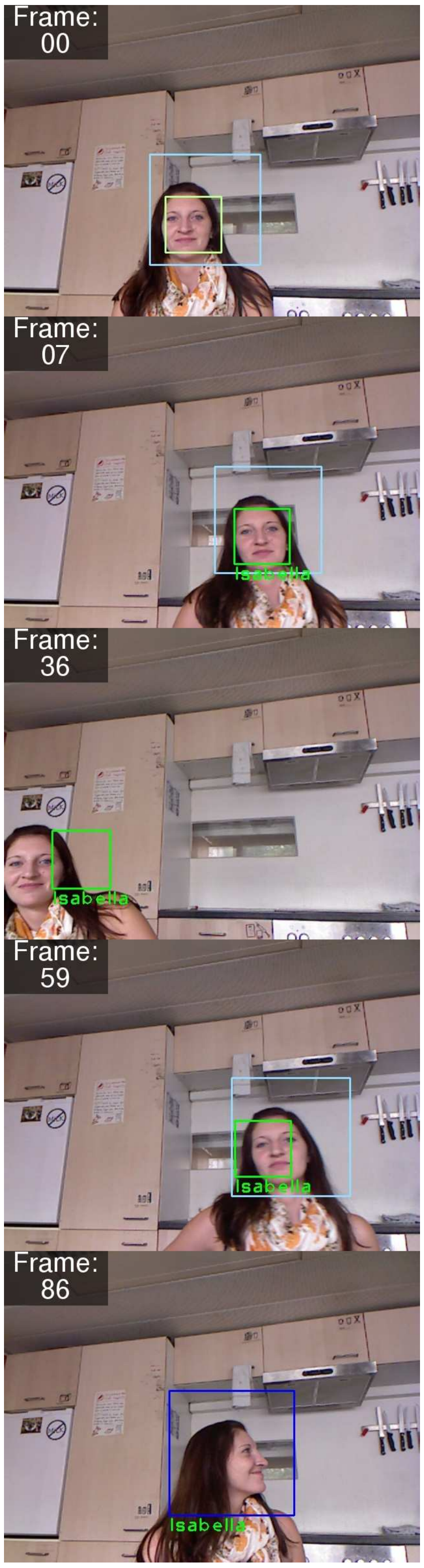


(b) 3D head and face detection

Figure 7.1: Tracking and recognition of Andreas

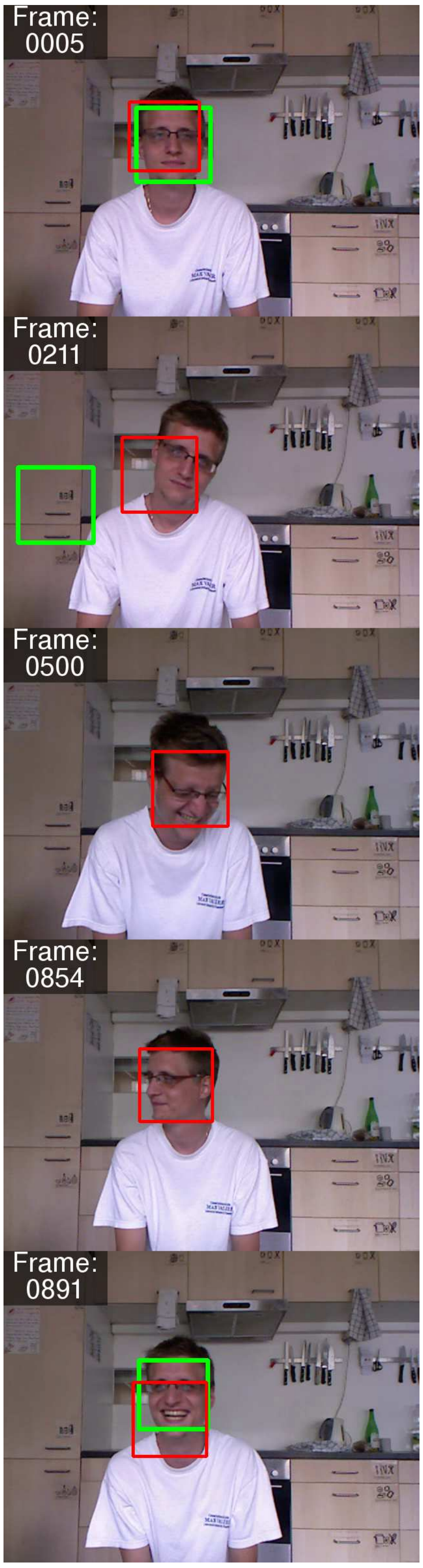


(a) 2D detection and Kalman filtered tracking

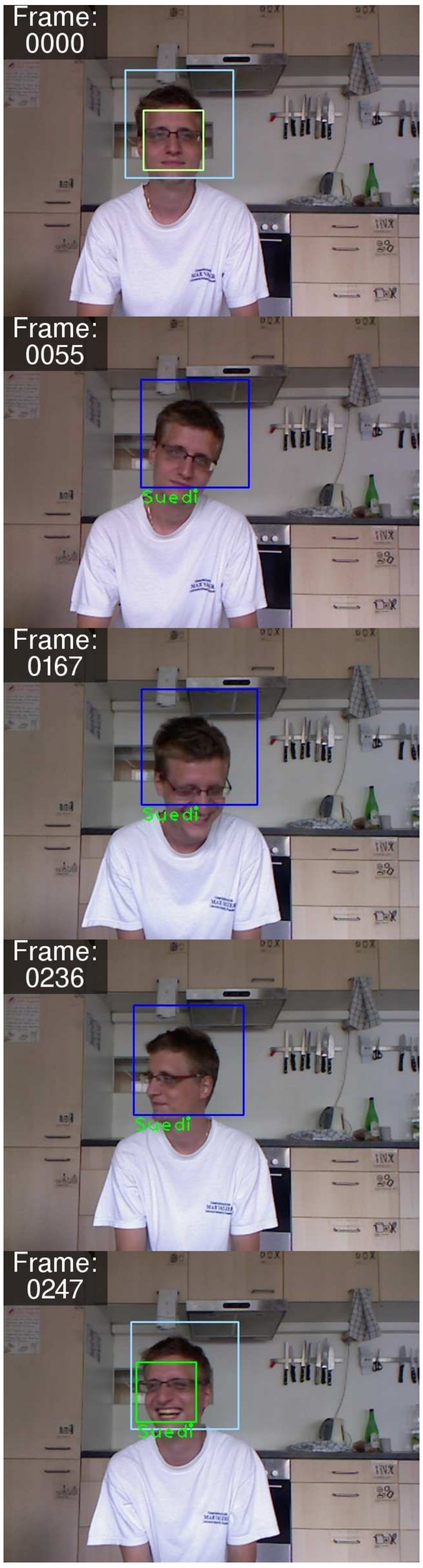


(b) 3D head and face detection

Figure 7.2: Tracking and recognition of Isabella

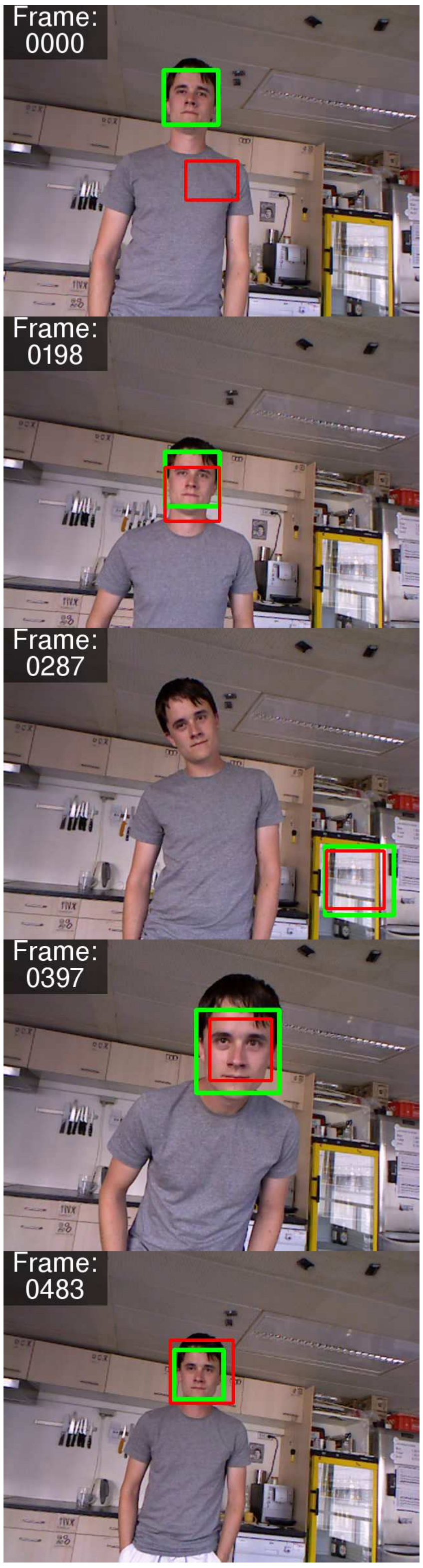


(a) 2D detection and Kalman filtered tracking

