

Master's Thesis

at the Chair of Software Engineering, Department of Computer Science, ETH Zurich

Gesture-based user interface for SmartWalker

by David Itten

Spring 2014

ETH student ID: 07-804-446
E-mail address: ittend@student.ethz.ch

Supervisors: Dr. Jiwon Shin
Andrey Rusakov
Prof. Dr. Bertrand Meyer

Date of submission: 15 October 2014

Affidavit

I hereby declare that this master thesis has been written only by the undersigned and without any assistance from third parties. Furthermore, I confirm that no sources have been used in the preparation of this thesis other than those indicated in the thesis itself.

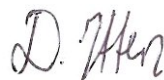
This thesis has not yet been presented to any examination authority, neither in this form nor in a modified version.

By signing this statement, I affirm that I have read the information notice on plagiarism, independently produced this paper, and adhered to the general practice of source citation in this subject-area.

Information notice on plagiarism:

[https://www.ethz.ch/content/dam/ethz/main/education/
rechtliches-abschluesse/leistungskontrollen/
plagiarism-citationetiquette.pdf](https://www.ethz.ch/content/dam/ethz/main/education/rechtliches-abschluesse/leistungskontrollen/plagiarism-citationetiquette.pdf)

Zurich, 15 October 2014



David Itten

Abstract

In this work, a novel gesture recognition system is presented that allows the control of the SmartWalker by performing simple gestures. The SmartWalker is a normal walker equipped with sensors and actuators that can move autonomously. It is constructed as a walking aid for elderly people. Therefore our gesture recognition system is developed for the same target group which means that gestures with more deviation from the specified gestures need to be supported.

We developed two gesture recognition systems. User tracking uses skeleton information extracted from the stream of an RGB+D camera and calculates relative distances and angles between body joints as features. Hand tracking uses the center position of one hand to extract features based on the current hand position relative to a gesture starting point. With the help of Dynamic Time Warping, gestures can be performed at different speeds compared to the training data. For elderly people this is important as persons with physical constraints perform the gestures with irregular movements.

To evaluate the performance of our gesture recognition system, we did a performance study where we tested both hand and user tracking at different distances. Additionally, we visited several nursing homes to test our robot and the new gesture recognition system with elderly people.

The achieved recognition rates are lower than state-of-the-art gesture recognition systems. Our system on the other hand can handle gestures that are given with large deviations from the specified gestures, it works in unknown environments and supports distances of up to 3.5 meters.

Zusammenfassung

In dieser Arbeit wird ein neues Gestenerkennungssystem vorgestellt, welches die Steuerung des SmartWalkers ermöglicht. Der SmartWalker ist ein normaler Rollator, welcher mit Sensoren und Aktoren ausgerüstet wurde und sich automatisch bewegen kann. Da der SmartWalker als Gehhilfe für ältere Leute entwickelt wurde, muss auch das Gestenerkennungssystem für diese Zielgruppe angepasst sein. Deshalb wurde bei der Entwicklung speziell auf die Erkennung von Gesten Wert gelegt, welche abweichend von ihrer Spezifikation und mit unregelmässigen Bewegungen ausgeführt werden.

Wir haben zwei Gestenerkennungsalgorithmen entwickelt. Der User Tracker benutzt die 3D Koordinaten verschiedener Körperpositionen, welche aus einem RGB+D Kamerabild berechnet werden. Für die Erkennung der Gesten werden relative Distanzen und Winkel zwischen Körperpositionen als Features verwendet. Der Hand Tracker verwendet die aktuelle Handposition relativ zu einem Gestenstartpunkt. Mit Hilfe von Dynamic Time Wapping ist es möglich, Gesten zu erkennen, die mit abweichender Geschwindigkeit

gegenüber den aufgezeichneten Gesten ausgeführt werden. Dies ist für unser System von spezieller Wichtigkeit, da ältere Leute die Bewegungen oftmals sehr langsam und zum Teil unregelmässig ausführen.

Um die Leistungsfähigkeit unseres Systems zu analysieren, haben wir einen Test des Systems durchgeführt, bei welchem der User und der Hand Tracker mit verschiedenen Distanzen getestet wurden. Weiter haben wir verschiedene Altersheime besucht, um unser System mit älteren Personen zu testen.

Die erreichten Erkennungsraten liegen unter denen aktueller Gestenerkennungsalgorithmen. Unser System ist jedoch fähig, Gesten mit starken Abweichungen gegenüber der Spezifikation zu erkennen. Zudem funktioniert das System in unbekanntem Umgebungen und mit Distanzen von bis zu 3.5 Metern.

Acknowledgments

I would like to thank my supervisors Dr. Jiwon Shin and Andrey Rusakov for their support and the interesting discussions we had during my work at the robotics group at the Chair of Software Engineering.

Prof. Bertrand Meyer I would like to thank for providing me the possibility to perform this project in his group.

Additionally I would like to thank all members of the group for the inspiring discussions and the good time I have had.

Contents

Abstract	v
Acknowledgments	vii
1 Introduction	1
1.1 Background	1
1.2 Contributions	2
1.3 Outline	2
2 System setup	3
2.1 Software	3
2.1.1 Robot Operating System (ROS)	3
2.1.2 OpenNI / NiTE	4
2.1.3 OpenCV	4
2.1.4 Transformation framework	6
2.2 Hardware	6
2.2.1 SmartWalker	6
2.2.2 PrimeSense RGB+D cameras	7
2.3 Architecture	7
2.4 Communication	9
2.5 Gestures	9
3 SmartWalker control application	13
3.1 Roboscoop	13
3.2 Operating modes	14
3.2.1 User Position Updates	14
3.2.2 Attention Check	14
3.2.3 Registered Users Check	15
3.3 Additions to Roboscoop	15
4 Gesture recognition	17
4.1 Overview	17
4.1.1 k-NN classifier	18
4.1.2 Dynamic time warping	18
4.2 Feature extraction	19
4.3 Gesture classification	20
4.3.1 Gesture specific improvement for the COME HERE gesture	21

4.4	Analysis of the used features	22
4.5	Training set	23
4.6	Determining the user position	24
5	Face detection and recognition	25
5.1	Face detection	25
5.1.1	Face used as goal position	27
5.1.2	Attention estimation	28
5.2	Face recognition	29
5.2.1	Teaching faces to the face recognizer	30
6	Evaluation	31
6.1	Performance analysis	31
6.2	Study in elderly homes	34
6.2.1	Setup of the study	34
6.2.2	Research questions	36
6.2.3	General findings	36
6.2.4	Change of general impression about the robot	37
6.2.5	Evaluation of the stopping position	38
6.2.6	Gestures	39
6.2.7	Software adaptations	42
6.2.8	Additional comments	43
6.3	Evaluation of different error measures	43
7	Related work	49
8	Conclusions and future work	51
8.1	Conclusions	51
8.2	Future Work	52
8.2.1	Stopping position	52
8.2.2	Gesture recognition	53
	Bibliography	55
A	Appendix	57
A.1	User Documentation	57
A.1.1	Installing the requirements	57
A.1.2	Robot Control Application	57
A.1.3	SmartWalkerTracker	58
A.1.4	Face detection	60
A.2	Developers Guide	61
A.2.1	SmartWalkerTracker	61
A.2.2	Camera publisher	62
A.2.3	User Tacking	62
A.2.4	Hand Tracking	66

A.3 Original questions of the nursing home study 68

1 Introduction

1.1 Background

Walkers are used a lot by elderly people or persons with physical constraints with the need of help during walking. As this device has a great popularity it makes sense to think about more functions that such a device could be used for.

The SmartWalker project started in may 2012 as a collaborative project between ETH Zurich and the iHomeLab of Hochschule Luzern. The idea of the project was to create a Walker equipped with sensors and actuators that can serve for additional purposes as for example supporting the users during a shopping trip with information on where products are placed in the store. Additionally, the SmartWalker has been developed as an application of the Roboscoop framework in the area of Ambient Assisted Living (AAL). With the ageing population we have, this is an interesting topic where a lot of research is undertaken.

The sensors and actuators in robots require the concurrent processing of large amounts of data. The Roboscoop framework facilitates the creation of robotic applications a lot. It is based on SCOOP, an Eiffel concurrency model for easier reasoning about parallel processes. Adapted for robotic applications, Roboscoop allows the reception of data from sensors and other software components of the robot. After processing, messages for controlling the robot can be generated.

The SmartWalker is equipped with a PrimeSense RGB+D camera. RGB+D cameras became popular with the Microsoft Kinect, first sold in November 2010. The initial usage of these cameras was the development of video games which were controllable by movements in front of the TV screen. These cameras provide a normal RGB image as well as a depth image, which records the distance from the camera to the captured object for every pixel. Since the introduction of this camera, a lot of research has been done to adopt this sensor type to other fields of usage.

1.2 Contributions

This thesis contributes a novel function to our SmartWalker, namely the control of the walker by gestures and the determination of the position of the user. The goal of the created user interface is that it should be intuitive, which means that natural movements should be used for commanding the robot. The gestures used in the system should be easy to learn and people with physical constraints should be able to perform them.

As a second achievement, our work adds a face recognition and detection system to the SmartWalker. Face detection allows the device to achieve higher accuracy for the detection of the user position and to measure the user attention. Face recognition is used to recognize the owner of the robot by its face and to only react on its commands.

1.3 Outline

Chapter 2 describes the software and hardware used during the development of the gesture recognition system and that is required to use the developed solution. In chapter 3, the robot control application is described which serves as interface for the user and sends commands to the robot. Chapter 4 introduces the software component used for recognizing user and hand gestures. In chapter 5 the face detection and recognition are described. The evaluation of an in-house study and a study in elderly homes can be found in chapter 6. Related work on the same topic is summarized in chapter 7 and the conclusion of this work can be found in chapter 8.

2 System setup

In this chapter, the Software and the hardware used for our gesture recognition system is explained. To have an overview of the system, the architecture and the interaction between the different components are explained.

2.1 Software

2.1.1 Robot Operating System (ROS)

The robot operating system (ROS) [9] used in this work, enables the parallel execution of the different components of the robotic application and the communication between them. The different components of the system that run in parallel are called nodes. A node is essentially a normal application that fulfils its dedicated task and communicates with other nodes. To facilitate the development of nodes, ROS provides libraries for C/C++ and Python. They contain everything that is necessary for the communication between the nodes and other useful functions that allow the developer of a node e.g. to limit the rate of a processing loop. The communication between the nodes is very important as in a robotic application nearly every component needs to communicate with others.

In ROS the communication is organized in topics. A node can advertise a topic and publish messages on it, whereas other nodes can subscribe for this topic. Every topic transports messages of a certain type which is understood by the publisher and the subscriber.

For the SmartWalker, ROS is mainly used to synchronize the different components of the gesture recognition system and to exchange messages. The component that detects the face of a user for example, publishes this information to the robot control application where the information is used to deduce if a user is looking towards the robot and to recognize the owner of the robot.

2.1.2 OpenNI / NiTE

OpenNI¹ is an open source framework for developing applications using RGB+D cameras. It supports a wide range of sensors including the popular Microsoft Kinect, Asus Xtion or the Primesense Carmine sensors. OpenNI facilitates the access to the hardware. It allows applications to read information from the connected camera or to change settings on the camera. It also enables easy access to the video streams of the cameras.

NiTE is a closed source middleware library developed by PrimeSense which is based on OpenNI. To run NiTE, an OpenNI device object is required. NiTE can be used with all the devices that are supported by OpenNI.

NiTE offers user and hand tracking capabilities. For user tracking, 15 body joints are tracked and can be read from a UserData object. For every joint, the exact 3D position relative to the camera is available as well as the orientation in the form of a Quaternion. The following joints are tracked by NiTE: left hand, left elbow, left shoulder, right hand, right elbow, right shoulder, neck, head, torso, left hip, left knee, left foot, right hip, right knee and right foot (see figure 2.1).

In this work, OpenNI is used to access the PrimeSense RGB+D camera. The data received is used for two purposes: Firstly, it is used by our face detection system and secondly it is used as input data for NiTE. The user and hand tracking capabilities of NiTE are used in this work to receive 3D position data of the users joints. This information is required by our gesture recognition system to extract features.

2.1.3 OpenCV

Open Source Computer Vision (OpenCV) [6] is a computer vision library for processing static images or video sequences. The library has an interface to ROS and supports conversion of ROS image messages to image formats that are used within OpenCV. The library has C, C++, Python and Java interfaces which makes the library usable for a wide range of operating systems. Some of the available algorithms can also be used on Android devices such as tablet computers or smart phones. The C++ interface of OpenCV supports multicore processing. With the help of threading building blocks (TBB) [2] most algorithms can have parallel working threads which speeds up the computations.

This library is used on the RGB data of the camera for doing the face detection as well as the face recognition (see section 5.1).

¹Unfortunately the OpenNI framework as well as the NiTE middleware are discontinued as PrimeSense has been bought by Apple. The official websites for downloading the libraries, the support forums and the documentation webpages are not online any more.

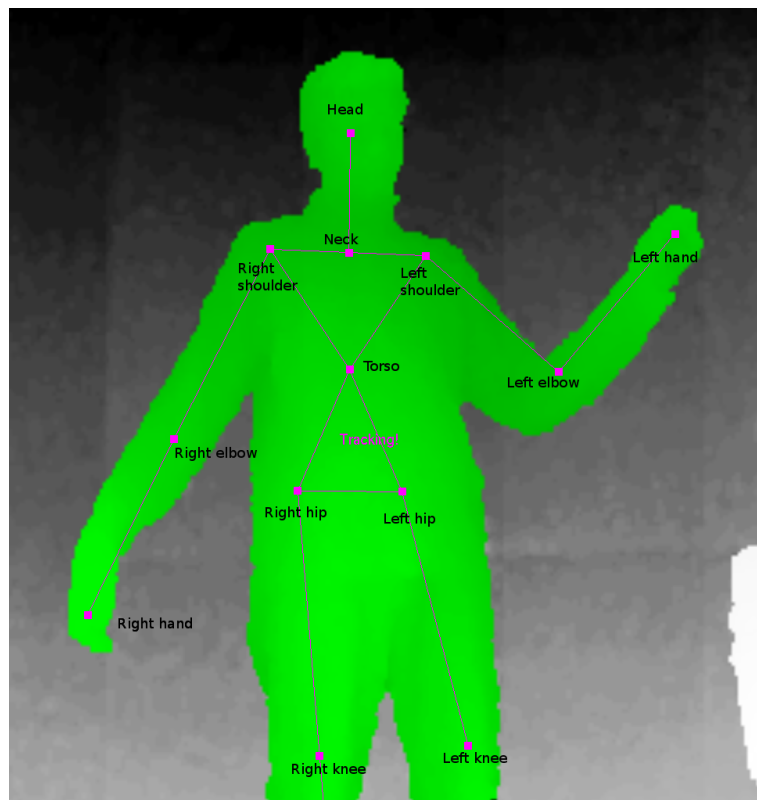


Figure 2.1: Depth image with joints tracked by NiTE

2.1.4 Transformation framework

The transformation framework (TF) [10] is a ROS package that publishes position information in different coordinate frames. Position information can be published in TF which is sent to ROS via a /tf topic. Transformations from one coordinate frame to another are done automatically and can also include past position information. TF uses a tree structure, starting from a common world frame. Except the world frame, every other node in the TF structure has to have a parent node. For robotic applications, this framework is very useful as several positions are known with respect to an other position.

As an example for the SmartWalker, the user position is known with respect to the camera. As the camera can be turned, the camera position is known with respect to the base of the robot. With TF, the position of the user with respect to the robot base can be calculated easily.

2.2 Hardware

2.2.1 SmartWalker

The SmartWalker has been developed collaboratively at the Chair of Software Engineering, ETH Zurich and at the iHomeLab of Hochschule Luzern. It is a high-tech extension of a normal walker. It is equipped with a laser sensor at the bottom, which is able to recognize the distances to the surrounding by one degree steps. At the handle bar, it is equipped



Figure 2.2: The SmartWalker with an additional Laptop for face and gesture recognition.

with a Primesense Carmine 1.08 long range RGB+D camera. This camera is mounted to a servo motor which is able to turn the camera around by 360 degrees. For mobility, the SmartWalker has two wheel hub motors which can move the walker automatically.

The actuators and the laser sensor are connected to a Beaglebone micro-controller board. As the RGB+D camera needs more processing power to handle the camera input, the PrimeSense sensor is directly connected to the tablet computer mounted on the handle bar. The tracking software as well as the robot control application are installed on the tablet computer. For controlling the robot, messages over a wired LAN network are sent to the Beaglebone board. For navigation, the robot publishes odometry information to ROS. This information is gathered by sensors in the wheels that register every movement of the robot no matter if a user pushed the robot or if it moved automatically.

The robot is powered by an e-bike battery, which supplies the motors as well as the beaglebone boards. The primesense camera gets its power from the tablet computer which has its own battery.

