NTNU
Norwegian University of
Science and Technology

Using the Kinect Sensor for Social Robotics

Sigurd Mørkved Albrektsen

Master of Science in Engineering Cybernetics

Submission date:	June 2011
Supervisor:	Geir Mathisen, ITK
Co-supervisor:	Sigurd Aksnes Fjerdingen, SINTEF

Norwegian University of Science and Technology
Department of Engineering Cybernetics

NTNU – Trondheim
Norwegian University of
Science and Technology

Master thesis

Using the Kinect Sensor for Social Robotics

Author:
Sigurd Mørkved Albrektsen

Supervisors:
Geir Mathisen
Sigurd Aksnes Fjerdingen
Øystein Skotheim

June 13, 2011

Abstract

This thesis presents an innovative approach to social robotics through gesture recognition. The focus is on recognizing gestures because this is an important aspect regarding interpretation of a person's intent when he or she gives commands to a robot.

The equipment used is a Kinect sensor, developed by Microsoft, attached to a moving platform. The Kinect communicates with software running on a PC through the OpenNI interface and uses the NITE middleware by PrimeSense.

The results of this thesis are:

- a broad literature study presenting the state of the art of gesture recognition
- a system which handles the problems that arise when the Kinect is non-stationary
- a gesture recognizer that observes and analyzes human actions

There are mainly two problems that are solved by the implemented system. First, user labels might be incorrectly swapped when the Kinect's standard algorithm loses track of a user for a few frames. Second, false-positive users are detected, as the Kinect is assumed stationary. Because of this, everything that moves relative to the Kinect is marked as a user. The first problem is counteracted by mapping the observed label to where it was last seen. The second problem is solved using a combination of optical flow and feature analysis.

The gesture recognizer has been developed to allow robust and efficient segmentation, joint detection and gesture recognition. To achieve both high efficiency and good results, these algorithms are tailored to be used with the high quality user silhouettes detected by the Kinect. In addition, the default Kinect algorithm needs some time to initialize when a new human user is detected. The implemented gesture recognizer has no such delay.

Problem Description

The Kinect sensor from Microsoft has recently given the opportunity to interface with machines in a more natural way than ever before. This may reveal new possibilities for creating social robots – robots that interact and coexist with humans. Such robots include e.g. mobile service robots in home or office environments and robot manipulators lending an extra hand in work environments. This assignment will focus on how the Kinect sensor best is able to assist a social robot when interacting with humans. The assignment is held in conjunction with an ongoing SINTEF project (Next Generation Robotics for Norwegian Industry), where SINTEF is interested in investigating new methods for communicating and interacting with robots.

1. Perform a literature survey on social robotics, focusing on
    a) Sensors for visual perception.
    b) How human interaction is handled.
2. Give a practical analysis of the performance of the Kinect sensor. The analysis should at least include accuracy, update frequency, and an overview of good and bad environmental conditions for the sensor.
3. Design a set of algorithms which allows reliable user detection from a non-stationary platform.
4. Design a set of algorithms demonstrating use of the Kinect sensor in a social robotics setting using a mobile robot driving in an office environment. The robot should be able to recognize a human while moving and detecting defined gestures telling the robot to come or go away.

5. Implement and analyse the algorithms in simulation.
6. If time allows, implement the algorithms on a physical robot available at SINTEF.

Acknowledgements

First of all I would like to thank Geir Mathisen for accepting me as a student and supervising this thesis. Second, I would like to thank my co-supervisors Sigurd Aksnes Fjerdingen and Øystein Skotheim for supporting me and advising me regarding the problem description, in addition to their review of parts of my thesis.

Finally, I would like to thank Tonje Gauslaa Sivertzen for her invaluable support and reviewing. Thank you very much for using your precious time in your exam period on my thesis.

Contents

Terms and Abbreviations

C#	A programming language developed by Microsoft available for Windows through the .Net package and Linux through the Mono project.
Emgu CV	A cross platform .Net wrapper to the Intel OpenCV image processing library [7].
Fps	Frames per second
GUI	Graphical User Interface
Middleware	"In a distributed computing system, middleware is defined as the software layer that lies between the operating system and the applications on each site[sic] of the system." [16]
OpenCV	**Open** Source **C**omputer **V**ision. An open source library of algorithms for use with real-time image data.
OpenNI	**Open** Source **N**atural **I**nteraction. An open source library which provides communication between low level devices and high level middleware.
px	From the word "pixel" which is an abbreviation for "picture element". A pixel is the smallest unit of a picture which can be represented or controlled.
Polling	Polling is the process of repeatedly checking if a variable has changed, also known as "busy waiting".