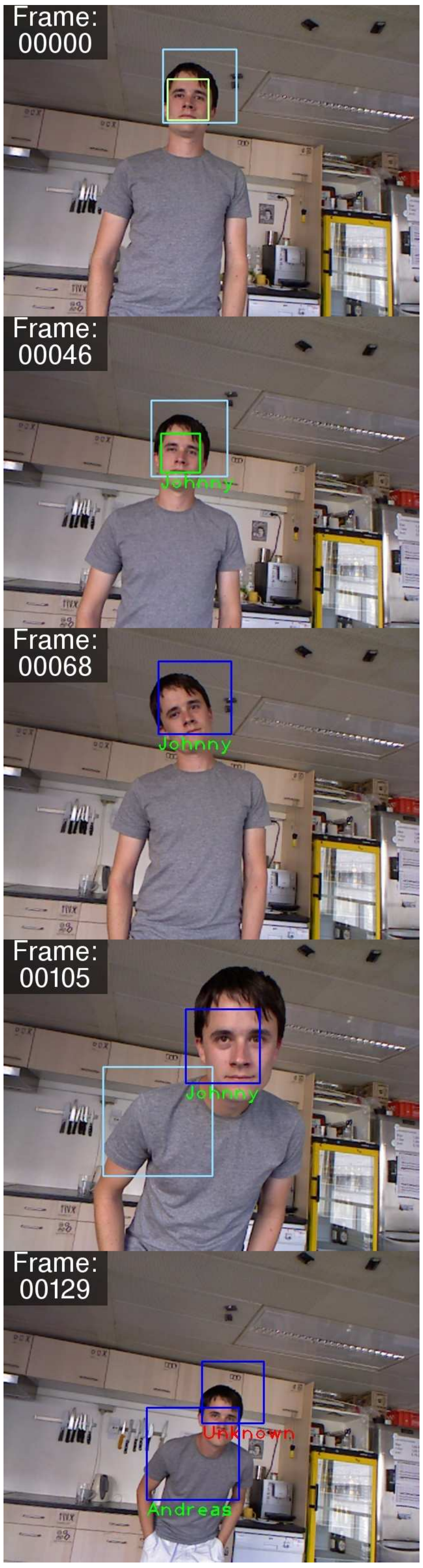


(b) 3D head and face detection

Figure 7.3: Tracking and recognition of Suedi

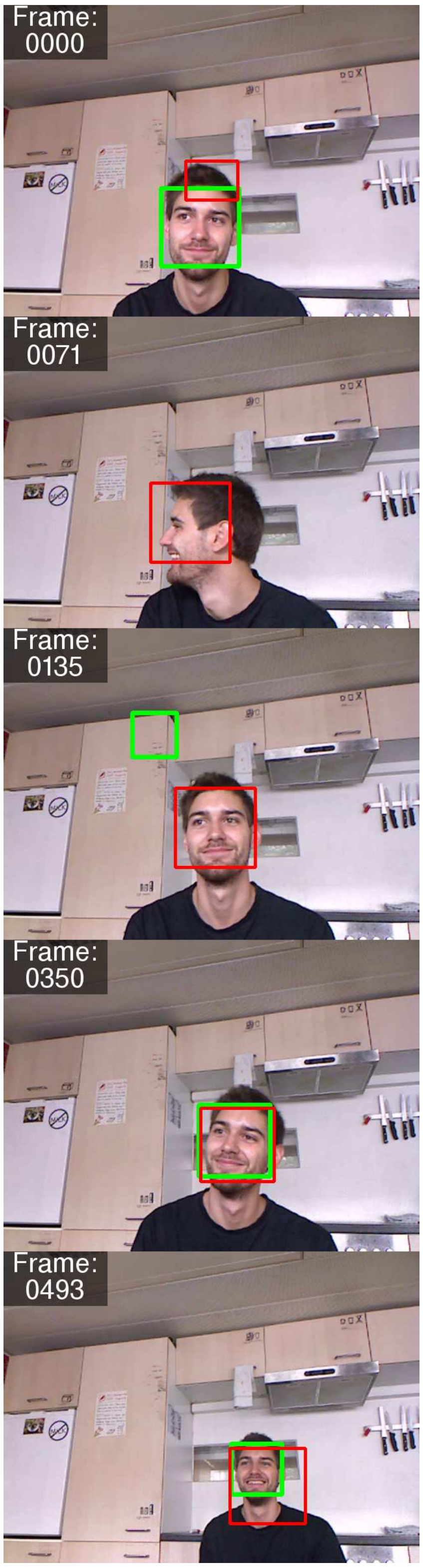


(a) 2D detection and Kalman filtered tracking

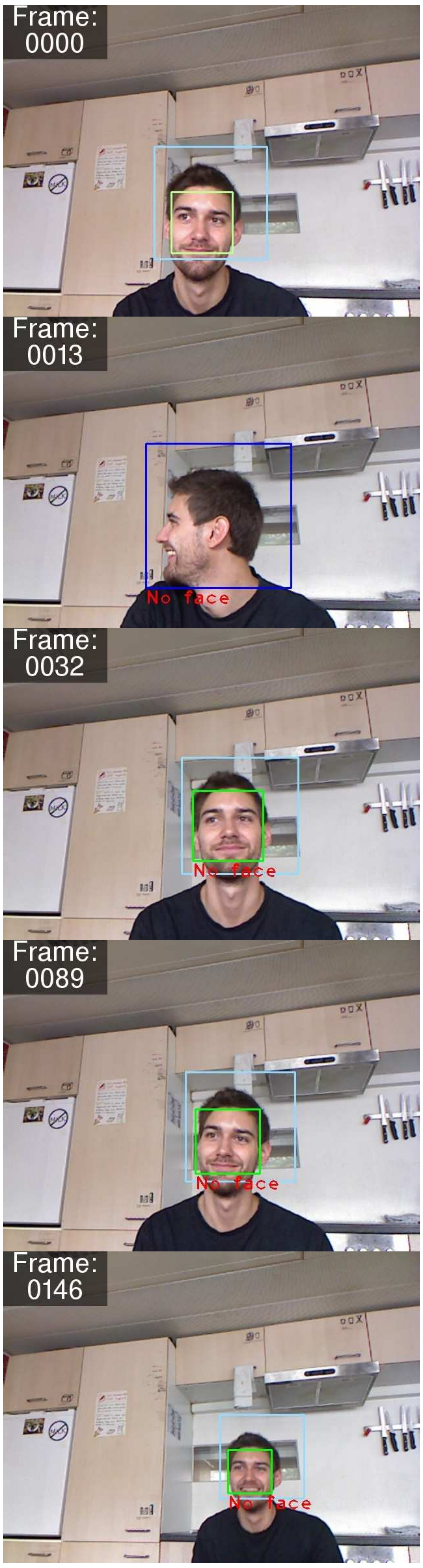


(b) 3D head and face detection

Figure 7.4: Tracking and recognition of Johnny

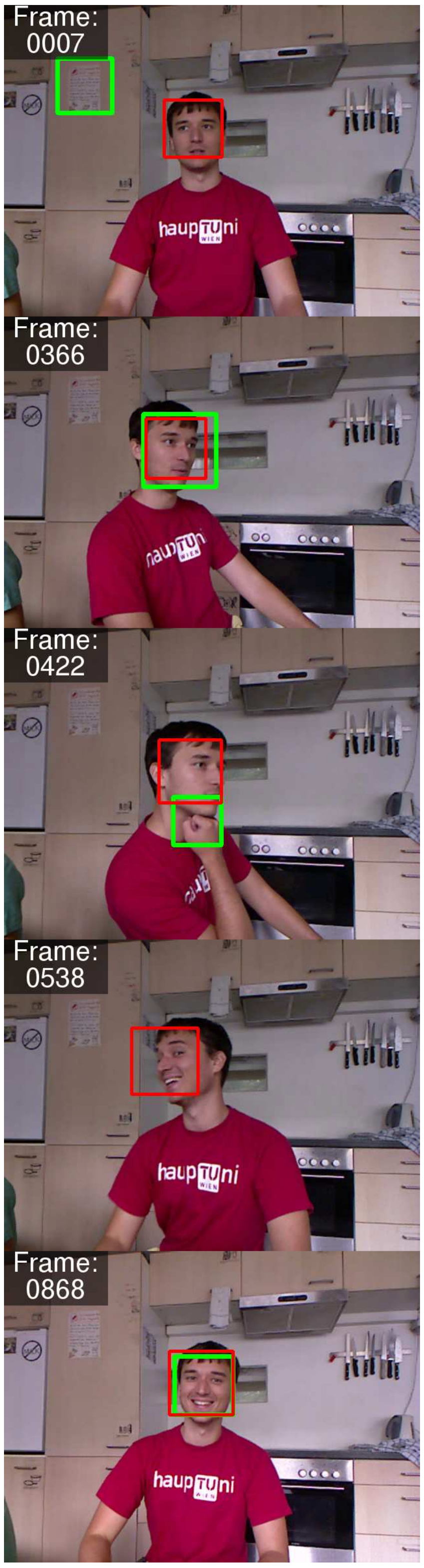


(a) 2D detection and Kalman filtered tracking