2.2.2 PrimeSense RGB+D cameras

For this work, two versions of the PrimeSense Carmine camera have been used. For local testing a PrimeSense Carmine 1.09 short range camera was used. This camera has an operating range from 0.35 up to 1.4 meters. The robot is equipped with a PrimeSense Carmine 1.08 long range camera with an operating range of 0.8 up to 3.5 meters. Both cameras provide a normal RGB image as well as a depth image, which is generated from a reflected infra-red light that was emitted by the camera. The PrimeSense sensor only works in indoor environments with no direct sunlight exposure. For the depth as well as for the RGB image, a VGA resolution of 640 x 480 pixels is captured. This camera is very low cost in comparison with conventional stereo cameras.

The camera provides RGB+D data. These consist of a normal RGB image and a depth image, which provides the distance for every pixel from the camera to the captured object. In figure 2.3 the left image shows the RGB camera image. On the right side, the depth image is shown, with white pixels for distances longer and black pixels for distances shorter than the range of the camera.

2.3 Architecture

The software architecture of the SmartWalker consists of the Robot control application, the SmartWalker tracker and the Face tracker. These applications are developed as ROS nodes. ROS offers publish and subscribe mechanisms to all nodes of the system and organizes



Figure 2.3: RGB and depth image of the Primesense camera

the communication with topics. It also maintains a directory of available services that can be used in the system.

An overview over the architecture of the system is shown in figure 2.4. The robot control application is the interface between the gesture recognition and the robot. This software controls all actions of the SmartWalker. The actions implemented in the robot control application depend on input data of the face tracking, the SmartWalker tracker and feedback of the robot.

Roboscoop uses signalers to store the newest informations received from other components or sensors. The robot control application contains several signalers that hold the updated information from the SmartWalker tracker and the face tracker. Two of them are of special importance for this work: The User position signaler holds all positions of the tracked users. If faces are tracked, also the face positions are available. The gesture signaler stores the last gesture given to the system. Additionally, it can notify subscribers, if a new gesture arrives in the system. For controlling the robot, more signalers are required. There are for example two pose signalers. One stores the current position of the robot, whereas a second pose signaler holds the position of the target that the robot has to approach.

Depending on the data stored in the signalers, the robot control application decides actions. For example the control application will move the robot in front of a person, when an updated user position and a COME HERE gesture have been received. As the proportional-integral-derivative (PID) control is also part of the control application, the control application sends low level information such as the wheel speeds to the robot. The robot keeps track of the movements and provides odometry information back to the control application.

A separate node is dedicated to face detection and recognition. These two tasks are implemented in two separate modules. Face detection first detects faces in the camera

image. After that, the face recognizer tries to find known faces in the camera image and reports the found user id's. These separate modules form a hierarchy and also allow parallel execution of the two tasks.

The SmartWalker tracker is the central unit for gesture recognition. It is split into three parts which can all execute in parallel. First there is a camera publisher, that publishes the RAW camera data to ROS. The second component is the hand tracker, which is responsible for finding and tracking hands. After that, the data is analysed to find hand gestures and report them to ROS. For estimating the location of the user, the hand center position is also reported to ROS. The user tracker is the last component. It does full-body tracking of the users and reports all joint positions to ROS (as TF frames). It performs gesture recognition on this data and if gestures were detected, these are also reported to ROS. The camera is connected to this node and can be accessed directly for getting the input data for the gesture recognition.

2.4 Communication

The communication between the software components of our gesture recognition system is message-based via ROS topics. In figure 2.5 all nodes of the system and the most important topics are shown. The SmartWalkerTracker2 node is the main component for the gesture recognition. The arrows from this node to the camera show the four topics published containing the camera info and the image raw data. Face tracking is also a separate ROS Node. It uses the camera data and publishes the face position data for user and hand tracking. This data contains the results from face recognition. Roboscoop_ros is a ROS node that is used as an interface between ROS nodes written in C or C++ and Roboscoop, which is an Eiffel framework. The SmartWalkerTracker publishes data about the recognized gestures and the current user position detected by the user or the hand tracker.

2.5 Gestures

For commanding the robot, three different gestures have been defined. The gestures should be done efficient and uninterrupted but not extremely fast. The chosen gestures have been selected as they should be similar to natural movements if humans communicate to each other.

The arm-raise gesture is used for calling the robot. It consists of a straight upward movement of at least 20 centimetres, with the palm directed to the camera. When the hand is up, the hand should return back to the initial position without waiting. This gesture can be made at an arbitrary starting position, but it should not go out of the visible range of

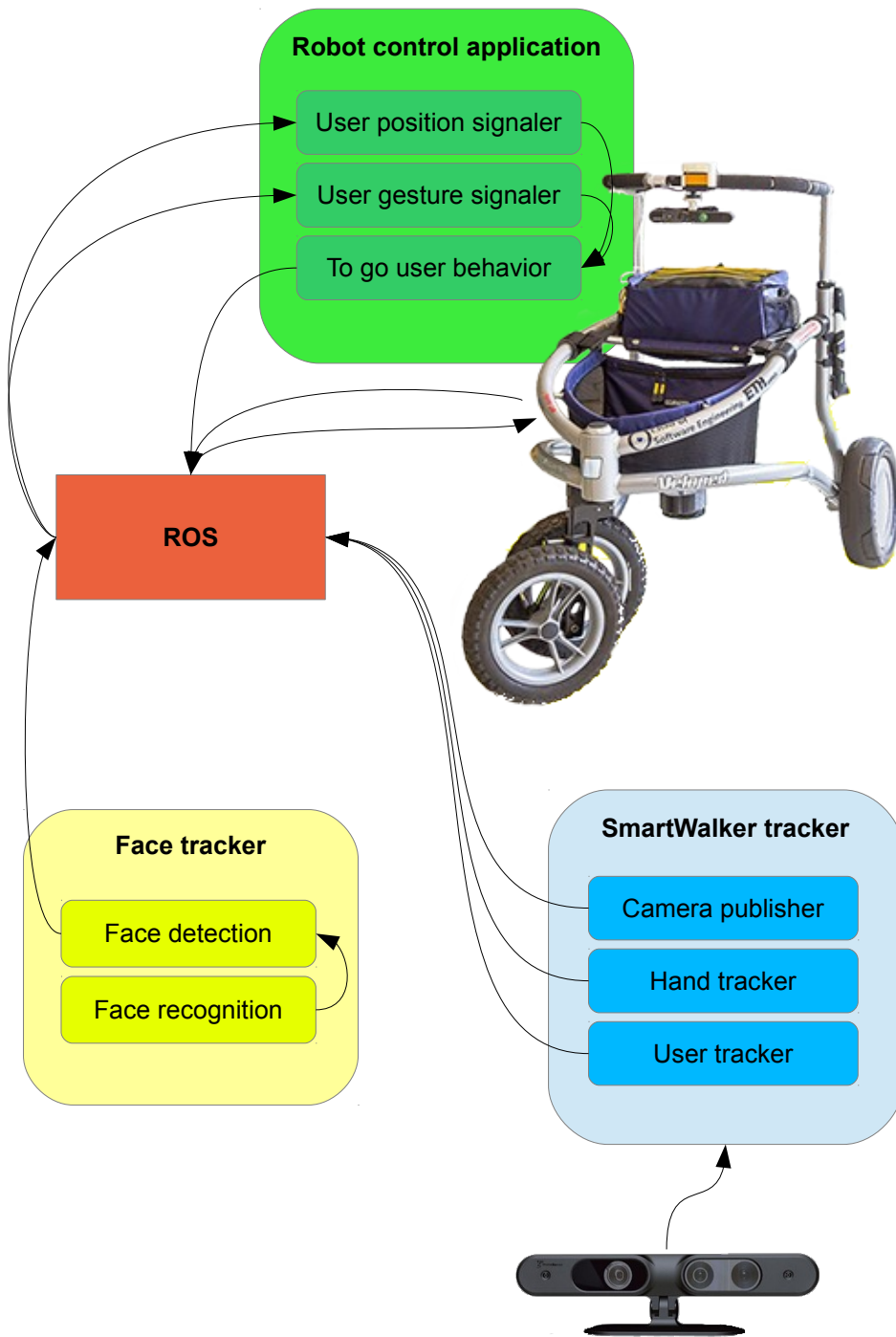


Figure 2.4: Architecture of the system. Every software component is coloured differently. The software components are split into modules which can run in parallel. Arrows show the messages that are exchanged in the system.

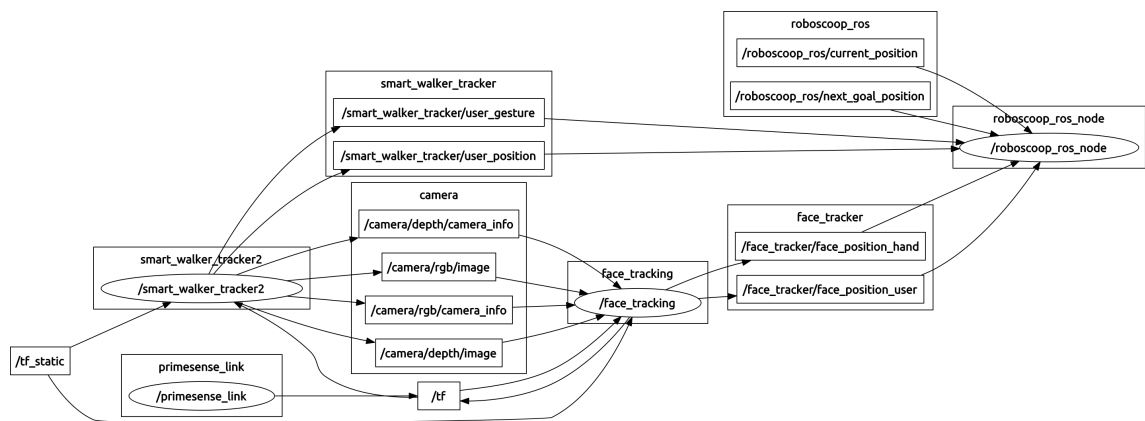


Figure 2.5: ROS nodes and topics. Ovals represent software nodes that process incoming data and publish data to other topics. Rectangular boxes represent topics. The arrows show where nodes publish information or subscribe for data.

the camera. This gesture has been chosen because humans often signal their positions to others in a crowd by raising their arm.

The hand-front gesture is used to stop the robot at the current position. The hand is held in front of the body, with the palm visible for the camera. Then the hand is moved perpendicular to the front of the body for at least 30 centimetres. Without waiting at this position, the hand is returned to the initial position. This gesture is often used in daily life to signal a stop command to other people. For regulating the traffic for example, this gesture is used to make traffic participants stop.

The arm-out gesture is used to send the robot back to a defined position, such as a charging station. For this gesture, the hand with the palm turned towards the camera is held 10 to 20 centimetres in front of the chest. Then the hand is moved at least 30 centimetres to the left, if the left hand is used, or to the right, when it is the right hand. Without waiting at the outer position, the hand is immediately returned to the initial position. A pointing direction to the side is a natural way of commanding someone to a place. In our case the robot can not follow the direction of the pointing but the gesture should still be natural.

3 SmartWalker control application

The gesture control for the SmartWalker consists of two components. One component is the gesture recognition itself called SmartWalkerTracker and the control application for the robot. This chapter explains, the control application.

3.1 Roboscoop

The robot control application is based on the Roboscoop framework. The Roboscoop framework [11] is divided into three layers which are all using SCOOP to handle concurrency. The Roboscoop framework manages the communication between the different layers:

- **Control layer:** Contains stateless feedback control loops to couple sensors to actuators. The control of the robot is written in this layer, which incorporates all primitive robot behaviour.
- **Sequencer layer:** This layer interacts with the control layer to fulfil a task. It is stateful and relies on primitive behaviours from the control layer.
- **Deliberator layer:** The Deliberator layer performs time-consuming deliberative computations.

The robot control application written in this work is done in the Sequencer Layer. A new behavior called "Go to user behavior" has been created. It can be used like the other behaviors that are already present in Roboscoop. For this behavior an event based architecture has been chosen.

The general process of this behavior is as follows: First the behavior has to be started by a Roboscoop application. Then the behavior waits until a `RS_UserGesture` message arrives from the SmartWalkerTracker, which tells the application that a gesture was received. Depending on the gesture id, a `STOP`, a `GO BACK` or a `COME HERE` command will be executed.

To operate the SmartWalker, currently only a "go to goal behavior" is supported. This behavior works with two signalers. One holds the current position of the robot, whereas the other signaler can be used to set a goal position for the robot. In a later state of the

SmartWalker project, other behaviors will be added that support obstacle avoidance for example. The following actions are executed after a gesture has been received:

- **STOP:** Update the goal position with the current position of the robot, which makes the robot stopping immediately.
- **COME HERE:** Update the goal position of the robot with the newest user position that has been received from the SmartWalkerTracker via a RS_UserPosition message. When obtaining the current user position, available user positions messages are always prioritized as follows: UserTracker messages have the highest priority as they deliver the best results. The stopping position is determined by the two knee positions of the user, which is exactly the position where the robot should stop. Second priority have the messages from face tracking. They use the users face position to determine the goal position of the robot. The lowest priority have the messages of the HandTracker. As the hand moves around for giving the gestures and the gesture can be recognized at different points during the movement, larger deviations from the optimal stopping position are possible.
- **GO BACK:** Update the goal position of the robot with the position where the robot should move back. In the current implementation, we send our robot back to the position where we started from. Therefore the goal position is updated with the origin (i.e. $x=0, y=0$).

3.2 Operating modes

The control application supports additional operation modes that can be enabled by the user in the graphical user interface. Besides the normal operation the following operation modes are supported:

3.2.1 User Position Updates

This mode updates the goal position, after the user gave the COME HERE command. If the user moves after commanding the robot to come by more than 30 cm, the goal position of the robot will be updated. If the robot is closer than 50 cm to the user, the updating process stops again.

3.2.2 Attention Check

If this mode is enabled, the robot only reacts on gestures if the attention of the user is directed towards the robot. The attention estimation is done by checking how much the

user looks in the direction of the robot. The details of the attention estimation are described in section 5.1.2. If the attention check was successful, the gesture command is executed. Otherwise the gesture is ignored. As the attention check only works if there are faces detected, this mode can be turned off, as for long distances or bad lighting conditions, the face detection does not work anymore.

3.2.3 Registered Users Check

This mode enables the face recognition of the robot. If this mode is turned on, only registered users are allowed to operate the robot. In this mode, the robot recognizes the faces in the camera image and only executes commands of users that are registered in its face database. Details about the implementation can be found in section 5.2.

3.3 Additions to Roboscoop

Besides the control application some additions to the Roboscoop framework have been created. One part of the additions concern the new message types that were used in Roboscoop. For the two new message types a signaler had to be created. Signalers in Roboscoop hold the newest value that was published on a topic, as the signalers are updated with every arriving message. The two created signalers are more sophisticated, because there is not just one user position to keep, but every user has its position that has to be accessible. Furthermore the event based setup does not regularly check the contents in the signaler, but the signaler has to notify observers about new gestures that have been received.

Additions were also made to the graphical user interface of the SmartWalker. A switch for every additional mode has been added to the GUI and better support for termination of Roboscoop after closing the GUI has been added.

4 Gesture recognition

In this work we developed a gesture recognition system for SmartWalker. As walkers are mostly used by elderly people, the created gesture interface should be easy to use and not require users to carry additional tools such as remote controls. The gestures should be easy to understand and the required movements should be easy to perform for elderly. As the SmartWalker is a mobile device, the gesture interface needs to work in many different environments and should adapt to different lighting conditions and backgrounds.

In this thesis we developed a software component called SmartWalkerTracker for recognizing user gestures based on skeleton data that contains 3D positions of the joints of a user. Our gesture recognition system provides a hand tracking and a user tracking mode. In user tracking mode, all tracked joints above the hip are used for generating feature vectors. Additionally, angles at the shoulder and the elbows are used as features. This tracking mode provides good accuracy and higher recognition rates, as many features can be calculated with the available skeleton data. Furthermore, the position information is very accurate because the joint positions can be verified by adjacent joints. As this solution requires the user to stand in front of the camera or at least to move the whole body to allow the recognition of the different joints, this tracking technique cannot be used by every person of our target group. Elderly persons often need the help of the walker to stand up and are not able to move without any assistance. The hand tracking mode has the advantage that the users of the walker can sit and it requires less movement for calibration. As only the center of one hand is tracked, parts of the body may also be occluded by obstacles. The drawback of this method is the lower detection rate as fewer features are present and the less accurate position information that is available for finding the target position if the robot should approach its user.

4.1 Overview

Our gesture recognition system takes a feature-based approach. The system extracts features from skeleton or hand center data. In user tracking mode, relative distances between body joints and angular information are used as features. In hand tracking mode, the 3D position of the hand with respect to a gesture starting point is used as features. Classification is done with a k-NN classifier. The costs for matching online data to recorded gesture templates are calculated by a dynamic time warping approach. Thus the system allows for different execution speeds of the gestures.