Segmentation and labeling	Segmentation is, in the context of image analysis, the procedure of splitting specific parts of an image into regions. Labeling is the process of identifying what the region represents. In this thesis segmentation and labeling are done at the same time, thus both terms are used for the same procedure.
ROI	**R**egion **O**f **I**nterest. When using images, it is not always necessary to calculate features for the whole image. If what you are looking for is in a defined area of the image, the ROI can be set so that only this part is processed.
Wrapper	A wrapper in programming language context is a translation between two languages. This makes features written in a specific language available in another and often helps to make the features available in a way which is natural for the target programming language.

1. Introduction

The discipline of controlling mobile robots has until recently consisted of computing paths for the robot to follow, with or without object collision avoidance. Robots are traditionally controlled using a special interface, such as an operative panel or a computer. To change a moving robot's objective will in most cases consist of stopping the robot and then reprogram it using an on-board interface or a computer.

However, one might imagine a scenario where a robot interacts with humans operating in a more fluent way. This is the idea behind social robotics. What if the robot would not only detect you as an object it has to avoid, but as a human it has to obey? What if you could tell the robot to perform a task, only using your own body?

This thesis brings that scenario one step closer.

The main aspect of social robotics boils down to solving one problem: for robots to understand human commands. Although humans may communicate using speech alone, gestures such as pointing or signaling actions are frequently used in daily life, especially when explaining actions. Hence, gesture recognition is an important task to master for robots and humans to coexist in the same environment.

November 4th, 2010 the Kinect sensor for XBox 360 was launched by Microsoft in North America. According to the retailer Play.com [29], the Kinect allows "Full-body play":

> Kinect provides a new way to play where you use all parts of your body - head, hands, feet and torso. With controller-free gaming you don't just control the superhero, you are the superhero. Full-body tracking allows the Kinect sensor to cap-

> ture every move, from head to toe, to give players a full-body gaming experience

However, as developers discovered the Kinect's potential beyond use in games, efforts were made to be able to connect it to a PC and use the sensor's depth camera. Drivers were released after short time due to effort made by open source communities, and high quality closed source middleware modules were released soon after that. The drivers allow robust tracking of human users when the Kinect is stationary. However, as will be shown in this report, problems arise when the Kinect moves.

This thesis aims to conquer these problems, and thus to allow the Kinect to be used on a mobile platform in a social robotics context.

1.1. Contributions

This thesis has four major contributions:

- An innovative, customizable approach to identify human silhouettes - User Detector Filter.
- A "segmentor" which partitions a human silhouette into different body parts.
- A body joint detector which detects certain joints of a segmented human body.
- A robust gesture detector, which uses approximations of the positions of a user's shoulders, elbows and hands to detect dynamic gestures.

One of the two main problems with using the Kinect on a moving platform is that when it is non-stationary, many false-positive (non-human) users are detected. This thesis presents a system for filtering out these false-positives, which is implemented with focus on extendability and efficiency. The user detector filter consists of three components: a data handler, filters and a voter. The data handler buffers and stores data, the filters apply different criteria which specify if a detected user is human or not, and the voter combines the output from the filters.

Furthermore, an efficient segmentation algorithm which is optimized for use on the high-quality silhouettes from the Kinect interface is implemented. This algorithm labels the legs (if they are visible), head, torso and arms of a user with low computational complexity.

In addition, a joint detector, which approximates the shoulders, elbows and hands of a user, was created. Effort has been made to make the joint detector robust and reliable. The joint detector focuses on making accurate estimations of the vertical position of the shoulder joint, the horizontal position of the elbow and an accurate position of the hand.

A robust gesture detector has also been made. This consists of a finite state machine (FSM) with four states per arm: *Undetected*, *ArmStraight*, *ArmRaised* and *ArmTowardsHead*. As this needs very little initialization time per user, the overhead for detecting new users is very low, compared to the Kinect's library.

1.2. Report Overview

This thesis starts with presenting the state of the art of devices and algorithms used for gesture recognition in Chapter 2. The chapter focuses on a range of equipment that could be used for detection of both static and dynamic gestures. As already mentioned, the Kinect depth sensor from Microsoft was selected for this thesis. This will be described in Chapter 3, with focus on hardware specifications, the detection algorithm which is already implemented by PrimeSense and limitations of the Kinect sensor.