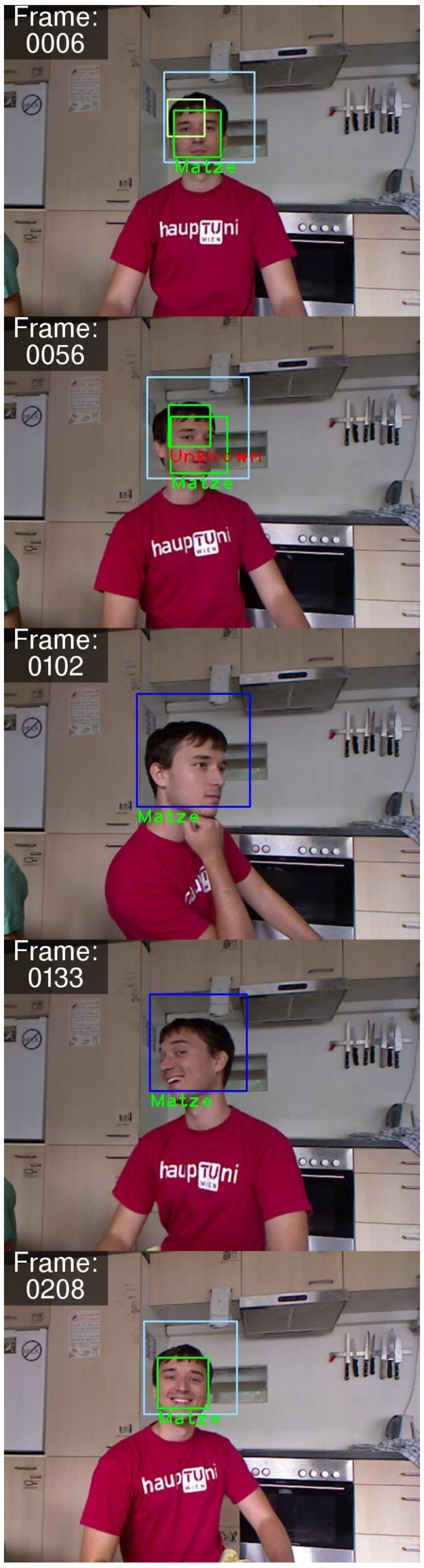


(b) 3D head and face detection

Figure 7.5: Tracking and recognition of Berni

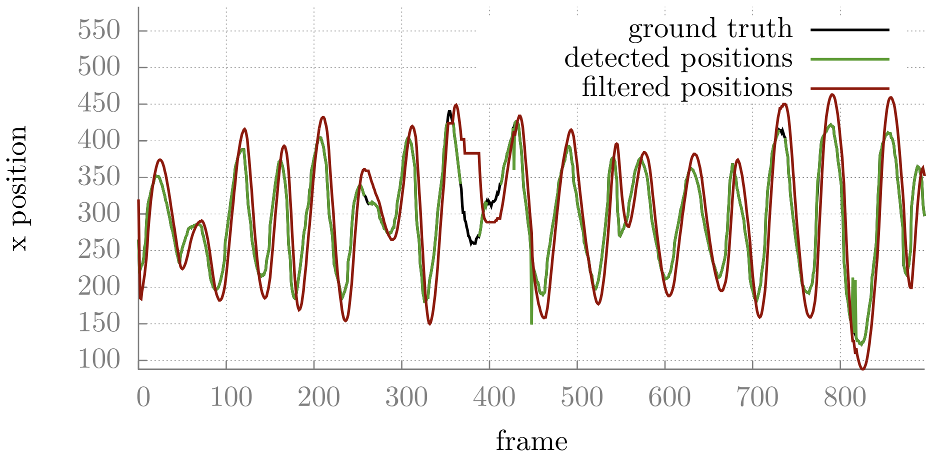


(a) 2D detection and Kalman filtered tracking

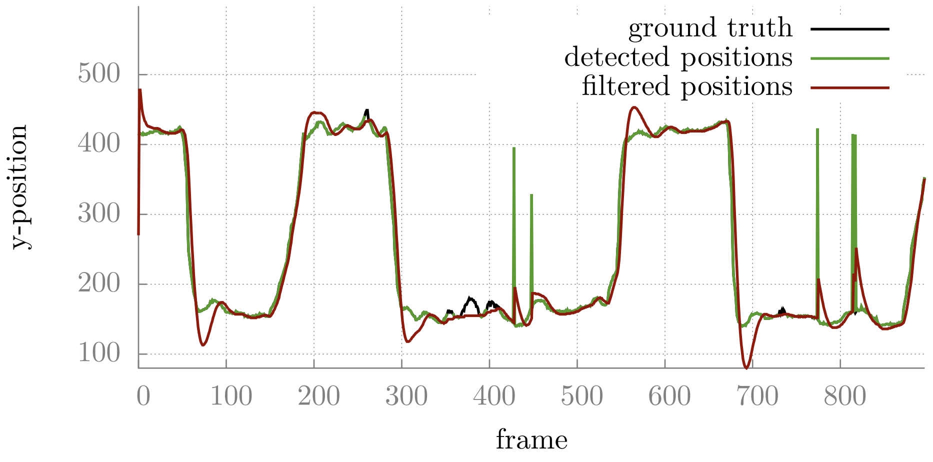


(b) 3D head and face detection

Figure 7.6: Tracking and recognition of Matze

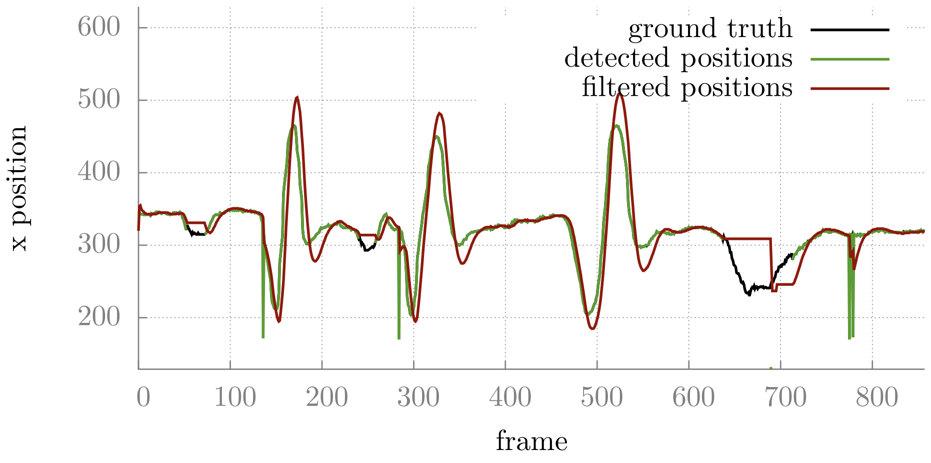


(a) center x-position over time

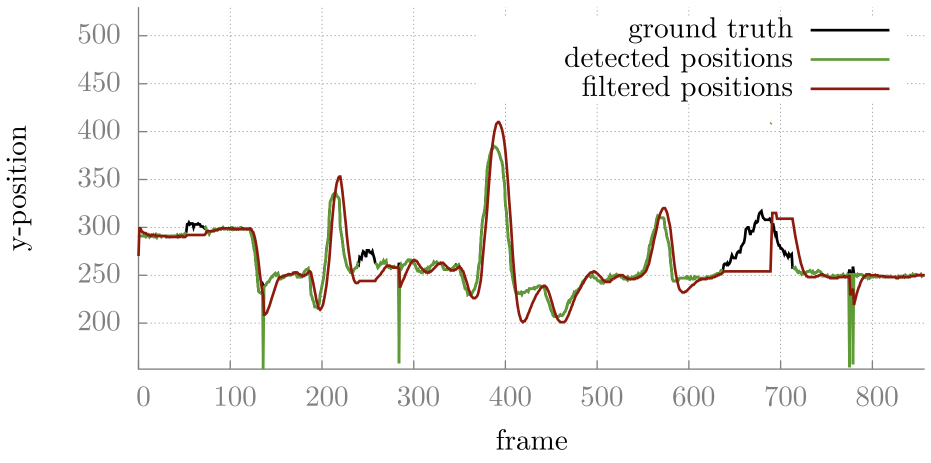