4.1.1 k-NN classifier

With a k-nearest neighbor (k-NN) classifier, a feature vector is classified by a selectable number k of closest neighbors, which need to be classified already. In the case of gesture recognition, the already classified feature vectors are taken from the training set of the classifier. For these feature vectors, it is known to which gesture class they belong.

The classifier needs to find the closest neighbors. In order to be able to do this, the feature vector which should be classified needs to be compared to all stored gesture templates. For this, the matching costs of the live data to the gesture templates is calculated. Afterwards the k lowest costs are used to count the number of occurrences that each gesture class has within this k cost values. This can be done, as every cost is related to a gesture class depending on the affiliation of the involved feature vector out of the training set. The gesture class with the most occurrences is then selected as output of the classification.

4.1.2 Dynamic time warping

Dynamic time warping (DTW) is an algorithm in time series analysis, which measures the similarity of two timeseries of data. DTW can be applied to different kinds of data such as audio, video and other data to find an optimal match between two temporal series of data. When calculating this optimal match, several conditions have to be applied: When matching the data of the first timeseries to the data of the second series, we can either go to the next frame or wait at the current frame in the first timeseries, depending on which variant produces the lower error. It is never possible to go back in the first timeseries.

DTW is implemented with a dynamic programming approach. In the dynamic programming table, every cell contains the lowest possible cost when matching the two timeseries of data up to time index i in the first timeseries, for the i 'th table row and up to time index j in the second timeseries, for the j 'th table column. With this scheme, the two timeseries are completely matched in the table cell at position (n,m) , with n being the number of columns and m the number of rows. Therefore the total, minimal cost for matching the two timeseries of data can be read from this table cell.

Pattern Recognition [13] describes dynamic time warping in speech recognition. Speech and gesture recognition are very similar topics as with a Fourier transform, a timeseries of feature vectors can be extracted from speech. DFT is applied to speech segments and the highest coefficients are used as features. Therefore, for skeleton data and speech we have feature vectors for comparison. Thus the methods described in the book can easily be adapted for gesture recognition. Besides the description of the algorithm, different types of constraints are described:

- **Global constraints:** The global constraints limit the field in which an optimal path is searched. The global constraints define for example the limit of stretching and compression of the comparison vector.
- **Local constraints:** Local constraints define which transitions from one time combination (time first vector, time second vector) to which other combinations are allowed. A constraint that is always required is monotonicity. Which means, that it is never allowed to go back in time during the comparison of the vectors.
- **End-point constraints:** It is required, that every recorded gesture starts somewhere after the first frame of the recording and ends before the last frame. The frames before the gesture start and after the gesture end should approximatively match the compared data. These can be feature vectors from the neutral pose of the user.

When calculating the cost for matching two feature vectors, a suitable norm has to be found. In section 6.3 an evaluation of different norms for usage during the calculation of DTW has been done. For the user tracker we chose the euclidean norm with weighting the features by their variance in the training set. This norm provided the best results in the evaluation. For hand tracking the normal euclidean distance is used as the three coordinates are approximately equally important for the gestures we use.

4.2 Feature extraction

The user tracking mode uses relative distances between all the joints above the hip and the angle between the joints as the feature vector. These are the distances that usually change when hand gestures are given. Additionally, smaller obstacles like a chair or a table will not occlude these joints. As users have different arm or leg lengths, the relative distances for the same gestures may vary. To overcome this problem, all relative distances are normalized by the length of the path on the body between the two joints. The distance between the left and the right arm for example is normalized by the length of the two arms. Additionally, angular features are used containing the measured angles at the elbows, the shoulder and the rotation of the arms.

In hand tracking mode, the 3D position of the hand with respect to a gesture starting point is used as feature vector. With hand tracking the user can remain seated as the hand tracker can be started by simply waving the hand.

Using relative distances as it is done for user tracking led to wrong detections for some of the gestures we use for our gesture interface. In our system we use the STOP gesture (see section 2.5), which is one arm stretched to the front of the body. The GO BACK gesture, for which the users stretches one arm to the side, is a completely different gesture but in terms of relative distances, there are only very few differences. As in both cases, the arm which is not involved in the gesture, is resting loosely, the distance between

joints of this arm (hand, elbow, shoulder) approximately equals the distance of the hand involved in the gesture. Also the distance between the gesture hand and the head or the torso position (see figure 2.1 for tracked joint positions) is about the same for the two gestures. To avoid such wrong detections, we added angular information to the feature vectors. We calculate the angles at the elbows, between the arm and the body and the rotation of the arms. For the two gestures mentioned before, this angular information allows for better distinction. The rotation of the arm has a difference of 90 degrees between the two gestures. In this example, the arm rotation is of permanent importance for a reliable recognition between the two gestures. Weighing of the features with more variance between the recorded gesture templates is used to emphasize important features (see section 4.1.2).

For the hand tracker, the gesture starting point is set to the position where the hand was located when the hand tracking started. After that, this position is updated if the hand does not move more than 0.1 mm between two successive frames for twenty camera frames. This settings were made heuristically to avoid updates of the gesture starting point when the user is currently performing a gesture. The system on the other hand is still able to update the gesture starting point if a user has a trembling hand.

4.3 Gesture classification

Gesture classification is done by a k -NN classifier. The classifier matches the online data from the camera to every recorded gesture template. The costs calculated during the matching are then sorted and the classifier takes the k lowest cost values and counts which gesture occurred most. The recognized gesture is then published for usage by our control application.

Depending on the number of recorded gesture templates, the parameter k of costs considered can be changed. As every person recorded every gesture three times, we use $k=3$. If a gesture is performed closely to the recordings of a person in the training set, usually all recordings of this person have low errors (except the person performed the gestures in different ways during training phase).

When comparing between online data from the camera with the recorded gesture templates, different gesture execution speeds are possible. To allow for this, we use Dynamic Time Warping (DTW). With DTW, an execution of the gesture can be slower or faster than the recorded template by a maximum of the used window size. Our algorithm captures skeleton data with a framerate of 15 frames per second. This framerate is sufficient for a reliable recognition of the gestures and uses an acceptable amount of system resources. With a window size in DTW of 30 frames, the gesture could be performed at maximum two seconds faster or slower than the gesture template.

For gesture detection, features generated from live data of the camera need to be cached, as the detection of the gestures can only occur when the gesture has been performed completely. As we use dynamic time warping to allow for different gesture performing speeds, it is necessary to store more data than the actual length of the gesture templates. The gesture templates for user tracking have been recorded with a length of 25 frames. In DTW we allow a time shift for another 25 frames. Therefore feature vectors of 50 frames need to be buffered in order to allow for usage of the full window size of DTW. For hand tracking we also keep the feature vectors of 50 camera frames. As the recordings of the hand tracker are shorter (only 18 frames in average) we allow for a larger timeshift of 35 frames in DTW.

4.3.1 Gesture specific improvement for the COME HERE gesture

The tracking middleware we use for retrieving the hand position data, requires the execution of a built-in gesture to activate hand tracking. As most of these build-in gestures are prone to false-positive detections we used the wave gesture. For activating the hand tracker, the user needs to wave the hand he wants to be tracked. As soon as the hand tracker is started, the gesture recognition takes place. With this setup, parts of the wave gesture may become part of the executed gesture, if the gesture is executed immediately after the hand tracker has been started. This happens because the gesture starting point is set as soon as the hand tracker started. If the user continues waving after the tracking started, the gesture starting point cannot be updated, but is not in the place where the gesture starts. For the GO BACK gesture this was found to be unproblematic, because this gesture contains a similar movement as the wave gesture and therefore only requires a larger movement to the side. For the COME HERE gesture on the other hand, the upwards motion is recorded as a skew movement because of the misplaced starting position.

To overcome this problem we implemented a detection enhancement for the COME HERE gesture, which updates the gesture starting point if necessary. Figure 4.1 shows how this improvement works. As the gesture starting point has not been updated before gesture execution, it is on the right side of the executed COME HERE gesture. The hand returned from this position back to the position where the COME HERE gesture is executed. As there was no waiting period at this point, the gesture starting point was not updated to this position. We therefore check for upward movements that start with a wave like movement at the beginning. This is done by calculating the step-wise area below the line drawn by the hand movement and the y-coordinate axis. Additionally the area of the triangle between the gesture starting point to the current hand location and y-axis is calculated. If the area of the second calculation (direct line from the gesture starting point to the current hand position) is at least 1.3 times as large as the step-wise area, the gesture starting point is updated to the current y position whereas the x position is not changed. The threshold value for updating the starting position has been chosen empirically by performing the

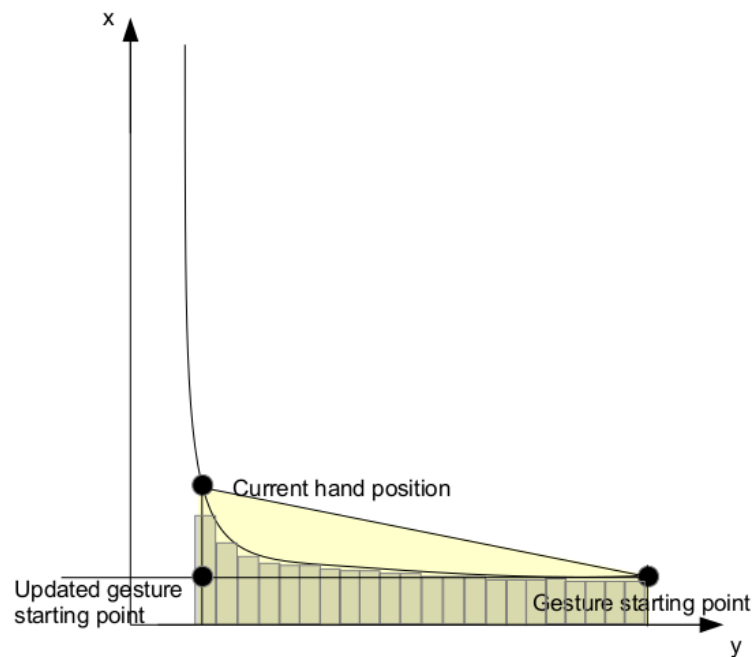


Figure 4.1: Automatic updating of the gesture starting point

COME HERE gesture directly after waving. With the factor used, the main priority lies on the reliable recognition of all gestures with waiting after starting the hand tracker. As a second priority, this improvement can be used.

After updating the gesture starting point, 15 intermediate points between the current hand position and the updated gesture starting point are added. This helps to avoid the usage of the DTW window to overcome the gap between the updated gesture starting point and the current hand position (the feature vectors recorded, starting from the old gesture starting point, become obsolete). If the gesture detection is then run again, it should be able to recognize the COME HERE gesture.

4.4 Analysis of the used features

Xin Zhao et al [19] found three key challenges for online gesture recognition:

- **Viewpoint and anthropometry variations:** The gesture recognition system should be independent in case of variations in the camera viewpoint. Also different body sizes, e.g. the arm length should not influence the recognition.
- **Execution rate variations:** The gesture recognition should work independent of the execution speed of the gestures.

- **Personal style variations:** The gesture recognition should allow for smaller personal variations of performing the gestures.

The features we selected adhere to these challenges:

Viewpoint and anthropometry variations: As mentioned before, the SmartWalker-Tracker uses pair-wise joint distances as features. As these distances are always relative to some other body joint, they are moved together, which makes them viewpoint invariant. Another problem is the different anthropometry of the users. As we use normalization of the relative distances by the length of the path on the body between the joint positions, our system is anthropometry invariant.

Execution rate variations: The difference in how fast a user executes a gesture is an important criterion. The variations here are very high and inclusion of all possible execution rates in the training set of the gesture prediction is impossible. To cope for different execution speeds, a dynamic time warping (DTW) approach has been implemented. Employing DTW, the user can perform the gestures at arbitrary speeds as long as it fits into the time window that DTW allows (see section 4.1.2).

Personal style variations: For a gesture recognition system it is important to carefully define the gestures. But also after explaining to a user, how a gesture has to be performed, there are personal style variations of the gestures. To recognize the gestures, even if they are done in various styles, a large training set should be used. Like this, a lot of possible execution variants are covered. If the differences between the used gesture commands are higher, the recognition can be made more tolerant, which also helps to recognize gestures more easily.

4.5 Training set

The training set for gesture recognition consists of gesture templates. A gesture template is a time series of feature vectors that contains the measurements recorded during a performance of the gesture. For good performance of the system, an extensive training dataset should be used. We recorded gestures from ten people in their 20's and 30's. Every person performed the STOP, the COME HERE and the GO BACK gesture three times which results in 90 recorded gesture templates.

To record gestures, we implemented a tool that extracts the features and records a customisable number of frames from the camera input. When recording gestures, it is important to avoid non-gesture parts in the gesture template. There are two reasons why non-gesture parts should be avoided: Firstly, the detection is slowed, as the non-gesture part after the gesture must also be matched to the template. Secondly, after activating the gesture recognition system, the user has to wait before the gesture can be performed, because the gesture template also contains a waiting period.

As it is not trivial to record gestures such that all recorded frames are used for performing the gesture, we added a pre-processing step for hand tracking. After the gestures have been recorded, the pre-processing step is used to eliminate non-gesture parts from the templates. During pre-processing, the feature vectors that describe the hand position which did not move at least five centimetres from the initial position, are eliminated. Likewise, feature vectors at the end of the gesture that do not contain any movement are deleted.

4.6 Determining the user position

After the COME HERE gesture has been given, the robot has to move to the person that gave the command. Therefore the current position has to be detected. Running user tracking, this is a simple task as all body joints are tracked. In this case, the position of the knees are used to calculate a position 40 cm in front of the user. If hand tracking is used, it is more difficult to find an appropriate position for the robot to stop. The COME HERE gesture consists of an upwards movement of the hand. Therefore the hand position at the time of the gesture detection provides an approximate position of the user. As this gesture can be performed in front of or besides the body depending on personal preference, errors in the target position of the robot can occur. To overcome this, the face position of the user is used (see section 5.1.1) to determine the target position of the robot.

5 Face detection and recognition

For locating faces, face detection can be used. Its goal is to find faces in the camera image, locate them and extract them from the image. After that, face recognition can be applied to find out, if this extracted face belongs to a known user in the database.

5.1 Face detection

During the testing of the first versions of the SmartWalkerTracker, the problem arose, that the walker started moving without intentionally giving gestures. As the gestures are designed to be very easy, they can be given inadvertently. The idea of face detection is therefore twofold. Firstly, face detection should estimate the attention of a user towards the robot and second, it should deliver the face position to be used as goal position when the robot is called.

For the face detection, the OpenCV implementation of the Viola and Jones face detector [15] is used. Haar features are calculated using so called integral images that contain black and white regions arranged as a line or edges. The integral images are used as a pattern for the following calculation. For calculating the features, the pixel below the black parts of the integral image are summed up. Afterwards the sum of all pixel below the white region of the integral image are subtracted. This calculation has to be done at all possible locations in the large image, which leads to a very high number of features. The problem is that too many features exist and it is not possible to use all of them during the detection phase. Therefore the features are tested during the training phase, where positive images (images of faces) and negative images (images of other objects) are read. The features that have the highest variance between the positive images and the negative ones are chosen. Starting from a large amount of possible features, the best 6000 features are taken for image patches of 24 x 24 pixels.

To improve the speed of the detection, a cascade is used. The goal of the cascade is to successively eliminate non-face images (or patches out of a larger image) at very high speed. Therefore the images are processed by a cascade of classifiers. The first classifiers apply only a very low number of features that determine if this image could be a face. If this is the case, the image is processed by the next classifier of the cascade. The Viola and Jones face detection cascade has 38 stages where as the first stage starts by checking for

only one feature. Successive stages contain more and more features and the images that passed all the stages are then classified as positive face detections.

This implementation of OpenCV allows for fast face detection in the regular RGB image coming from the PrimeSense sensor. This detection algorithm is quite fast for most applications. For SmartWalker it is required that face detection also works with rather long distances of up to 3.5 meters (which is the specified limit of the camera). In order to make face detection work at this distance, high resolutions are required, which lowers again the face detection speed. Our current solution works with the maximum supported resolution of the Primesense Carmine sensor, which is 1280 x 1024 pixels.

To further improve the face detection speed, a method described in [1] has been implemented. The authors of this paper use the depth information of an RGB+D camera to restrict the area in which faces need to be searched. They call this additional depth information depth cues. First of all they use this information to limit the scales at which a face needs to be searched and secondly they exclude regions with too large distance to the camera from the search. A similar approach has been used in our work. OpenCV allows the implementation of own masks which are then used during face detection. A mask specifies the areas in which the algorithm has to look for faces, whereas the rest of the image is not considered. To create such masks a DepthMaskGenerator has been developed. This class uses the depth image of the camera to filter out regions where no face has to be detected. The filter gets for every pixel of the rgb image the distance by reading out the appropriate pixels from the depth image. As a lower resolution of the depth image is used than for the RGB image, the depth image is first scaled to the size of the RGB image and then the distances for every pixel of the RGB image is read. If a pixel is farther distant to the camera, as the depth sensor can recognize, then the pixel is marked as background and therefore not used for face detection. If the pixel is closer, the pixel is considered for face detection. This mask improves the speed of the face detection for larger rooms if there is no wall or larger object behind the user.