As the Kinect's algorithms assume that it is stationary, a system has been made to improve performance when placed on a moving platform. An overview of the system is presented in Chapter 4, and a more detailed description of how the most important parts of this system works is provided in Chapter 5 and Chapter 6.

The system's general behaviour, in addition to some special cases, is shown in Chapter 7 where each of the interesting parts are presented in its own section. A discussion of the implementation and behaviour of the system, in addition to limitations, is presented in Chapter 8. Chapter 9 summarizes the thesis and suggests further work.

2. State of the Art

Gesture recognition is the process of interpreting motions or signs that a user performs. There are several approaches of detecting a gesture, which differ both in the equipment used and how the information is processed. This chapter gives an overview of existing technologies with respect to both equipment and gesture analysis.

Gestures can be divided into two main categories, static and dynamic gestures. Examples of static gestures are holding up the index finger, indicating the number one, holding up the index and middle finger, indicating the number two, or showing the palm of your hand, indicating a stop signal. Examples of dynamic gestures could be nodding or shaking your head, indicating yes or no, or waving your hand to gain attention. Some dynamic gestures can be thought of as moving static gestures.

Several approaches that focus on recognizing hand gestures use signs from the American Sign Language (ASL). ASL is a visual language which deaf people use to communicate. Recognizing ASL signs by just observing the hands of the user is rather difficult. One of the reasons for this is that whole of the body is used when communicating in ASL, as National Association of the Deaf [23] states:

> The shape, placement, and movement of the hands, as well as facial expressions and body movements, all play important parts in conveying information.

2.1. Gesture Recognition Equipment

To recognize gestures, the first step is to obtain information about the object that performs the gesture. This object could for example be the head of a human being, the hands, the arms or the whole body. To perform this task, a diversity of different sensor systems can be used. These systems could be divided into two main categories: systems that use close proximity sensors, where one or more devices are attached to or held by the user, and systems that only perform measurements from a distance.

2.1.1. Close Proximity Sensors

Close proximity sensors often provide high quality and accurate information. However, they tend to impose additional time to prepare for usage, and may restraint natural behaviour while using the system. Natural behaviour could be restrained as these devices often have wires attached to them, and even holding an object may change the way a person moves. In addition, this sort of equipment is usually rather expensive as it is produced to perform very specific tasks.

Data Gloves

There exists a variety of data gloves [22, 32, 38, 39] which provide real time information about a hand's current configuration. These data gloves consist of a glove covered with sensors, typically at the joints of each finger. Some of these systems also provide information about the placement and orientation of the hand, and some rely on a supporting system for this kind of information.

As finger joints' angles are measured directly, extraction of measurements requires low software complexity and calculation imposes small overhead. In addition the measurements are generally of high quality and the measurement frequency is high.

Data gloves, however, are generally quite expensive with the P5 glove as a notable exception [22]. Furthermore, as these sort of devices typically are connected to a computer by cables they might be cumbersome to put on and, more importantly, may hinder natural movement [21].

If full body gestures are necessary, a data suit [11] could be used. This is an extension of data gloves, which provides measurements of multiple limbs at once. As a result, information about the whole body configuration is measured and, as Goto and Yamasaki [11] state:

> A performer wears this suit, but doesn't hold a controller [. . .] in his hands. Therefore, [. . .] his gesture could be liberated to become a larger gesture, like a mime.