(b) center y-position over time

Figure 7.7: Detected and Kalman filtered face xy-positions of Andreas

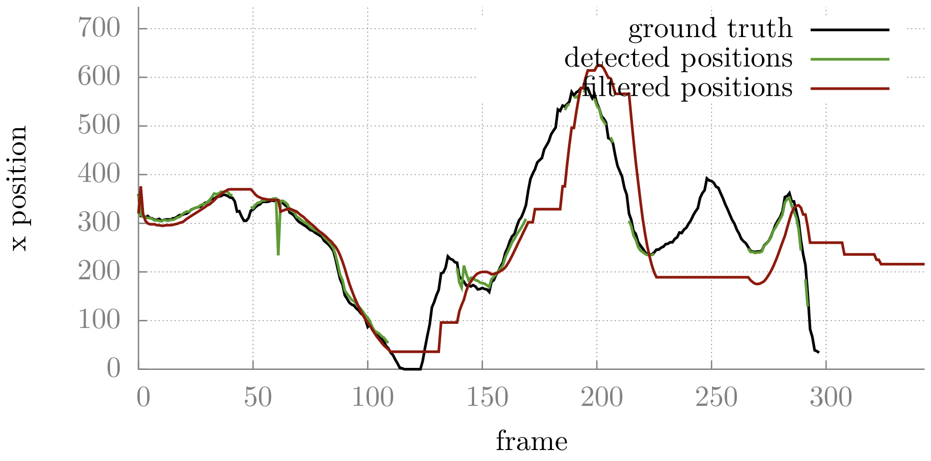


(a) center x-position over time

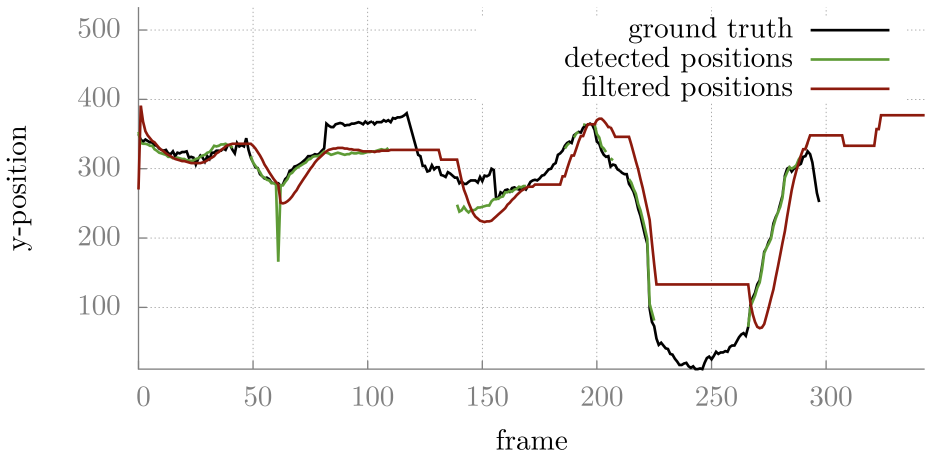


(b) center y-position over time

Figure 7.8: Detected and Kalman filtered face xy-positions of Davor



(a) center x-position over time



(b) center y-position over time

Figure 7.9: Detected and Kalman filtered face xy-positions of Manuel

7.2 Evaluation of the object classification

The object classification was also subject to verification. For this a training database was generated from 3D models, freely available from Google 3D Warehouse [16]. Nine categories were chosen for the evaluation of the implementation. The categories and there recognition rate is later shown in table 7.1, but first the process of the analysis is explained.