Face detection regularly leads to false detections. To eliminate these, more depth cues, also described in [1], can be used. If a face is detected, the depth information of the camera is used to calculate the size of the image. For this, the face bounding box is calculated which contains only the frontal region of the face (e.g. without ears or the wider back of the head). Then the width of the face (the calculated bounding box) is checked if it is larger than 8 cm and smaller than 23 cm. If this is the case, the face detection is considered as a valid detection, otherwise the detection is discarded. By the help of this validation, objects that differ significantly from regular face sizes, are ignored. Figure 5.1 shows an example of false positive detections, which can be identified by the help of the depth information. The face images in the prospectus are detected by the haar cascade face detector. As the face images are too small for real faces, they are not considered (red ovals around the detected faces). The real face on the right side is also detected and as the size lies within the defined bounds, the detection is considered as correct (green oval around the face).

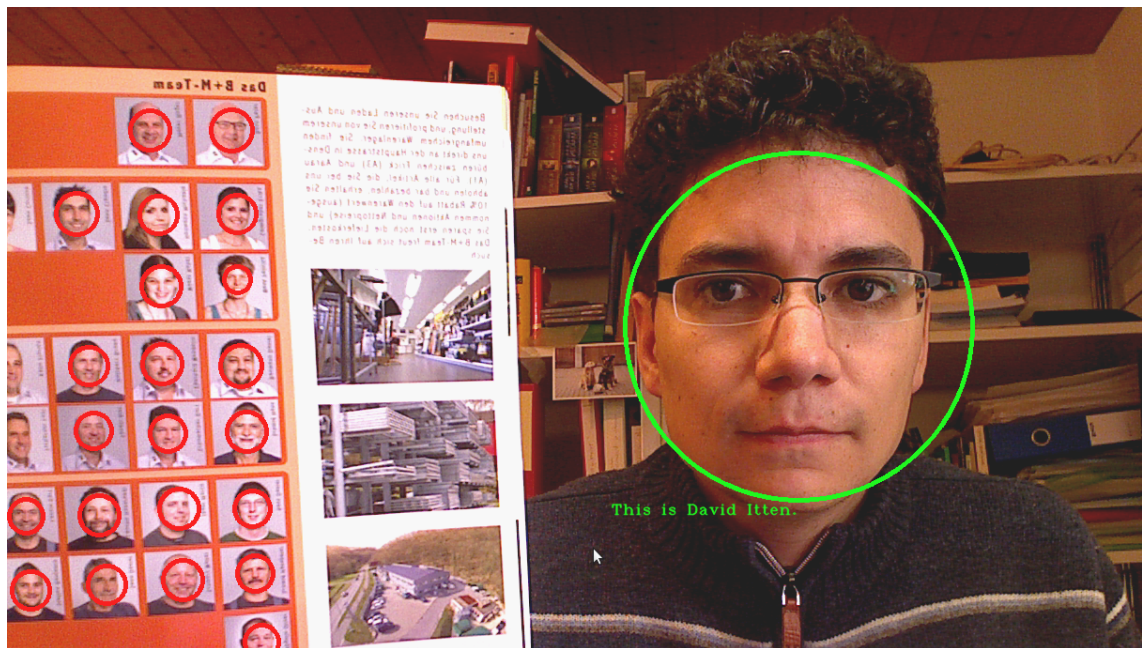


Figure 5.1: A real face and faces from a company brochure.

5.1.1 Face used as goal position

In case hand tracking was used to give a COME HERE command to the robot, the face position is used to determine the goal position of the robot. As the face position leads to a better goal position for the robot than the position of the commanding hand, the face position is preferred for determining the target position of SmartWalker. In order to map a commanding user to a face, faces in the closer region of the commanding hand need to be searched. This is done by using the data published in TF by the hand tracker. If a face has been detected, the euclidean distance between this face and the currently tracked hand is calculated. If this distance is lower than 50 centimetres (currently configured threshold), the face detection is assigned to this hand. For every detected face, two RS_UserPosition messages are sent to ROS. The following data is sent: x,y and z position of the face, the id of the user that was recognized by the face recognition (recognized_id), the id of the tracked hand by the hand tracker (user_id) and an id showing that this position message comes from the face tracker. Then the same message is sent again but this time with the id of user found by the user tracker (user_id). To have only one message sent by the face tracker, the message type could be changed to include both hand tracker and user tracker id's for every detected face. Two messages have been preferred, to allow for other tracking variants that might be used later (e.g. with the help of the laser sensor).

5.1.2 Attention estimation

On the touch screen interface of the SmartWalker, the user can choose if the walker should only react on gestures given by a person that is looking towards the robot. A similar idea has been described by [5]. A group of unmanned areal vehicles (UAV) where measuring how well a face looks into the direction of this robot. Every measurement was registered as a face score. All UAV's then sent their face score to a central server which then used the best value (if above a threshold) to tell which UAV should get the command for execution. In our case only one robot with one sensor is available. As we are using the haar-based face detection cascade of OpenCV, there is no confidence value returned that could be used as face score. To overcome this, two methods can be used. To calculate a face score, the `minNeighbours` parameter of the `detectMultiScale` function in OpenCV is set to 0. In this way, every real face is detected several times, as the face detection is ran with very small shifts all over the image. Then the number of detections in the neighbourhood can be counted and used as face score. As for our purposes the actual face score is not important, because we do not need to compare it with other face scores, it is sufficient to have a threshold to only detect faces that match the trained faces well enough. Important for this to work is, that the face detector is trained on frontal face views only. Therefore the `minNeighbours` parameter can be used directly to only recognize faces that look towards the robot. In the documentation of OpenCV, the use of `minNeighbors = 3` is proposed which works well for a reliable detection without double detections. In this work, `minNeighbors = 10` was used, to enforce the frontal view constraint. With this setting, faces are recognized reliably, but must be oriented towards the camera, in order to be recognized. Figure 5.2 shows the face detection in action. If the user presents its face frontally to the camera, the face is recognized. The more the head is turned away from the camera, the detection stops.

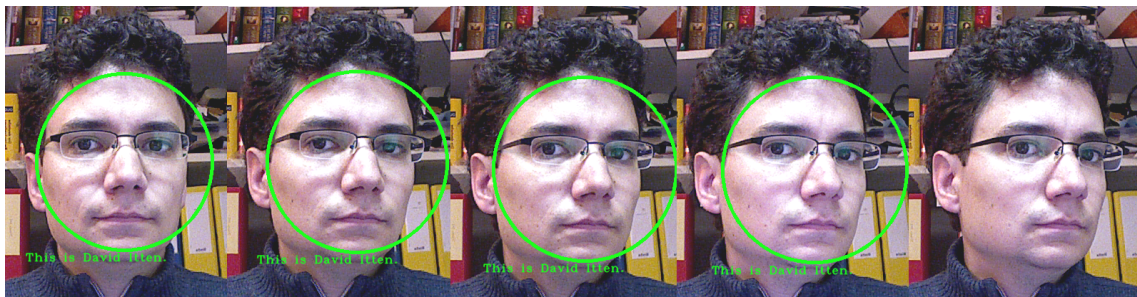


Figure 5.2: Face detector trained with frontal face images.



Figure 5.3: Debug output window of the face recognition

5.2 Face recognition

A more restrictive way to limit the number of users that can operate the robot is user recognition. This part has been cascaded to the face detector. In this way the face detector detects faces out of the camera image. If a face was detected, it is cut out of the image and forwarded to the face recognition. The face recognizer then tries to map this image to one of the pre-taught faces in the database. For face recognition, the Local Binary Patterns Histograms (LBPH) [7] Face Recognizer of the OpenCV library was used. LBPH uses LBP operators to capture also very fine grained details in images by comparing single pixels with its neighbourhood, which are placed in a circle around the pixel in observation. To cope for different scales of details in an image, these LBP operators can be applied with different sizes. Different LBP operators are used to record different kinds of details. There are operators for details as spots, lines, edges and corners. As these binary operators record differences in intensity between pixels in the image, they are invariant to different lighting conditions. To allow the comparison of images from a database to the captured faces from the camera, feature vectors have to be created. This is done by dividing the LBP image into local regions and calculating histograms of them. The final feature vector is created by stacking the data of all local histograms on top of each other to form a larger vector. When a recognition is done, the feature vector from the current camera input is compared to the feature vectors of all stored images. All faces in the database are stored as face views with a size of 200 x 200 pixels. The images are cropped with the help of face detection. If the faces of the stored user are not well recognized, manual cropping of the faces could help to improve detection rate. It is very important to crop the faces very tight and to exclude as much from the background as possible. If too much background is part of the training data, the background has too much influence of the detection. It is possible to store several images for every user. For good detection rates, about ten images per user should be trained and stored in the face recognition database.

5.2.1 Teaching faces to the face recognizer

To facilitate the teaching of new faces to the face recognition, an application has been created. This application grabs the RGB image of the camera and again uses the Viola and Jones face detector of OpenCV to detect the faces within an image. For the training of the face detection, it is required, that only the face that should be trained to the face recognition is visible in the camera image. Therefore only the first recognized face will be marked in the output window. As explained in section 5.1 the face detector is only trained on frontal face images. Therefore, the images that are stored during the training of the face recognition should also be frontal or with a very small angle. For a reliable recognition of the faces, every trained face should be recorded 10 - 15 times to the database with a small movement of the head.

6 Evaluation

6.1 Performance analysis

To evaluate our gesture recognition system, a performance analysis of the system has been made. For this experiment we asked the test persons to perform a COME HERE, a GO BACK and a STOP gesture at four different distances. If the gesture has not been recognized, the test person repeated the gesture several times until the gesture was recognized or we skipped this gesture at the current distance. The test has been done twice, once for the user tracker and once for the hand tracker. For testing the user tracker, we asked the person to stand for this test. During the experiment of the hand tracker, the test candidates were sitting. Additionally we checked, if the face detection was able to recognize the users face at all the distances tested.

13 persons participated in this study (two elderly persons were not able to participate in the user tracker experiment, as they cannot stand without help). We trained the hand tracker with 10 persons from our group where every person performed all gestures 3 times with one hand. The GO BACK gesture has been trained with both hands as the movement of the second hand goes to the opposite direction. The user tracker has been trained with three persons, performing the gestures five times for each hand. This leads to 30 gesture templates per person.

In figure 6.1 the recognition rates at the distances of 2.0, 2.5, 3.0 and 3.5 meters are shown for hand tracking. Except for the GO BACK gesture, shorter distances seem to work better for hand tracking. For the GO BACK gesture, the recognition rate did not change much. As this gesture consists of the longest hand movement, it seems to be recognized also with larger distances. For all gestures, some test candidates seemed to learn how to perform the gestures for better recognition. As the test always started at two meters, some persons performed better for larger distances. This could explain the small increase of the recognition rate for the GO BACK gesture. The STOP gesture was the most difficult for recognition. A possible explanation could be the difficulty for recognition of the hand for this gesture: As most of the test candidates performed this gesture in front of their bodies, the tracking of the hand was more difficult than the gestures that were besides the body of the person. During testing, the hand tracker sometimes lost the hand when it was too close to the body. The candidates then had to wave again (to restart hand tracking) to continue the test with the next gesture.

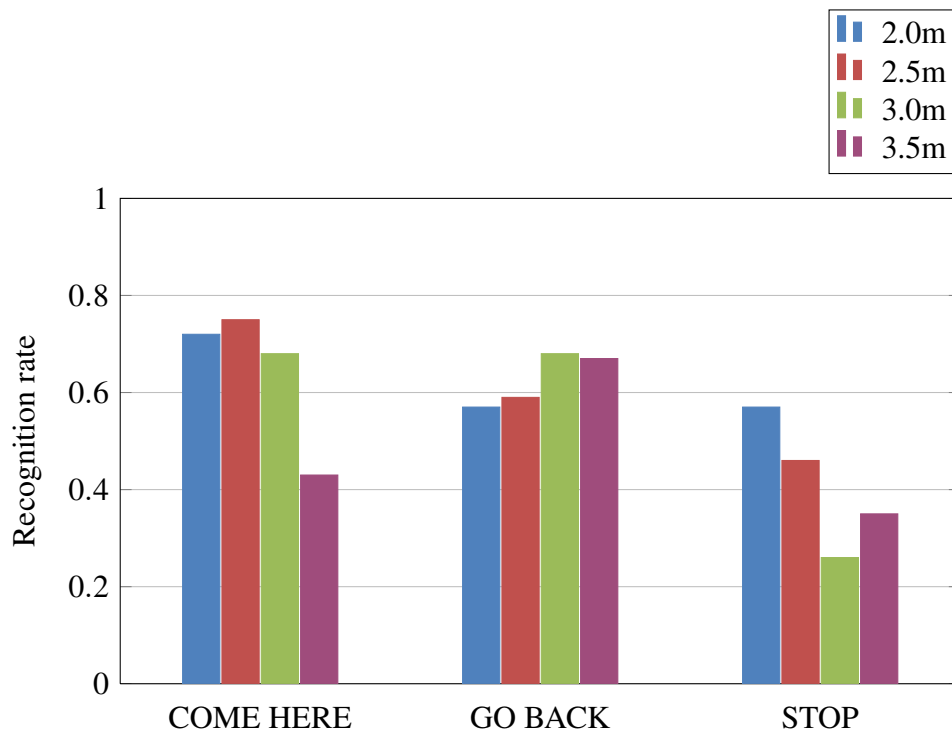


Figure 6.1: Recognition rates of the hand tracker

Figure 6.2 shows the recognition rates for the user tracker. Except for the STOP gesture, the recognition rates for the user tracker are generally slightly higher. Most probably, this is because of the more sophisticated feature vectors used with this tracking technique. Instead of having the 3D displacement of the hand position, relative distances between the joints above hip and the angles at the shoulder and the elbow are used. Except for the STOP gesture, which worked badly with large distances, the distance to the camera seems to have a rather small influence. As for hand tracking, the COME HERE and the GO BACK gestures worked about equally well whereas the STOP gesture worked worse.

Table 6.3 shows the precision of the two tracking techniques. The precision of the hand tracking is higher than the one for user tracking. The reason for this is, that the hand usually gets lost if a person moves around and the hand tracking has to be started by waving the hand explicitly. For user tracking, the tracker is always running and every movement is tracked. Like this, it can happen more easily, that a false recognition occurs. Most false positives have been recorded for the GO BACK gesture. In hand tracking it happened several times, that a GO BACK gesture was recognized because of the strong waving that has been done to activate the hand tracker.

The face detection has also been tested at the four distances used for the gesture recognition test. The results are shown in table 6.4. Face detection worked for most of the test candidates at all distances. For one person, the face detection did not work at all. After repeating the test without wearing glasses, the detection worked for all distances.

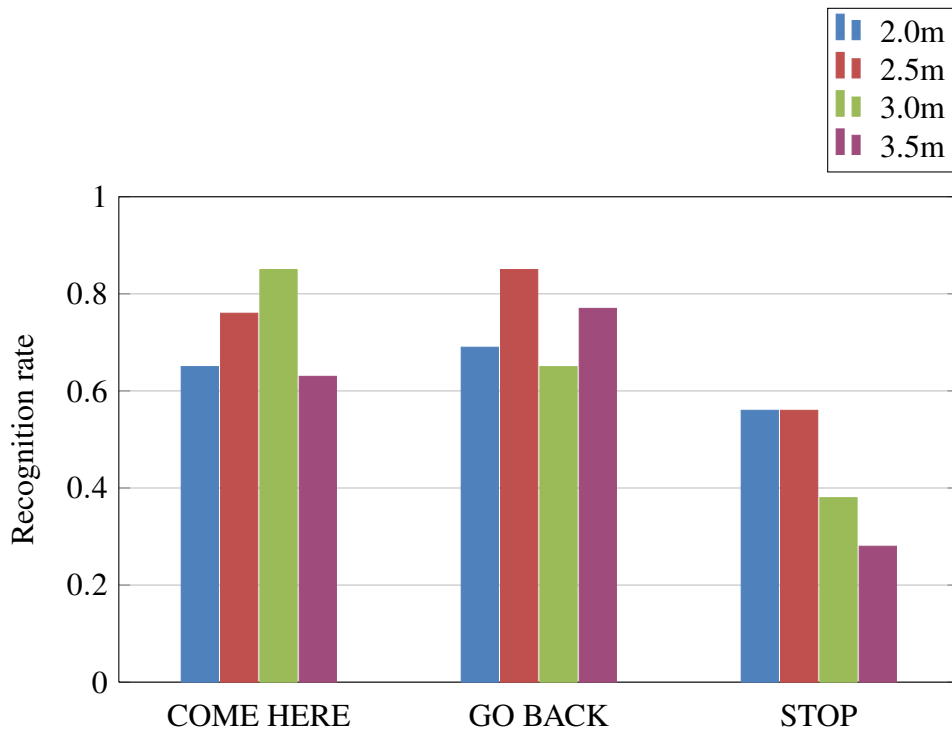


Figure 6.2: Recognition rates of the User tracker

	COME HERE	GO BACK	STOP
Hand tracking	93%	89%	94%
User tracking	90%	78%	88%

Figure 6.3: Precision of the SmartWalkerTracker

The recognition rate varied between 79% for a distance of two meters to the camera and 93% for a distance of 3.5 meters. For this test a minNeighbor setting of 5 has been used. This means that five, slightly overlapping detections, are required to detect a face. During this experiment we saw, that some faces are detected faster than others. The faces that are detected more easily are allowed to look in the direction of the robot, other people had to exactly look at the robot to have their faces detected. For the face detection we also tested how much light is required for the face detection to work. We generally found that strong sunlight from the back or the side of the camera poses problems, because the faces are then over illuminated. The general lighting conditions of the room are less important as the camera adapts automatically and brightens the camera image if necessary. Table 6.5 show the results we collected at different distances under various light conditions. There was a row of lamps starting at the position of the camera and ending behind the test person, which ensured an equal lighting at every position. The light intensity has been measured with a lux meter at head position. The first measurement with 340 lux has been done with a good office light. With 7 lux, the face detection was still working but the camera was not able anymore to brighten the image to the normal level. At a light intensity of 2 lux, the camera was not able to adapt anymore and stopped publishing data.

6.2 Study in elderly homes

For evaluating the gesture recognition system, we visited five nursing homes. 23 residents tested our SmartWalker. 14 men and 9 women participated in our study. 16 of the candidates never used a computer or a Smart Phone, 6 use such devices and one person uses a computer and a Smart Phone every day. Only three persons were able to walk without any assistance, 6 persons use and 14 persons require walking assistance. 14 persons always use their walking assistances when they walk outside of the building, 5 persons use them regularly outside and 4 persons can walk outside without any help.

6.2.1 Setup of the study

For the gesture recognition tests in this study we used the hand tracker with the same training data as for the performance evaluation. The training set has been recorded with 10 people from our group at ETH, performing every gesture three times. The hand tracker has been trained with one hand only, except the GO BACK gesture that has been trained with the second hand as well.

2.0m	2.5m	3.0m	3.5m
79%	89%	93%	93%

Figure 6.4: Recognition rates for face detection

Lighting [lux]	2.0m	3.0m	4.0m
340	✓	✓	✓
220	✓	✓	✓
80	✓	✓	✓
23	✓	✓	✓
10	✓	✓	✓
7	✓	✓	✓
2	✗	✗	✗

Figure 6.5: Face recognition with different light conditions



Figure 6.6: An elderly person using SmartWalker

Our study was split into three sections (for reference the original questionnaire can be found in the appendix A.3). First we asked the participants some general questions about age, aids they use for walking and their skills with computers and smart phones. All answers have been recorded on a 5-valued likert scale. After the first set of questions, we asked the participants to use our walker as they would use a normal walker and walk around in the room. In a second set of questions, we then asked the people about their general impression of the walker. Furthermore we asked questions about the size, weight and easiness of using the walker. In the last part of the test, the elderly were asked to call the robot by performing the "COME HERE" gesture and sending it back to its initial position using the "GO BACK" gesture. In the last set of questions, we asked again about the general impression of the walker and how well the people were physically able to perform the gestures. We also asked if it was difficult to understand how the gesture recognition system works and what their impression of how well the system understood the gestures was.

6.2.2 Research questions

The goal of the performed study was to find answers to the following questions:

- How useful is a walker that can approach its user?
- Are the users capable of performing the necessary gestures for commanding the SmartWalker?
- Does the performance of the gesture recognition system change the users general opinion about the walker?
- Can the users easily understand how they have to perform the gestures and is there a correlation with the recognition rate?
- Is it comfortable when the robot approaches and does it stop in the right place?
- Does a good working gesture recognition system increase the acceptance of a motorised walker?

Furthermore we collected data about the recognition rates to analyse the performance of the system.

6.2.3 General findings

We asked the elderly if it is useful to have a walker that can automatically come and return back to a predefined position (results in Figure 6.7). About half of the persons liked this function. Another 5 persons found this function rather useful and only 4 persons

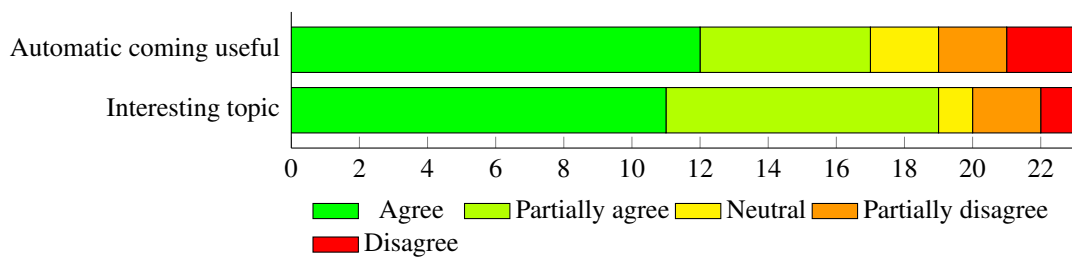


Figure 6.7: The test persons found the function that the robot can approach automatically useful. The topic of robotic walkers was interesting for our test candidates.

found this function rather useless or completely useless. We also asked, if they find the topic of robotic walkers interesting. The scale was ranging from very interesting to very dangerous. 19 of the 23 candidates found the topic very interesting or interesting. One person was neutral about the topic whereas the remaining 3 persons judged robotic walkers as dangerous.

The function which allows the robot to approach users seems to be generally liked by the test persons. In section 6.2.8 test candidates also mention problems that they see for the gesture recognition system. The test candidates seem to be interested in this new technology and are willing to test a robotic walker.

6.2.4 Change of general impression about the robot

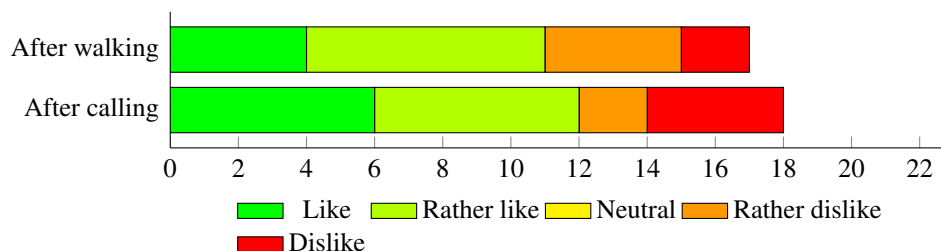


Figure 6.8: General impression of the walker after trying it without any assistance and after using the gesture based interface.

As shown in figure 6.8, the experiment helped the test persons to form their opinions. After testing the gesture recognition system, the number of participants that responded "rather yes" or "rather not" reduced whereas the values increased for "yes" and "no". Before and after testing the gesture recognition, nobody answered "neutral". The number of persons that answered "yes" or "rather yes" and "rather no" or "no" remains stable (one blind person did not want to answer the question before she tried the gesture recognition). The relative amount of candidates that are positive about the SmartWalker did not change significantly. However there are people that changed their opinions after testing the gesture

recognition system. 5 persons that initially answered positive to the question whether they like the robot, changed to a negative answer. But also the opposite happened for 2 persons which changed from a negative answer to a positive one.

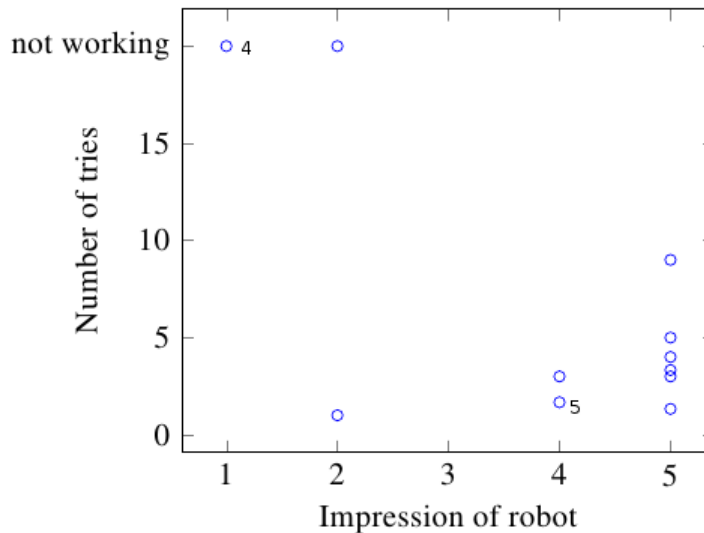


Figure 6.9: Recognition rate versus liking the robot.

In figure 6.9 the average number of tries for a successful recognition is shown versus the impression of this test candidate of the robot (numbers next to the dots show the number of data points at this position). Persons with only a few tries until the robot recognized the gesture, also liked the robot more. Except one person, all candidates with 5 or less tries in average liked the robot with a rating of 4 or 5 (highest score) on the Likert scale. Candidates where at least one gesture did not work after several tries, rated the robot 1 (lowest score) or 2.

6.2.5 Evaluation of the stopping position

With two questions we tried to find out if the persons felt comfortable when the robot moved towards the inhabitants and if the position where the robot stopped in front of the persons was good. The intention of the first question was to find out if the speed of the robot and the speech announcement was sufficient for the user to have a good feeling when the robot moved towards them or if they feared being hit by the walker.

This question was asked to 20 persons (all participants for which the COME HERE gesture worked). Except for one person, all candidates were satisfied of how the robot moved towards them. The person that voted to be rather uncomfortable, the robot came too close. Instead of stopping with a distance of 40cm, the test supervisor had to stop the robot to protect the test person. The second question aimed to find out if the orientation and the position of the robot were comfortable in order to reach the SmartWalker. 15 of 20

persons found the position good or rather good, 5 persons found it rather uncomfortable or uncomfortable. In one case the robot went to the person sitting next to the test candidate as the gesture was given with the left arm just in the middle of this two persons. As the wrong face position was used, the robot stopped in front of the wrong person. A second problem is that the robot currently does not support the definition of an orientation when stopping. Therefore the orientation of the robot when reaching the stopping position was sometimes not optimal.

We also asked if more visible or audible feedback during the movement of the robot would be required. 11 people liked or rather liked such an idea, whereas again 11 persons found it rather unnecessary or completely unnecessary. Interestingly there is no significant difference in the answers of visually impaired inhabitants. About half of this group would like to have an acoustic signal when the robot moves, the other half found that they can hear the robot approaching.

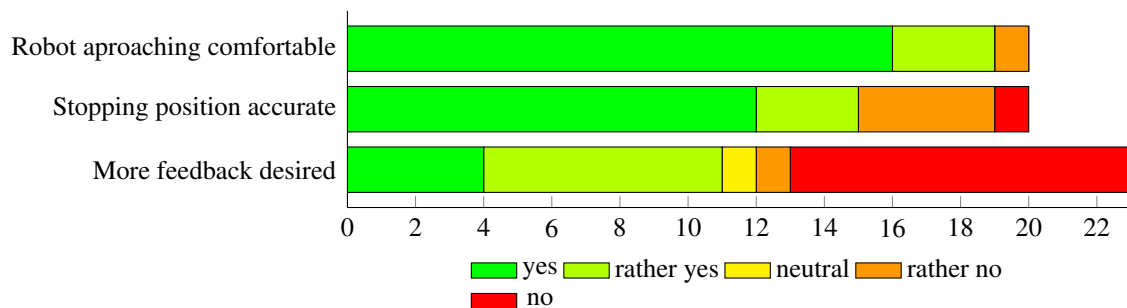


Figure 6.10: Questions asked after the usage of the gesture based interface.

6.2.6 Gestures

For testing the gesture based user interface, the test persons were sitting between 2.5 and 3 m distance in front of the robot. After explaining how they can enable the hand tracker by waving and how to perform the "COME HERE" gesture, we asked them first to call the robot. We asked the candidates to perform the gesture several times until the robot reacted. If it was necessary, hints were given on what should be done differently for the next tries. The test subjects did not use the system before the test.

After the SmartWalker came, we explained how to send the robot back. Then, the test persons were able to try this gesture. Again we gave hints on how to perform the gesture, if the last try was not recognized. We encouraged the participants to try the gestures several times. If they were not successful after 5-6 tries we told them to stop whenever they like.

We asked questions about the difficulty of the gestures. We wanted to know if the gestures can be performed well physically and also if they are easy to learn. We also asked the people if they found the gesture recognition system understood them well.

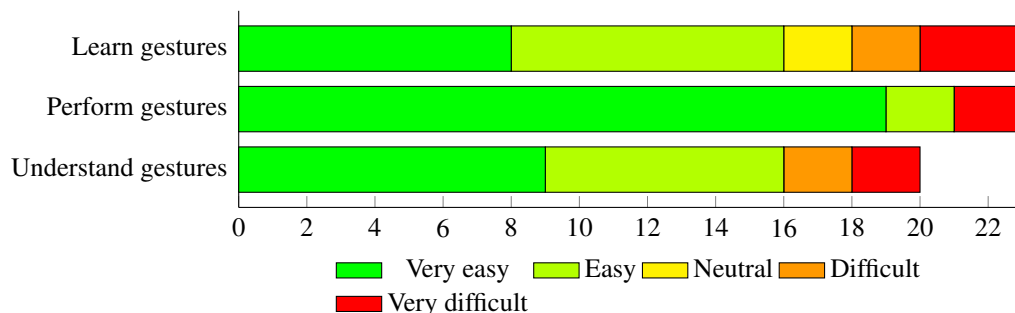


Figure 6.11: Difficulty for learning and performing the gestures. Impression of the test persons of how well the robot understood the gestures.

Most people found the required gestures easy to understand. Only 5 persons found the gestures difficult to learn. We saw that performing the gestures we defined seems to be no problem. Only 2 candidates found it difficult to perform the gestures; one of these people was suffering Parkinson's disease and therefore a gesture based interface is rather inappropriate for this person. Interestingly, the gesture recognition worked quite well (3 tries for COME HERE, 1 try for GO BACK) for this person, even though the person supported the performing hand with the second arm. The questions about the test persons opinion on how well the robot was able to recognize the gestures, was answered mainly positive. Only 4 candidates found the robot did not understand them. For three of them, at least one command was not recognized after several tries and the test had to be aborted.

Figure 6.12 shows the average number of tries that were required for a gesture recognition. The average recognition rate of the COME HERE gesture was 44%. For the GO BACK gesture it was 38%. These rates are inferior to state-of-the-art gesture recognition systems. The fact, that the elderly persons often made the gestures different from what we showed them, was a challenge for our system. Additionally, some persons were not able to turn their palm towards the camera as we did for recording and in order to have a good tracking of the hand position. Another problem was that some person required more tries until they understood how to activate the hand tracker (for which we use NiTE and is separate from our gesture recognition). This is done by performing a wave gesture, namely waving the hand from the left to the right and repeat this again. Two of the test persons only moved the palm instead of the hand which was not recognized by NiTE to activate the hand tracker. Another problem was that some persons did not understand that we switched to a second test and performed again the COME HERE gesture instead of the GO BACK gesture.