Acquire and prepare 3D model

Download the model for the respective class from 3D Warehouse. As these are in the Collada file format we use the open source program MeshLab[17] to convert it into the Polygon File Format. The resulting *.ply* files are placed in a subfolder, named after the corresponding class inside
/Model3DClassification/database/TrackingDB.

Database training

Startup the *Model3DClassification* and sent the String "train" to the ROS topic */M3D/trainer_requests* once. From these *ply* files the random decision forest is trained by creating point clouds from the 3D models and running those through the forest.

Object classification

The point clouds, recorded with the Kinect sensor, are fed into the decision forest which is now in its classification mode. For each object class label and the probability of this class is returned.

	average success rate
mug	97.54%
horse	95.21%
banana	98.30%
plane	96.48%
water filled bottle	07.21%
car	95.42%
hammer	93.21%
microphone	47.82%

Table 7.1: Average success rate of object classification

Most of the items we used for this tests had a successful classification rate above 90%. The water filled bottle proved to be the most problematic candidate. This is not due to problems with the classification itself, but with the recording of the depth data with

[16]http://sketchup.google.com/3dwarehouse/
[17]http://meshlab.sourceforge.net

the Kinect sensor. The transparent surface of the bottle combined with the reflections from the water within the bottle did result in multiple separated point clouds which were classified as a bottle with a single-digit success rate.

The images (7.10, 7.11, 7.12 and 7.13) were taken during the classification experiments and represent a small subset of data that was recorded and processed.

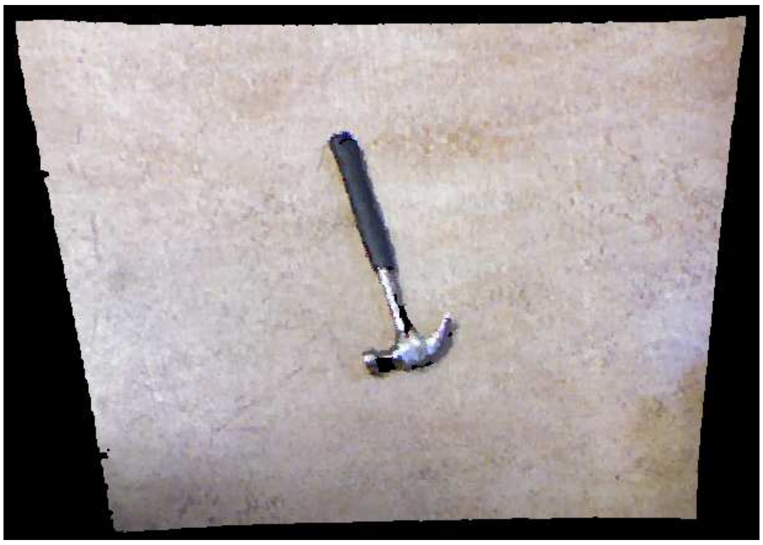


Figure 7.10: Coloured point cloud for object: hammer

7.3 Comparison of speech recognition packages

For voice recognition three systems were evaluated. These are PocketSphinx from Carnegie Mellon University, Dragon Mobile from Nuance and Voice typing from Google. The evaluation was done with the following procedure.

- Create a list of words and sentences which should be recognized.
- Create language model and dictionary for PocketSphinx.
- Record sentences and plain words to test against the candidates.
- Obtain or build sample applications for evaluation.
- Play the words and sentences to the candidates and save the number of tries the program needs to correctly identify the spoken words. Repeat this up to five times.

The results for a subset of individuals from the evaluation process are shown in table 7.2. The number represent the quantity of repeats after which the tested recognition engine would deliver a correct result. A '-' denotes that the speech recognition engine