The design of the gestures seemed quite good, as they were easy to perform for the participants and they do not require sophisticated movements. The gestures differ signifi-

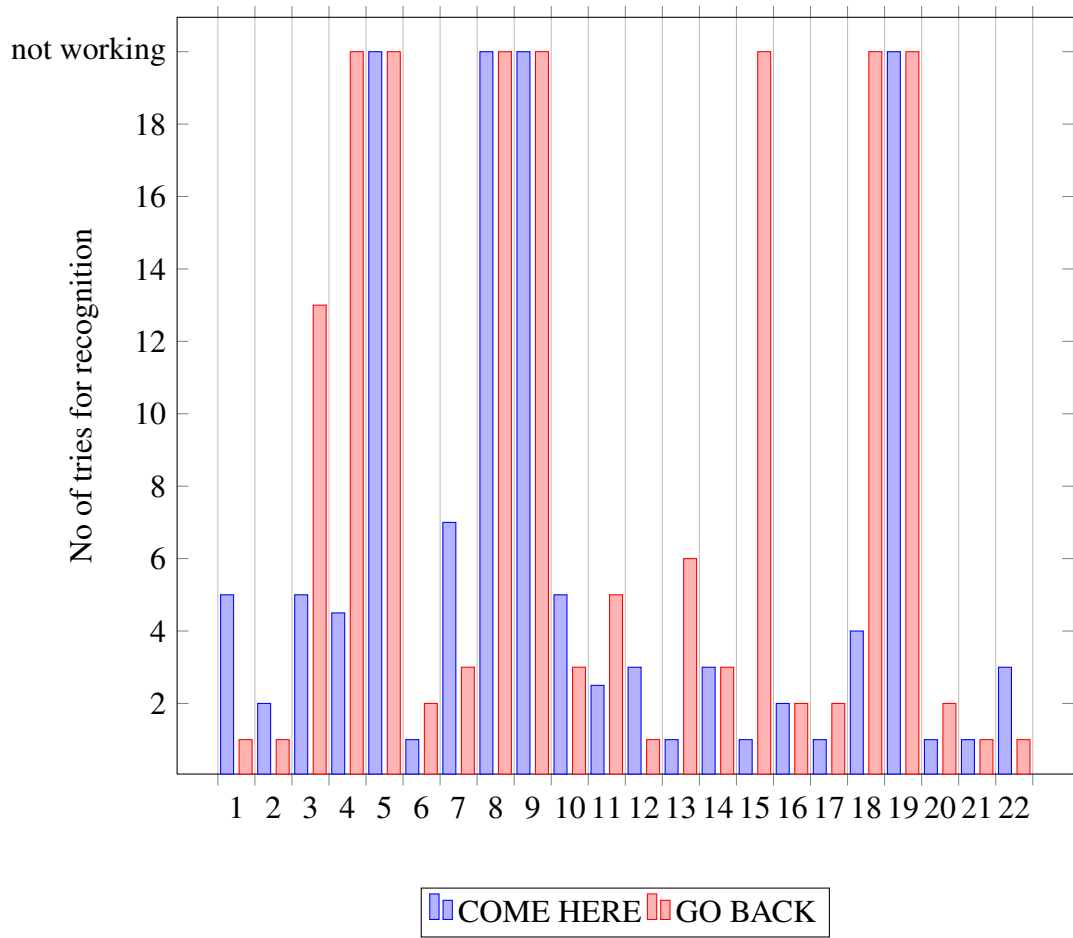


Figure 6.12: Recognition rates for the test persons

cantly from each other. Therefore the gesture recognition can be configured more tolerant, which also helps to recognize gesture executions, which deviate from the specified gesture.

6.2.7 Software adaptations

During the first week of testing in nursing homes, gestures were often not recognized because the inhabitants did not wait for 1 or 2 seconds after waving, before performing the gesture. However, this waiting time is necessary for two reasons: Firstly, NiTE has to start the hand tracking, which needs less than a second. Secondly, since the stored gestures templates are used directly from gesture teaching, it often happened that there were frames without any hand movement before the actual gesture. This was because it was not always possible to perform the gesture to fit in the number of frames that are defined for recording. Often there was a period without any hand movement before and after the real gesture. As this waiting period was also part of the gesture template, a gesture was not well recognized if it was performed without enough waiting time before and after the gesture.

We can not influence the start-up time of NiTE as it is closed source. However, the second problem can be eliminated. This has been done by adding an automated post processing step to the learned gesture templates. All gestures are processed and periods without any movement before and after the gesture were cut from the gesture template (see section 4.3.1). After applying the described post processing to the already created gesture templates, we had a second week of experiments in elderly homes and were therefore able to compare the recognition rates before and after the change:

In figure 6.12 we can see how many tries every test subject needed for a successful recognition of the tested gesture. In the first week of testing, we tested the candidates 1 to 12, the other persons did the test in the second week after applying post processing. If we calculate the recognition rate of the two test groups separately we get the following data:

The data shows a big improvement in the detection of the COME HERE gesture. Before applying the changes we had a recognition rate of only 27%. Afterwards this rate increased a lot to 63%. The post-processing of the gesture templates allowed us to eliminate the waiting time after activating the hand tracker from approximately 2 seconds to nearly no required waiting. If the hand is not moved too fast, no waiting is necessary any more. When testing the improved version in the second week of testing in the elderly homes, we increased the error threshold for the COME HERE gesture. We did not increase the threshold for the go back gesture, as a strong waving for activating the gesture recognition

Gesture type	Without post processing	With post processing
COME HERE	27%	63%
GO BACK	37%	39%

Figure 6.13: Recognition rates for post-processing

could be confused with the GO BACK gesture. The higher threshold for the COME HERE gesture led to two false positives for the same person. This person did long wave movements for activating the hand tracker with rather large displacement in x and y direction. The COME HERE gesture was therefore recognized just after the person stopped waving.

6.2.8 Additional comments

As a last question of our study, we asked every participant, what should be changed or added to the robot, to make it better. We received valuable feedback for feature improvements. The following comments have been made concerning the gesture control:

- **Remote control** A person mentioned the problem, that the walkers are placed outside the dining room, from which there is no intervisibility to the parking positions of the walker. Therefore this person proposed to have a remote control with a button to call the robot.
- **Identification of the user** One person wanted the SmartWalker to know its owner. Otherwise a lot of walkers could react to a command, when they are placed next to each other. (This function is already available, but was not tested in the elderly homes)
- **Recognition of gestures from all sides** A blind participant remarked that it is not possible for him to know if the robot is oriented towards him. Therefore the robot should turn its camera automatically to recognize gestures from every direction.
- **Good working functions** One person mentioned that a device for elderly should not have too many functions, but the basic functions should work reliably.

6.3 Evaluation of different error measures

For choosing an error function for the comparison of feature vectors, the following error function have been evaluated:

- **Error sum** This was the first error function we used during the development of the gesture recognition. It calculates the absolute value of the difference for every feature of the feature vectors under comparison. Then it sums up all this differences.

- **Cosine similarity** The cosine similarity, opposite to error functions, is increasing if two vectors match better. It is calculated with the following formula [16]

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n (A_i)^2} \times \sqrt{\sum_{i=1}^n (B_i)^2}}$$

- **Euclidean distance** The euclidean distance calculates the distance between two feature vectors in the n-dimensional space (with n, the number of features). The following formula is used: [18]

$$d(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_i - q_i)^2 + \dots + (p_n - q_n)^2}.$$

- **Euclidean distance with weighing features by their variance in the training set** This error function uses the normal euclidean distance but weights the features by the variances they have in the training set. This helps to give more priority to features which are more distinctive.
- **Mahalanobis distance** The Mahalanobis distance is used in statistics where it calculates the distance in a multidimensional vector space. The covariance of two vectors is used for weighting. We used a slightly modified version of this distance in which we replaced the covariance of the points with the confidence of the data that we received from NiTE.

For evaluating the error functions, a recorded camera sequence has been used, in which the supported gestures have been performed twice. At the beginning of the sequence, the COME HERE gesture has been performed twice. It was followed by two executions of the STOP and the GO BACK gestures. All the tests have been performed in hand tracking mode.

The Mahalanobis distance has been tested during the implementation of the SmartWalkerTracker but is not evaluated here as the confidence values of the tracked joints did not contain enough information. The tracking middleware returns a confidence value of 0 for inferred joints (which are currently not visible to the camera) and 0.7 for the tracked joints. As there are no intermediate values, it turned out that the results are better without using the confidence values. Additionally NiTE infers the positions of occluded body joints by its neighbours. This positions are often quite precise, which leads to better error values if they are used for the error computation.

The error sum was an easy measure for doing the first tests of the gesture recognition. But as shown in figure 6.14, especially for the STOP gesture, the error for the COME HERE gesture was also low. As this evaluation was performed under good conditions (2.5 meters to camera), the recognition would be feasible in this case. Under more challenging conditions or with test persons that perform the gesture differently, the recognition wouldn't be possible anymore.

The evolution of the similarity values for the cosine similarity are shown in figure 6.15. The cosine distance is a good error measure for angular feature vectors. In contrast to the

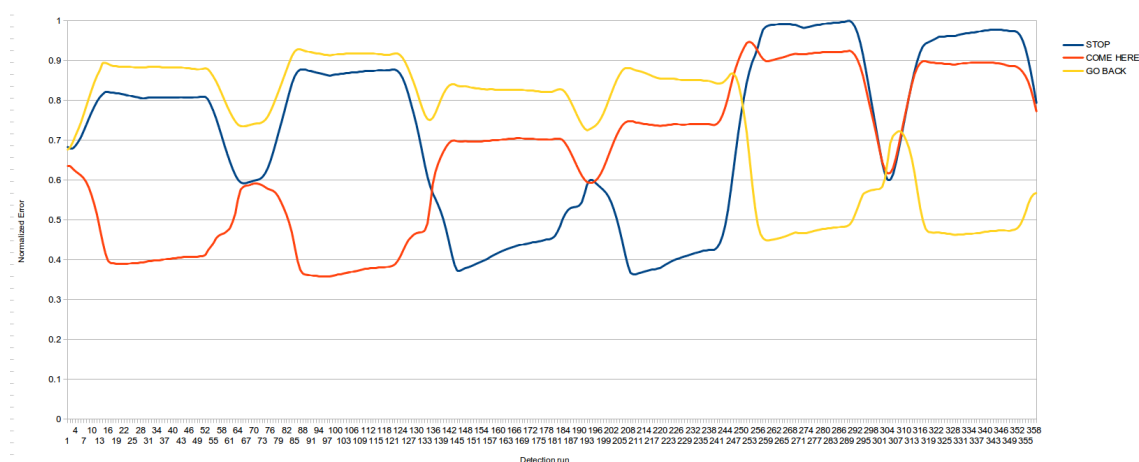


Figure 6.14: Recorded gesture sequence by using the error sum

error functions, this similarity function signals occurrences of gestures with higher values. Therefore the two COME HERE gestures that we tested at the beginning are represented with higher values. The STOP and the GO BACK gestures have quite similar values, especially if the GO BACK gesture is executed, which makes this error norm inappropriate for the hand tracking.

The Euclidean error function is especially useful for spatial information. For hand tracking, the 3D positions of the hand with respect to a gesture starting point is used. Therefore the euclidean error function seems to be appropriate for the usage with the hand

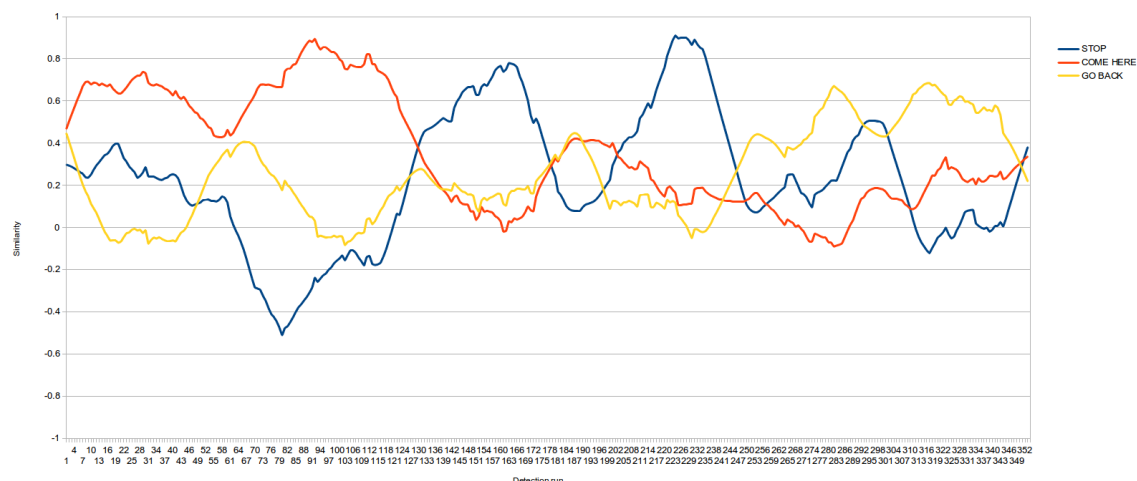


Figure 6.15: Recorded gesture sequence evaluated using the cosine error function. Average similarity values for all templates of a gesture class are drawn.

tracker. As shown in figure 6.16 this error function has a desired behaviour that was not visible as clearly for the other error functions we evaluated before. If a gesture has been performed, the error for this gesture decreases massively and at the same time, the error of the other gesture classes increases. This makes a classification of the live data to the gesture classes more precise.

As a last variant we evaluated the euclidean distance with a weighting of the individual features with their variance in the training set. This error measure is very useful for the user tracking, as there are much less angular features than relative position features. For hand tracking, the three coordinates of the hand position, which are used as features, are approximately equally important. Therefore a weighing of the features does not make sense. In figure 6.17 the recorded video sequence has been used to determine the error values with this method. A small improvement of the STOP gestures can be seen. On the other hand, the error function for the GO BACK gesture gets worse.

Based on the evaluations done in this chapter, we decided to use the euclidean distance with weighting features based on their variance in the training set for user tracking. For hand tracking the normal euclidean distance is used as an error function for DTW calculation.

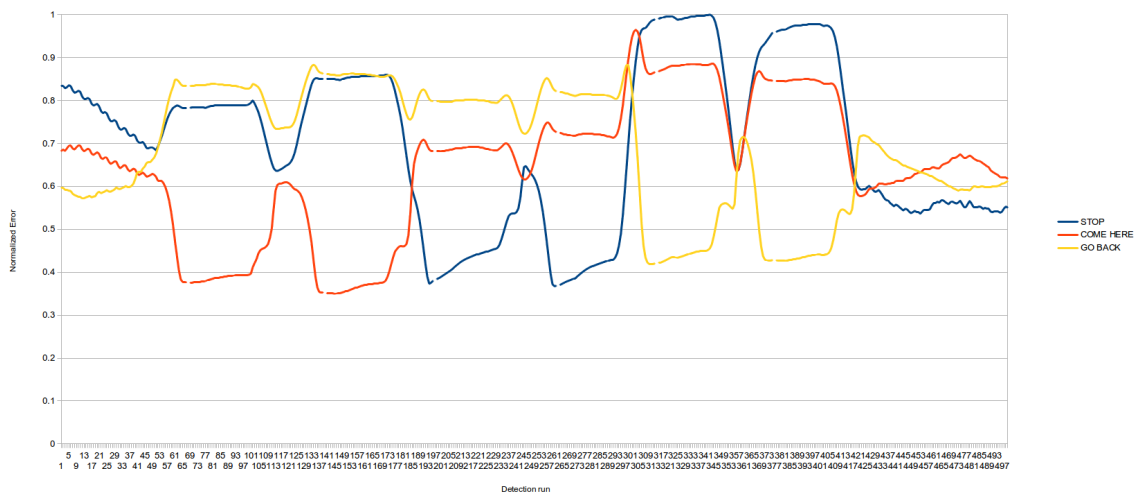


Figure 6.16: Recorded gesture sequence evaluated by using the euclidean error function. Average errors for all templates of a gesture class are drawn.

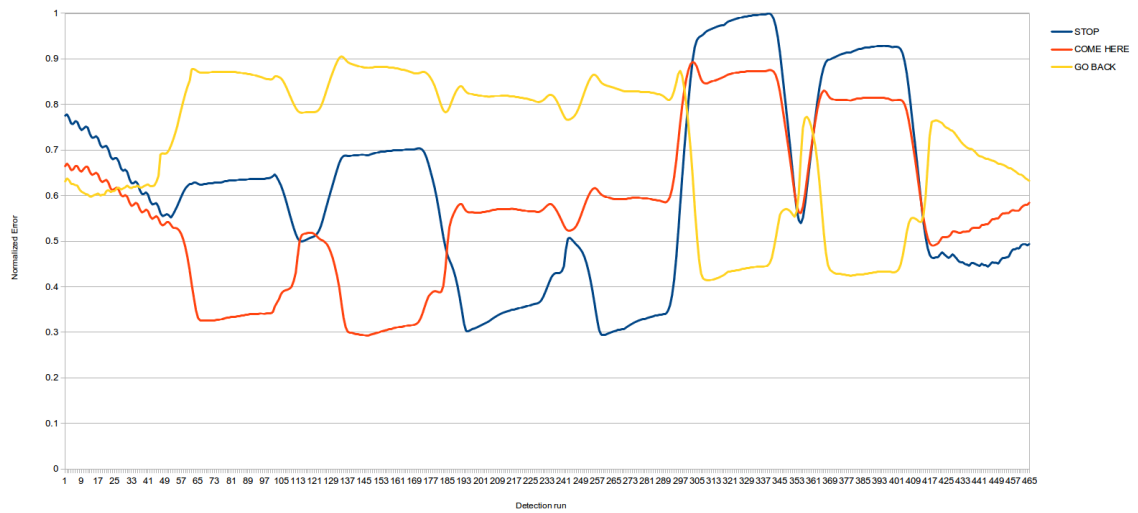


Figure 6.17: Recorded gesture sequence evaluated using the euclidean error function and weighing features with their variance in the training set. Average errors for all templates of a gesture class are drawn.