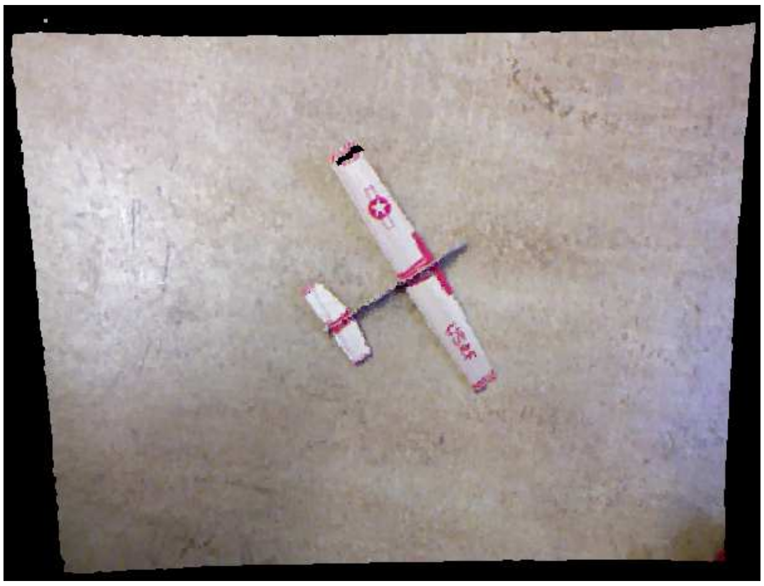


Figure 7.11: Coloured point cloud for object: plane

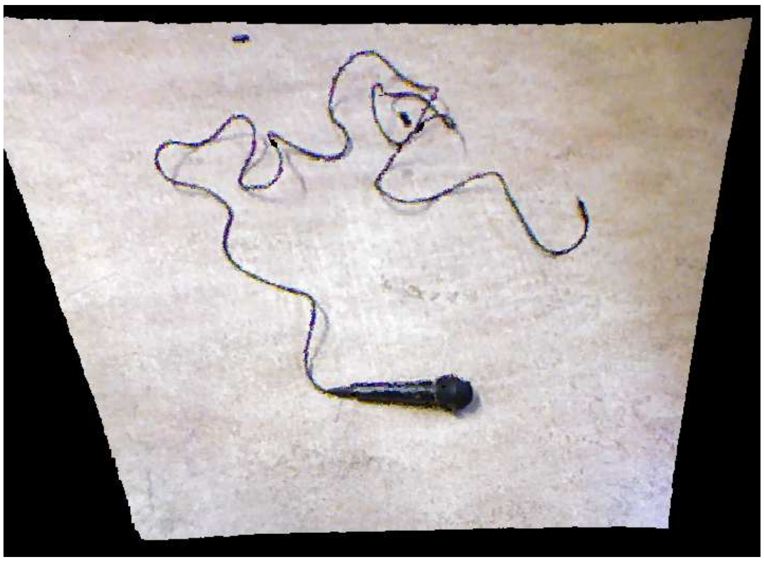


Figure 7.12: Coloured point cloud for object: microphone with cable

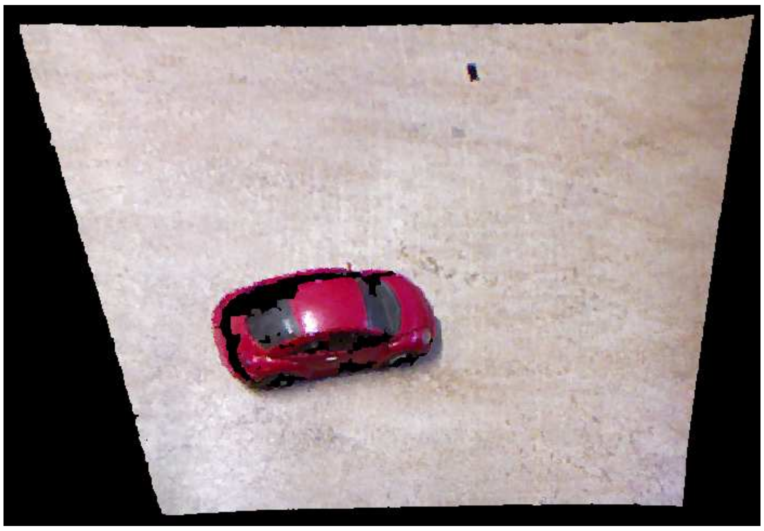


Figure 7.13: Coloured point cloud for object: car

did not successfully return the precise dictated sentence or word. As we can see PocketSphinx can recognize most of the words without the need for more than two repeats. Sentences represent a major problem, especially when a sentence was slightly altered. Just by changing an indefinite to a definite article the basic recognition by PocketSphinx shows severe problems. Nuance's and Google's recognition services handle this kind of work much better. This higher recognition rate is due to the fact that their applications can rely on a bigger and optimized language and acoustic model to work with, as well as the possibility to examine sentences after the recognition against a plausibility model. By doing so a command like "take out the car" which probably meant "take out the cat" is corrected before it is sent to the user. After the evaluation of the recognition results we chose to use the recognition library provided by Nuance. Its recognition rate was at least on par with the system provided by Google. PocketSphinx could not keep up with the other two. One main reason to use Nuance was the fact that changing the recognized language can be handled from within the system itself, a feature which could not be replicated in the Google application.

As these trials were conducted at the beginning of this master thesis the circumstances could have changed. Google has implemented a new speech recognition, Google Now, in its applications for Android. This provides, in contrast to Nuance, the ability to implement an offline-only speech recognition application. The problem with this is that no stable Application programming interface has been released yet.

	candidate 1 (female)			candidate 2 (male)			candidate 3 (male)		
	Google	Nuance	PocketSphinx	Google	Nuance	PocketSphinx	Google	Nuance	PocketSphinx
AGAIN	1	1	1	1	1	3	2	2	1
AIRPLANE	2	1	1	1	2	1	2	2	2
APPLE	-	1	-	4	1	-	4	2	3
ARE	-	5	-	-	4	1	5	5	-
BANANA	1	1	1	1	1	1	1	1	1
BOWL	-	4	3	4	3	2	-	5	3
BYE	2	1	1	2	2	1	3	1	1
CAR	-	2	1	3	2	2	-	3	1
DONE	4	2	1	3	3	1	4	3	1
EVA	1	1	2	1	1	1	2	1	1
FIND	1	1	1	1	1	1	2	1	1
GOOD	-	2	5	4	3	4	4	2	4
HAMMER	-	2	2	3	2	1	-	3	1
HOW	3	2	-	3	2	4	3	1	4
JOB	3	1	1	3	1	1	2	2	1
LOOK	1	1	-	1	1	3	1	1	3
MUG	-	-	1	5	5	1	-	-	2
SEARCH	1	1	1	1	1	1	1	1	1
SLEEP	3	2	4	3	3	3	2	1	5
START	-	2	-	2	2	1	3	1	4
STOP	2	1	3	3	1	2	2	2	4
WELL	-	-	1	3	3	2	4	4	2
WHO	1	2	2	2	2	1	1	1	1
YOU	1	1	1	2	2	1	2	1	3
FOLLOW	4	2	-	4	2	4	3	1	4
HELLO	1	1	-	1	2	4	1	1	4
HOW ARE YOU	1	1	5	2	2	4	1	1	4
FIND HAMMER	2	2	1	2	2	3	2	1	2
FIND BANANA	2	1	-	1	2	3	1	1	4
FIND MUG	3	-	-	2	4	2	3	2	4
SEARCH CAR	1	2	1	2	2	2	2	2	5

Table 7.2: Speech recognition results from three speakers
Green entries mark the best result
Blue entries denote a tie between the marked speech engines

8 Conclusion

In this thesis we showed that a natural multimodal interaction between human and robots is reliable, but by no means perfect. The voice recognition has a good success rate and is able to understand most commands after one or two repetitions. Possible ways to improve the recognition rate include the usage of a dedicated microphone instead of the tablets internal or the implementation of a context aware recognition logic which can evaluate the results based on the surroundings of the robot. The Viola-Jones face detection performed well inside the expected limits and provided, in combination with a Kalman filter, a reliable method to implement a face tracking solution. Using filter cascades for different views of faces other than frontal could provide improved results. The face recognition based on the data from this approach could not provide solid findings.