7 Related work

Several papers have been published about the subject of general hand gesture recognition. Suarez and Murphy [12] categorized and summarized different techniques used for hand gesture recognition with the help of RGB+D cameras. Zhou Ren et al. [8] measures the dissimilarity of different hand shapes using Finger-Earth Mover's Distance (FEMD) to detect static gestures. Van den Bergh et al. [14] used the OpenNI library to detect hand positions. Image segmentation was used to get the arm orientation with the aim of recognizing pointing commands to direct a robot. Kishore Konda et al. [3] used a convolutional neural network to extract the arm movements of a human guide to manoeuvre a robot. Another project [4] created a walker for visually impaired persons called PAM-AID. This device also has a manual and an assistive mode. In manual mode the device provided information of the environment with a speech interface. In automatic mode the device was able to guide the user to a target by avoiding obstacles.

Our walker provides a novel gesture based user interface and supports autonomous movement. The used gestures have been optimized for, and tested with elderly persons that might have physical constraints. The focus of our system lies in the easiness of usage, the natural design of the gestures and the recognition of gestures which are done not precisely as specified. Furthermore, our gesture recognition systems does not require a well defined setup and is supposed to work in many different environments and lighting conditions with distances up to 3.5 meters.

8 Conclusions and future work

8.1 Conclusions

We have presented a gesture based interface for the SmartWalker. The implemented solution allows the control of our walker by instrumented persons. Furthermore, natural gestures have been used that are intuitive to use and easy to remember. As the system is developed for the usage by elderly persons, it does not require an exact execution of the specified gestures. Our system has been evaluated with an in-house performance evaluation and with an experiment in nursing homes.

During our test we found that a gesture interface seems to be an intuitive way of controlling the SmartWalker. Inhabitants of nursing homes generally liked the fact that a robot can move towards them after giving hand gestures. A gesture interface has of course also some drawbacks. One of the main problems is the rather small operating range of only 3.5 meters and the restriction that the SmartWalker needs to be placed such that the user is always in the visible range of the camera.

The current implementation of the SmartWalkerTracker relies on NiTE for user and hand tracking. The advantages are that NiTE offers an easy to use and very precise interface for accessing the position data for the users joints and the hand center for hand tracking respectively. As it entirely relies on the depth data it is also not sensitive to changing lighting conditions. As NiTE is released as a closed source project, no improvements can be made for aspects which are not entirely satisfactory for our usage. One problem is that a fixed camera position is assumed, which is not the case for SmartWalker. When the walker is moving, users are often re-detected as a different user. Sometimes also objects in the background are detected as users as they seem to move in the perspective of the middleware. As the COME HERE and GO BACK gestures are given when the robot is not moving, this restriction does not pose any problems for these two gestures. The gesture recognition system also implements the STOP gesture. This gesture would immediately stop the robot at its current position. Unfortunately it is nearly impossible to give this gesture in hand tracking mode, as hand tracking first needs to be started by waving and afterwards the hand is lost because of the movement of the walker. In user tracking mode, the STOP gesture performs better, but did not meet expectations, because of the mentioned user re-detection. When the STOP gesture is given during such a re-detection, which always starts with the calibration phase, no gesture is recognized.

The recognition rate of 63% for the COME HERE gesture and 39% for the GO BACK gesture is low in comparison with other gesture recognition systems, especially as only simple gestures have been used. The difference is that gesture recognition systems are generally tested with persons that know how to perform the gestures and are physically capable of performing the needed movements. Additionally the test environment is optimized for best possible recognition. Our system was tested in nursing homes with all, except one candidate, older than 70 years. Some of the inhabitants were not capable of opening their hands completely or turning the palm towards the camera. Others needed several tries and further explanations to understand on how the gesture has to be performed. One person did not understand that a different gesture is needed for sending the robot back. This factors can be mentioned as reasons for lower recognition rates. Generally the gestures we implemented seemed to be well adapted for elderly persons because all participants were able to perform the gestures. Also most of the persons, for which the gesture recognition did not work, responded that they were physically able to perform the gestures.

Two gesture recognition systems have been tested. For the study in the nursing homes only the hand tracking was used. As the user tracker has difficulties to calibrate if a user is sitting in front of the camera, the hand tracking solution has to be used, if a person is not able to stand before the walker arrives. For the hand tracking it is no problem, if parts of a person are covered by obstacles. It has also the advantage, that the same gestures can be done with both hands without training them separately. The detection of the user position is not very accurate for hand tracking but this problem has been successfully solved by using the face position as target for the robot.

During the tests it turned out, that the training sets used were rather small. If the test persons did new variations of the gestures, the recognition often failed. If persons that trained the walker tried to give gestures, the recognition worked well. Therefore larger training sets would cover more variations of the gestures and help to recognize more different ways a gesture is performed.

8.2 Future Work

8.2.1 Stopping position

During the evaluation in the nursing homes, once the robot would have hit a test person because the position of the person was not measured well. It is essential that this never happens when the walker is used in a real scenario. Therefore the laser sensor of the robot should be used to have better precision when approaching the user.

8.2.2 Gesture recognition

As mentioned before, NiTE does not work well, when the camera is moving. For most of the applications this does not pose any problems. Nevertheless is it not possible to continuously track the users in front of the robot. Therefore, the target position can not be updated once the robot started moving. The robot is also not able to recognize additional gestures during its movement. If NiTE could be replaced by a tracking middleware that is able to compensate the movement of the camera, this would improve the gesture recognition in these situations.

Bibliography

- [1] W. Burgin, C. Pantofaru, and W. D. Smart. Using depth information to improve face detection. *HRI*, 2011.
- [2] Intel Cooperation. Threading Building Blocks FAQ. <http://www.threadingbuildingblocks.org/faq>. [Online; accessed 03-October-2014].
- [3] K. R. Konda, A. Königs, H. Schulz, and D. Schulz. Real time interaction with mobile robots using hand gestures. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, pages 177–178. ACM, 2012.
- [4] S. MacNamara and G. Lacey. A smart walker for the frail visually impaired. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 1354–1359. IEEE, 2000.
- [5] V. M. Monajjemi, J. Wawerla, R. Vaughan, and G. Mori. Hri in the sky: Creating and commanding teams of uavs with a vision-mediated gestural interface. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 617–623. IEEE, 2013.
- [6] OpenCV Website. About OpenCV. <http://opencv.org/about.html>. [Online; accessed 03-October-2014].
- [7] OpenCV Website. Local Binary Pattern Histograms Face Recognizer. http://docs.opencv.org/modules/contrib/doc/facerec/facerec_tutorial.html#local-binary-patterns-histograms. [Online; accessed 03-October-2014].
- [8] Z. Ren, J. Meng, J. Yuan, and Z. Zhang. Robust hand gesture recognition with kinect sensor. In *Proceedings of the 19th ACM international conference on Multimedia*, pages 759–760. ACM, 2011.
- [9] ROS Website. About ROS. <http://www.ros.org/about-ros/>. [Online; accessed 03-October-2014].
- [10] ROS Website. ROS Transformation Framework. <http://wiki.ros.org/tf>. [Online; accessed 03-October-2014].
- [11] A. Rusakov, J. Shin, and B. Meyer. Simple concurrency for robotics with the roboscoop framework. In *International Conference on Intelligent Robots and Systems, 2014 IEEE/RSJ. IROS*, 2014. [to appear].

- [12] J. Suarez and R. R. Murphy. Hand gesture recognition with depth images: A review. In *RO-MAN, 2012 IEEE*, pages 411–417. IEEE, 2012.
- [13] S. Theodoridis and K. Koutroumbas. *Pattern Recognition, 4th Ed*, volume 19. Academic Press, 2009.
- [14] M. Van den Bergh, D. Carton, R. De Nijs, N. Mitsou, C. Landsiedel, K. Kuehnlentz, D. Wollherr, L. Van Gool, and M. Buss. Real-time 3d hand gesture interaction with a robot for understanding directions from humans. In *RO-MAN, 2011 IEEE*, pages 357–362. IEEE, 2011.
- [15] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.
- [16] Wikipedia. Cosine similarity. http://en.wikipedia.org/wiki/Cosine_similarity. [Online; accessed 03-October-2014].
- [17] Wikipedia. Dynamic Time Warping. http://en.wikipedia.org/wiki/Dynamic_time_warping. [Online; accessed 03-October-2014].
- [18] Wikipedia. Euclidean Distance. http://en.wikipedia.org/wiki/Euclidean_distance. [Online; accessed 03-October-2014].
- [19] X. Zhao, X. Li, C. Pang, X. Zhu, and Q. Z. Sheng. Online human gesture recognition from motion data streams. In *Proceedings of the 21st ACM international conference on Multimedia*, pages 23–32. ACM, 2013.

A Appendix

A.1 User Documentation

A.1.1 Installing the requirements

The following software and libraries need to be installed in order to run the software of this master thesis:

- **ROS Hydro:** <http://www.ros.org>
- **OpenNI 2.2:** <http://www.openni.org> (Offline as OpenNI is discontinued, mirrors for downloading are available)
- **NiTE 2.2:** <http://www.openni.org> (Offline as OpenNI is discontinued, mirrors for downloading are available)
- **OpenCV 2.4.9** or newer, compiled with TBB support: <http://www.opencv.org>
- **Eiffel Studio 13.11:** <http://sourceforge.net/projects/eiffelstudio/>

Some of this software packages can be installed in Ubuntu with the Synaptic package manager. In Ubuntu version Precise Pangolin, the available OpenCV version had no TBB support. Therefore an installation from the sources was required with an additional argument to enable the TBB support.

A.1.2 Robot Control Application

Installation

The installation of the Robot Control Application is done with the following steps:

- Download the source code from the repository and copy it to your catkin workspace directory. The robot control application does not require the catkin build system, but `roboscoop_ros`, which is the communication part between Roboscoop and ROS, needs to be built.
- Run `catkin_make` in your workspace. This will generate all message files and the classes that are required for the communication between Roboscoop and ROS.

- Start Eiffel Studio with the command

```
ec -gui
```

and add "Roboscoop_app" as a new project. You will find the necessary "roboscoop_app.ecf" file in the folder "roboscoop/roboscoop_app".

- Finalize the code by clicking the down array next to the compile button in the toolbar of Eiffel Studio and select "Finalize".

Starting the application

To run the application, you can either manually execute the generated executable in the folder "roboscoop/roboscoop_app/EIFGENs/roboscoop_app/F_code" or you use Eiffel Studio to run the application by clicking to the down array next to run and select "Run Finalized System".

Using the graphical user interface

The graphical user interface provides a mode switch where you can choose between "autonomous mode" or "assisted mode". When changing the mode, the camera will turn as it has to face forward in assisted mode and face backwards in autonomous mode. The three buttons at the lower left corner of the GUI are used to select additional operation modes. The first button enables automatic position updates. If this function is enabled, the robot tries to update the position of the user, after the COME HERE command has been given. Furthermore there is a button to enable the attention check. If this function is turned on, the robot reacts only on users that look into the direction of the robot. The last button controls the user check. If this is enabled, only users whose faces are registered in the face database are allowed to control the robot. If the last two functions are used, it is important to retain a maximum distance between the robot and the user of about 3 meters and the lighting conditions have to be relatively good. Otherwise face detection will not work and the robot does not react on gestures any more.

A.1.3 SmartWalkerTracker

Installation

To install the SmartWalkerTracker, download the source code from the repository and copy the code to your catkin workspace directory. If this is done, just execute the `catkin_make` command, which builds the application. As a final step, you should extract all data from the archive `SmartWalkerTracker_Files.tar.gz` to the `.ros` directory in your home folder.

Starting the application

To start the application, two launch files are present. One variant simulates the odometry information of the robot to be able to test the application with using a robot. To start the application with simulated odometry, type:

```
roslaunch smart_walker_tracker2 smart_walker_tracker_with_simulated_odo.launch
```

If the SmartWalkerTracker should be started in normal mode, to be used together with the robot, type:

```
roslaunch smart_walker_tracker2 smart_walker_tracker.launch
```

A separate roscore can be started before executing the launch files. If no roscore is running, the launch files will start it automatically.

Training the gestures

To train new gestures, you can use the "GestureTeacher" application. The following steps have to be performed: First check if a roscore is running. If this is not the case, start a new console and run "roscore". Open another console and change to the .ros directory in your home folder. Here you should backup the files user_gestures.csv and hand_gestures.csv. Then you can start the "gestureTeacher" by typing:

```
smart_walker_tracker2 gesture_teacher2
```

After starting the application, you will be asked which gesture you want to train. Enter the id of the gesture you want to teach (currently the id's are: 1: stop, 2: come here and 3: go back). Then you can choose if you want to train a hand gesture or a user gesture. Type 'h' for hand gesture and 'u' for a user gesture. Then you will have to enter how many times you want to repeat the gesture recording. As a last option, you can choose how many frames you want to record. As the gesture recognition operates at a frame rate of 15 frames per second, 30 frames correspond to 2 seconds recording time, which should be enough for most gestures.

For the hand gestures, a post-processing step has been added, to cut off the part of the recorded gesture, which is essentially a neutral pose and does not belong to the actual gesture. Hand gesture recognition works also without this post-processing step, but post-processing avoids the required waiting period before performing the gesture. Furthermore the gestures are detected faster, as there is no non-gesture part after the real gesture. For doing the post-processing on the hand gesture data, open a console and switch to the .ros folder of your home directory. Here type:

```
roslaunch smart_walker_tracker2 data_preprocessor
```

After the gestures have been trained to the SmartWalkerTracker, it might be necessary to adapt the error thresholds for the different gestures. The threshold settings can be changed in the file `smart_walker_tracker_constants.h`. There are two constants `MAX_ALLOWED_ERROR_USER` and `MAX_ALLOWED_ERROR_HAND` for the threshold settings of the user and the hand tracker respectively. The value at position 1 is for the STOP gesture, position 2 for the COME HERE and position 3 for the GO BACK gesture.

A.1.4 Face detection

Installation

For installing the face detector, download the source code from the SVN repository. Then copy all files into your catkin workspace and run `catkin_make`. The face detector needs an image directory in the `.ros` folder, which is located in your home directory. Create this directory by changing to the `/.ros` directory and typing `mkdir images`. Then run the `face_teacher` to store new faces.

Starting the application

A launch file has been created to facilitate the start-up of the face tracker. To start the face tracking, first check that `roscore` and the SmartWalkerTracker are already running in separate console windows. The second mentioned is required because face detection uses the raw data of the camera, which is published in ROS by the SmartWalkerTracker. Then launch the Face Tracker by calling

```
roslaunch smart_walker_tracker2 face_tracking.launch
```

Training new faces to the face recognition

For adding new faces to the face recognition, an application has been created for this purpose. The following steps show how a new face can be trained to the face recognition: Check if `roscore` is running. If this is not the case, open a new console and type `roscore`. Open a second console and change to the `.ros` directory in your home folder. Then start the face teacher application by calling

```
roslaunch smart_walker_tracker2 face_teacher
```

First you will have to enter the id of the user you want to teach. If you want to add more images of an existing user, you can enter his or her id here. If you want to add a new user, use the next unused id. For training new faces, it is important that no other faces except the one you want to train are visible in the camera image. You will then see a camera image with a bounding box around the detected face. If you want to record an image of this face in the current pose, you can press the `'c'` key, which then stores the image. It is recommended to store several images for every user of the system. Therefore

move your head a little bit and press the 'c' key about 10 to 15 times per user. To leave the application, press the 'q' key. After recording new faces you should check the recordings. You will find the stored faces in the `./ros/images` folder. The first number of the images file name is the id of the user, whereas the second number is a unique id for every recorded image. If you found images that are unsatisfactory, you can delete them and also remove the line, corresponding to the deleted image file name, from the `trained_user.csv` file, which is also located in the `./ros` folder.

A.2 Developers Guide

A.2.1 SmartWalkerTracker

The SmartWalkerTracker is the core application of this thesis. It is responsible for the gesture recognition and the publication of position data in TF and via messages in ROS. The SmartWalkerTracker consists of several parts which are explained in more detail in the following sections. For the explanation I try to adhere to the order of the data that is processed in the application.

After starting the SmartWalkerTracker, a connected 3D sensing device connected to the computer is searched. Only one sensor is required and supported. If several sensor would be connected to the same computer, it is not defined which one would be used. After opening the device, the video modes that the camera should use, need to be set. The PrimeSense Carmine sensors support the following video modes:

Depth image:

- 160 x 120, 30 fps
- 320 x 240, 30 fps
- 320 x 240, 60 fps
- 640 x 480, 30 fps

RGB image:

- 320 x 240, 30 fps
- 320 x 240, 60 fps
- 640 x 480, 30 fps
- 1280 x 720, 30 fps
- 1280 x 1024, 30 fps

For the depth resolution we had to use 320 x 240 pixels with a frame rate of 30 frames per second. For larger distances to the camera it would have been useful to have a higher depth resolution, but the NiTE middleware was not working with higher resolutions

even though nothing about this restriction was found in the documentation. For the RGB resolution we decided to use the highest resolution that is supported by the camera which is 1280 x 1024 with a frame rate of 30 frames per second. This resolution has been chosen to have better performance in face detection and face recognition (see chapter 5.1).

A.2.2 Camera publisher

As part of the SmartWalkerTracker a separate thread is used to publish the camera raw data in ROS. This is done in the SmartWalkerTracker because NiTE requires direct access to the device object that can only be created once in OpenNI as it gets exclusive access to the USB camera. The frame rate of the raw data publication is configurable in the configuration file. Currently only 2 frames per second are published, as the published raw data in ROS is only used by the face detection and for this purpose this rate is sufficient. If other nodes or image viewers would require higher publication rates, this setting can be set up to the maximum frames per seconds that is supported by the chosen video mode.

The raw data publication itself loops with the chosen frame rate and gathers images from the depth and/or RGB video streams as chosen by the user. This images are published in ROS as messages of the type `sensor_msgs/Image`. In order to use the published camera data in other applications, a `camera_info` topic is required as well. The camera info topic contains information about the current video frame that is published, such as the frame id, the used resolution, camera calibration data and so on.

A.2.3 User Tacking

The user tracker is one possibility for commanding the SmartWalker by gestures. The user tracker uses full body tracking for gesture recognition. It is split into several parts which will be explained in the order of the processing pipeline:

Tracking the joints of the user

As a first step, the user tracker has to publish the position and the orientation of the joints of a user. 15 joints explained in section 2.1.2 are tracked. The tracking itself is done by the NiTE middleware of PrimeSense, which returns the tracked joints for every user as `UserData` object. The data of this object is used to know which users are currently tracked and to publish its position and orientation in the transformation framework (TF). Every tracked user gets an ID and a state assigned by NiTE. The id is a unique identifier for this user as long as it is tracked. If a user disappears, the id will be reused. NiTE uses different tracking states: If a user appears in the scene, his state is "new" which means that this is the first frame in which this user appears. As long as the user is visible by the camera, his state is "visible". If the user leaves the range of the camera, his state is "lost". If a user is visible, the state of the skeleton tracking can be checked. First the user is in "calibrating" state in which the user should move around to allow NiTE to identify the body joints. In older NiTE versions, the user had to perform a PSI pose for calibration which is not required any more. If calibration is done, the skeleton state changes to "tracking" which

is the phase in which data of the joint positions and orientations is returned. If skeleton state switches to "none", the tracking stopped. There are more states that signal problems during calibration, which are not explained in detail.

An important point for the user tracker is the calibration phase. During this phase the user should move a little bit in order to allow NiTE the identification of the joints. During this phase the more of the body is visible, the better the calibration can be done. If this is not the case, the calibration can take very long or is not possible at all. If a person is sitting, calibration can become a very difficult step, as the joints are not well visible to the camera and not enough movement can be done for calibration.

For the user tracking, code has been used from the `openni2_tracker`, an open source ros package of the `ros-drivers` git repository (https://github.com/ros-drivers/openni2_tracker). The code for managing the tracking states of the users as well as the publication of the skeleton positions in TF are adaptations of the functions of the `openni2_tracker`.

Determining the goal position

If the user commands to robot to come, a goal position for this command is required. Therefore the user tracker publishes a message with the current user position once every second, which corresponds to one message per 30 camera frames. For this purpose a new ROS message type has been introduced for the communication between the SmartWalkerTracker and the robot control application.

Figure A.1 shows the content of a `RS_UserPosition` message. `x`, `y` and `z` are the coordinates of the goal position, which is 40 cm in front of the user. Of course, the stopping distance to the user can be configured. As the robot moves on the ground, the `z` coordinate is not read when sending the robot to the user but this information can be useful when for example the position of a face is required. `theta` is the orientation of the user. It is published by the SmartWalkerTracker but Roboscoop currently does not support an orientation at the goal position of the robot. The robot just stops when the goal is reached. The `userId` is the id given in the `UserData` object. The `recognizedId` is not used in the user tracker. The `trackingMethod` contains an id that shows if the user position was detected by the user tracker, the hand tracker or the face detection to allow prioritisation of the incoming position data in the robot control application. Additionally to the message that is sent to ROS, the goal position of the robot will also be published in TF.

```
1 time stamp
2 float64 x
3 float64 y
4 float64 z
5 float64 theta
6 int8 userId
7 int8 recognizedId
8 int8 trackingMethod
```

Figure A.1: The `RS_UserPosition` message

User gesture recognition

As soon as the joints positions and the goal position of the robot are determined, the gestures given by a user can be detected. This process is described step by step:

Creating features: In a first step, feature vectors have to be created. This is done by calculating the pair-wise distances between all the joints above hip. These are the joints that are moved for the gesture we used in our gesture recognition system and these joints should be always visible by the camera, also if the user is close to the camera or an object is in between the walker and the commanding person.

Building a camera input buffer: To be able to recognize a gesture as a series of camera frames, the camera data needs to be buffered from gesture start until the gesture has been performed. As it is not known in advance when a gesture has started and when the gesture is performed, a fixed window size of 50 frames has been chosen. All recorded gesture templates are scaled to a size of 25 frames. With 50 stored camera frames, it is possible to execute a gesture 2 times slower than the template.

Comparing the input buffer to the recorded gesture templates: After buffering the frames and reading the gesture templates into the memory, classification of the gesture can be performed. For the classification of the gestures, a dynamic time warping approach is applied. The implementation in SmartWalkerTracker is a improved C++ implementation of the pseudo code of DTW in Wikipedia [17]. To compare the feature vectors, the euclidean distance is computed and weighted with the variance of the features. Like this, features with more variance account more to the calculated cost. After applying DTW, a total cost in the sense of a sum of the calculated euclidean distances is returned. After comparing the input buffer with the stored gesture templates, a cost is calculated for every template. This costs are sorted and the gesture is predicted using a k-NN classifier. The gesture with the most matches using the lowest 3 error values, is taken.

Publish the recognized gestures: After a gesture has been recognized, it needs to be published in ROS, so that the Eiffel robot control application can process the command assigned to this gesture. For publishing gesture messages in ROS, a new message type shown in figure A.2 has been created:

Stamp holds the timestamp when the gesture was recognized by the SmartWalkerTracker. The field "userGesture" contains the id of the gesture, "userId" holds the id of the user that performed the gesture and "trackingMode" contains an id that tells the recipient of this message, that this gesture has been recognized by the user tracker. After creating the message, it is published in ROS in the topic /SmartWalkerTracker/user_gesture.

```
1 time stamp
2 int8 userGesture
3 int8 userId
4 int8 trackingMethod
```

Figure A.2: The RS_UserGesture message

Applying DTW to user tracking

DTW has been explained in section 4.1.2. DTW is a general approach and can be used for all kinds temporal data series. In user tracking, the data item to compare is a feature vector which contains distances and angular information. The knowledge of the kind of the data under comparison is important for the selection of an appropriate cost function. For this type of data either a cosine similarity could be good (because of the angular information) or an euclidean distance could be a good choice (because of the stored distances). It is also imaginable to use combinations of the two (e.g for parts of the feature vector) or adaptations of them. One adaptation would be to use a modified Mahalanobis distance, which weights data items more, if their confidence is higher. This approach seemed promising to us, as sometimes parts of the body are occluded and therefore not actually tracked but only inferred by NiTE. During testing we found out that NiTE returns a confidence value for the tracked joints, but these values were binary. Either a joint is tracked or it is inferred. We then tried to weight the joints less when they are inferred. Informal tests showed no improvement at all. Sometimes it was even better not to use the weighting, as NiTE infers the joint positions quite well and if a big part of the body is not visible there is not much data present any more. As only very few angular data items are present in the feature vectors, euclidean distance provides the best error measurement. Since the confidence values of NiTE seemed not very helpful to get better error estimates, we tried statistics over the used features. First of all the mean value for every feature has been calculated. This allows the normalization of the error values to have an equal weight of the feature vector items. Furthermore the variance for every feature vector item has been calculated. This allows to weight items with higher variance more than the ones with lower variance. During another informal test we got better error values than before. The term "better error values" is very imprecise, because it is very difficult to say if it is better to have higher or smaller errors. Generally it does not matter if the error values are very small numbers or larger values. They should be adapted to the data type that is used. As the errors will be used to say if the currently performed gesture is the same as a recorded gesture template out of the database, the error should be as much distinctive as possible. The error should be as low as possible, if a performed gesture is from the same gesture class as a recorded gesture. On the other hand, the error should be as high as possible, if the recorded gesture is of a different class.

When training the gesture recognition with more people, we found out, that different persons have quite large variations of how the gestures are performed. Therefore it seems to be the case that some gestures, even if they are in the same gesture class, lead to an increase in the variance of some items of the feature vectors. Unfortunately this increase is only desired, if it also leads to a better distinction between the gesture classes. This was not always the case as some features varied very much between the recorded instances of the same gesture. Therefore another variation of an error measurement has been developed. We call them "inter gesture variances". This is a calculation of the variance values for every feature in which, during the calculation of the variance, a mean per gesture has been used. Therefore this variance measures the deviation of this feature from the mean value of this gesture class. In chapter 6 you can find an evaluation of this error estimate.

Teaching the user gesture recognition

Before gesture recognition can be used, gestures must be trained to the gesture recognition system to be used as gesture templates. For this purpose an application called GestureTeacher has been created. This application is a simple console application that asks the user to enter the id of the gesture he wants to train, the number of frames that should be recorded and how many times the chosen gesture should be recorded. After that, the user can stand in front of the camera and wait for NiTE to start tracking the body joints. As soon as body tracking data is available, a countdown of 5 seconds starts, before the user can perform the gesture. The desired number of frames are then recorded at a speed of 15 frames per second (which is also the frame rate during gesture recognition). After the gesture has been performed, the countdown comes up again until the requested number of recordings have been made.

A.2.4 Hand Tracking

Besides the user tracker, hand tracking has been implemented. In contrast to user tracking the hand tracker requires only the hand to be visible by the camera.

Tracking the hand of the user

For user tracking several body joints had to be tracked. For hand tracking only the center position of the hand is required. For tracking the hand, the hand tracker of NiTE is used. In order to start the tracking process, the approximate hand position has to be known. This can either be done by getting the hand position from the user tracker, which makes no sense in this case, as either hand tracking or user tracking is used, or the hand position can be determined by a build-in gesture. The NiTE middleware supports three different gestures: Wave, click and arm-raise. For the hand tracker, the wave gesture has been used, which requires the user to wave the hand he wants to track. As soon as the wave gesture has been recognized, the hand tracking starts. After starting the hand tracking, a unique continuous id is given to this hand and a HandData object is created. In this object the current state of the hand tracker can be read as well as the coordinates of the hand center. The tracked hand position is published to TF.

Determining the goal position

When hand tracking is used, the goal position of the robot has to be calculated either by face tracking (see section 5.1.1) or if there is no face detected, by the current hand position. Face tracking is preferred over the hand position because gestures can be given at various positions. If the hand position has to be used, the robot will stop 40 cm away from the hand position at the time, the gesture was recognized. Also for hand tracking, the goal position is published as a RS_UserPosition message to ROS and to TF.

Hand gesture recognition

The hand gesture recognition process works as follows: A feature vector for the hand tracker consists of the 3D displacement of the hand from a gesture starting point. As for the user tracker, we also want to analyse the features used in hand tracking, if they are viewpoint, anthropometry, execution rate and personal style invariant: The gestures are clearly viewpoint invariant, as the hand tracker of NiTE returns the 3D positions of the hand centers and compensates for different camera viewpoints. It is not entirely clear, if arbitrary gestures with the hand tracker would be anthropometry invariant. If a gesture would be recorded with a long arm movement, it would be impossible for a person with smaller arms to redo it. The gestures we specified should be antropometry invariant as we require very small movements which are possible for every healthy person. With a large training set, differences in movements and different lengths of the gesture are covered and also personal style variations are possible. To ensure execution rate variations, the same DTW scheme as for user tracking has been used for hand tracking as well. The hand gestures in our training set contain 18 frames on average. Which means, that the recorded gestures require a bit more than a second to be executed. As DTW has an allowed shift of 35 frames, a timeshift of a bit more than 2 seconds is allowed.

The gesture starting point is determined at the position where the hand tracker has been started trough the initial wave gesture. This starting point is updated, if the user does not move the hand for 20 camera frames. During this 20 frames, the distance travelled with the hand between two consecutive frames must be smaller or equal to 0.1 mm. With this settings, a person with a trembling hand can still update the gesture starting position. A gesture consists of a time series of feature vectors. These are then compared to the stored gesture templates using a k-NN classifier which considers the 7 lowest errors under the euclidean distance. The output of the classifier is only published to ROS if the final error is below a threshold, which can be set for every gesture class separately. This allows to boost more difficult gestures, which are often done with a higher error than others. For publishing gestures to ROS, it is important to publish gestures only once when they are recognized. To achieve this, the gesture recognition system has several states and a publication to ROS is only possible upon a state change. There are states for every supported gesture and an additional state called `NEUTRAL_STATE`. This state is used when currently no gesture is recognized, i.e the calculated error under the euclidean distance is higher than the threshold value. A parameter sets how many detections in a series are required for a gesture to be published. At the moment 5 consecutive detections are required for a state change. This restriction is no problem for normal gesture detections as a lot of detections occur, but it successfully removes flickering between `NEUTRAL_STATE` and the state of a gesture at the beginning and the end of a detection. The state flickering would lead to a lot of publications of this gesture, even though only a single gesture has been performed by the user.

Teaching the hand gesture recognition

As for the UserTracker, also the HandTracker needs training for creating gesture templates. For this purpose the same application called GestureTeacher can be used. The process is exactly the same as for user tracking except that the hand recognition work differently. After all settings of the GestureTeacher have been made, the recording can be started by placing the hand in front of the camera and wave twice from left to right and back or the other way round. After that, the countdown starts and the gestures can be performed. For hand gestures it is important to remain the hand still during the countdown to allow the hand tracker to find the starting position of the gesture.

A.3 Original questions of the nursing home study

Part 1

1. Geschlecht:
 Männlich Weiblich
2. Wie alt sind Sie?:
 unter 70 70 – 79 80 – 89 über 90
3. Benutzen Sie ?:
 nichts Gehstock Gehbock Rollator Rollstuhl
4. Wenn ja, wie lange haben Sie ? benutzt?:
 weniger als 1 Jahr 1 – 2 Jahre 3 – 5 Jahre mehr als 5 Jahre
5. Wie oft benutzen Sie ? ?:
 nie selten manchmal oft immer
6. Wie oft benutzen Sie ? ausserhalb von Gebäuden?:
 nie selten manchmal oft immer
7. Wie oft benutzen Sie ein Smartphone, ein Computer, etc.?:
 nie selten manchmal oft immer

Part 2

8. Gefällt Ihnen der Rollator?:
 ja sehr eher ja neutral eher nicht überhaupt nicht
9. Wie finden Sie die Grösse?:
 zu gross ein bisschen zu gross perfekt ein bisschen zu klein zu klein
10. Wie finden Sie das Gewicht?:
 zu schwer ein bisschen zu schwer perfekt ein bisschen zu leicht zu leicht

11. Wie angenehm ist es, mit dem Rollator zu laufen?:
 sehr angenehm angenehm neutral unangenehm sehr unangenehm

12. Wie einfach ist es, den Rollator einzuparken?:
 sehr schwierig schwierig neutral einfach sehr einfach

Part 3

13. Gefällt Ihnen der Rollator?:
 ja sehr eher ja neutral eher nicht überhaupt nicht

14. Ist es nützlich, dass der Rollator zu Ihnen kommen oder fortgehen kann?:
 ja sehr eher ja neutral eher nicht überhaupt nicht

15. Wie angenehm war es für Sie, als der Rollator zu Ihnen gekommen ist?:
 sehr angenehm angenehm neutral unangenehm sehr unangenehm

16. War die Anhalteposition für Sie angenehm und konnten Sie den SmartWalker gut erreichen?:
 sehr angenehm angenehm neutral unangenehm sehr unangenehm

17. War es für Sie schwierig zu verstehen, wie Sie den Roboter bedienen können?:
 sehr schwierig schwierig neutral einfach sehr einfach

18. Waren die nötigen Handzeichen für Sie schwierig auszuführen?:
 sehr schwierig schwierig neutral einfach sehr einfach

19. Hat der Roboter Ihre Handzeichen gut verstanden?:
 ja immer meistens neutral manchmal nicht überhaupt nicht

20. Hätten Sie sich mehr Rückmeldung vom Roboter gewünscht (z.B. indem der Roboter mehr spricht oder Anzeigeleuchten hätte)?:
 ja sehr eher ja neutral eher nicht überhaupt nicht

21. Wie finden Sie einen Robotikrollator?:
 sehr spannend spannend neutral gefährlich sehr gefährlich

22. Würden Sie unseren Rollator verwenden, oder würden Sie lieber bei Ihrem gewohnten Rollator bleiben?:
 ja eher ja neutral eher nicht sicher nicht

23. Gibt es weitere Funktionen, die der Rollator haben sollte?