The mixed 3D and 2D face recognition process was able to recognize eight out of ten individuals after the short learning phase. The pre-recorded video and point cloud sequences of the remaining two test subjects failed to provide the required amount of data for the learning process. The amount of needed data was lowered for evaluation purposes, but it was soon discovered that these modifications showed a significantly higher false positive rate for face recognition rate. The best results were obtained when only one person was in the line of sight of the Kinect sensor and the face was frontal to the camera for the whole time of the recorded scene.

The object classification provided an accurate classification for the objects within the trained classes. Some objects outside of them were randomly classified into one of the trained classes (e.g. bowl classified as a mug). Problems occurred when multiple objects were close together as there point clouds were connected and formed an object which could not be recognized. Another problem was observed when the surface on which the objects rested could not be removed entirely, which changed the analysed cluster in such a fashion that the classification did not result in a correct manner.

The future outlook for multimodal interaction between humans and robots is bright. Speech recognition is well on the way to replace traditional input methods like touch input on a screen and will provide an easier way to control robots in the next years. Face detection suffers from false positive recognitions based on unrelated objects most often in the background of the user itself. With the help of 3D data we are able to reduce this problem already. Future detections should move away from the face centred detection of individuals and incorporate multiple sensors like contactless temperature and vital signs sensors to accurately detect and maybe even identify an individual. To the object classification the next step will be to the object recognition, which should provide the ability to distinguish between an object of one user or another, even if both of them are in the same object class.

Provided these improvements were made human robot interaction will be simplified in a significant amount.

Bibliography

[1] Peter N. Belhumeur, João P. Hespanha, and David J. Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection, 1997.

[2] Kevin W. Bowyer, Kyong Chang, and Patrick Flynn. A survey of approaches and challenges in 3d and multi-modal 3d & 2d face recognition. *Computer Vision and Image Understanding*, 101:1 – 15, 2006.

[3] Gary Bradski and Adrian Kaehler. *Learning OpenCV*. O'Reilly Media Inc., 2008.

[4] Chi-Ho Chan, Josef Kittler, and Kieron Messer. Multi-scale local binary pattern histograms for face recognition. In *Proceedings of the international conference on Advances in Biometrics*, pages 809–818. Springer-Verlag, 2007.

[5] Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, 2002.

[6] Antonio Criminisi, Jamie Shotton, and Ender Konukoglu. Decision forests for classification, regression, density estimation, manifold learning and semi-supervised learning. Technical report, Microsoft Research, 2011.

[7] H.K. Ekenel, Hua Gao, and R. Stiefelhagen. 3-d face recognition using local appearance-based models. *IEEE Transactions on Information Forensics and Security*, 2:630–636, 2007.

[8] J.C. Gower. Generalized procrustes analysis. *Psychometrika*, 40:33–51, 1975.

[9] European Telecommunications Standards Institute. Speech processing, transmission and quality aspects; distributed speech recognition; front-end feature extraction algorithm; compression algorithms, 2003.

[10] Cheuk Yiu Ip, Daniel Lapadat, Leonard Sieger, and William C. Regli. Using shape distributions to compare solid models. In *Proceedings of the seventh Association for Computing Machinery symposium on Solid modeling and applications*, pages 273–280, 2002.

[11] Chadawan Ittichaichareon, Siwat Suksri, and Thaweesak Yingthawornsuk. Speech recognition using mfcc. In *IEEE International Conference on Software Maintenance*, 2012.

[12] D. Jurafsky and J.H. Martin. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition.* Prentice Hall series in artificial intelligence. Prentice Hall, 2000.

[13] Kalman, Rudolph, and Emil. A new approach to linear filtering and prediction problems. *Transactions of the ASME - Journal of Basic Engineering*, 82:35–45, 1960.

[14] Slava M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. In *IEEE Transactions on Acoustics, Speech and Signal Processing*, pages 400–401, 1987.

[15] A. Koschan, V.R. Ayyagari, F. Boughorbel, and M.A. Abidi. Automatic 3d face registration without initialization. In Andreas Koschan, Marc Pollefeys, and Mongi Abidi, editors, *3D Imaging for Safety and Security*, volume 35 of *Computational Imaging and Vision*, pages 69–93. Springer Netherlands, 2007.

[16] Beth Logan. Mel frequency cepstral coefficients for music modeling. In *In International Symposium on Music Information Retrieval*, 2000.

[17] Binu Mathew. *The Perception Processor.* PhD thesis, School of Computing, University of Utah, 2004.

[18] Sebastian Mika, Gunnar Rätsch, Jason Weston, Bernhard Schölkopf, and Klaus-Robert Müller. Fisher discriminant analysis with kernels, 1999.

[19] T. Ojala, M. Pietikainen, and D. Harwood. Performance evaluation of texture measures with classification based on kullback discrimination of distributions. In *International Conference on Pattern Recognition*, volume 1, pages 582–585 vol.1, 1994.

[20] R. Osada, T. Funkhouser, B. Chazelle, and D. Dobkin. Matching 3d models with shape distributions. In *International Conference on Shape Modeling and Applications*, pages 154 –166, 2001.

[21] C Papageorgiou, M Oren, and T Poggio. A general framework for object detection. In *Proceedings of IEEE International Conference on Computer Vision*, 1998.

[22] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, pages 257–286, 1989.

[23] S. S. Stevens, J. Volkmann, and E. B. Newman. A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America*, pages 185–190, 1937.

[24] Richard Szeliski. *Computer Vision: Algorithms and Applications.* Springer-Verlag New York, Inc., 2010.

[25] Paul Viola and Michael Jones. Robust real-time object detection. In *International Journal of Computer Vision*, 2001.

[26] Greg Welch and Gary Bishop. An introduction to the kalman filter, 1995.

[27] Max Welling. Fisher linear discriminant analysis, 2005.

[28] W. Wohlkinger and M. Vincze. Ensemble of shape functions for 3d object classification. In *IEEE International Conference on Robotics and Biomimetics*, pages 2987 –2992, 2011.

[29] Ming-Hsuan Yang, D. Kriegman, and N. Ahuja. Detecting faces in images: a survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:34–58, 2002.