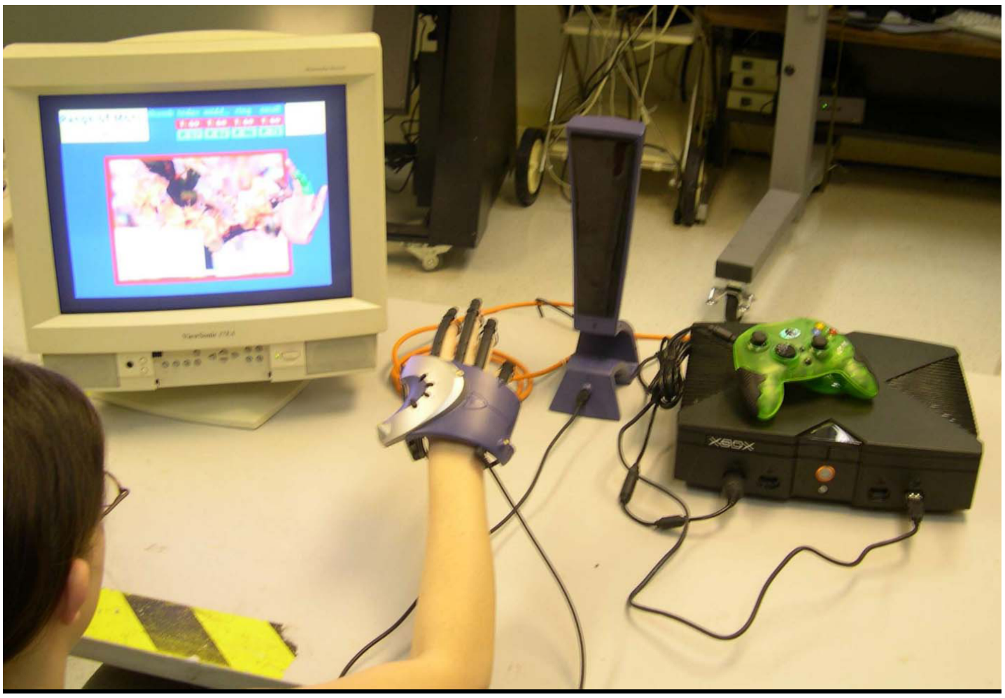


Figure 2.1.: The P5 data glove is a low-cost data glove with a 3D-positioning sensor. Image from [22].

Accelerometers

In comparison to the data gloves, accelerometers provide a different principle of gathering information for gesture recognition. Instead of measuring angles, accelerometers measure acceleration caused by a user's movement and Earth's gravity. A common way of using accelerometers for gesture recognition is to "train" a system to recognize how a gesture is performed by repeating it multiple times and storing the information produced. When the gesture is to be recognized, the action performed is matched with the database of measurements and the best match is chosen.

A specific example of this is Huang and Fu [13] who present a method which uses the "Wii Remote" produced by Nintendo. This device is less intrusive than most data gloves as it is wireless and does not take time to prepare for usage. In addition it is lightweight and will therefore not interfere with how one would perform the gesture without the device. Measurements are performed inside time windows which are intervals defined by a starting point and an ending point. The starting point and ending point of these windows depend on two parameters, the magnitude of the accelerometer's output and the sign of the acceleration's time derivative. Measurements inside each time slice are normalized, and each time slice is assumed to contain a single gesture.

Accelerometers are frequently used as a supplement to other sensors. This is because they are a very useful tool to find orientation in space as they can measure Earth's gravitational pull. This is especially useful when recognizing static gestures, as for example pointing upwards and pointing downwards may have very different meaning.

2.1.2. Vision Sensors

When systems can not rely on attached or held sensors, information about the user must necessarily be obtained from a sensor at some distance from the user. This approach gives users freedom to move naturally, as they are not bound by cables or inhibited by potentially heavy equipment.

Passive Monocular Cameras

A common vision sensor which often is used in gesture recognition is the passive monocular camera. A passive camera is, in contrast to an active camera, a camera which does not emit any light, but only responds the light which is provided by the environment. This sensor is often used as it is both inexpensive and highly available.

Passive cameras are often of very high resolution, typical consumer class cameras have a maximal resolution from 640x480 pixels to 1920x 1080 pixels, with an update rate of 30 frames per second (fps) [18, 19].

The main problem when using passive cameras, as with most vision sensors, is to recognize the pose of the user. As Huang and Pavlovic [14] state:

> The human hand as a geometric shape is a highly non-convex volume. Trying to detect the hand configuration from camera images is therefore a difficult, if not an impossible, task.

Due to this problem, many systems that are based on passive vision use some sort of markers. A marker is a device which is easily recognizable with simple imaging techniques. Davis and Shah [8] suggest using a glove with marked fingertips, and then performing a simple histogram analysis to remove all data which is not of interest. This approach is shown in Figure 2.2.

Other common approaches involve direct color segmentation without the use of markers. For example is the use of skin color in HSV or YUV color space rather common [44]. However, as stated by Zabulis et al. [44]:

> The perceived color of human skin varies greatly across human races or even between individuals of the same race. Additional variability may be introduced due to changing illumination conditions and/or camera characteristics.

Oka et al. [24] solve this problem in an interesting way. Instead of using a camera which detects information in the electromagnetic spectrum's visible region, it detects information in the infrared region. As warm

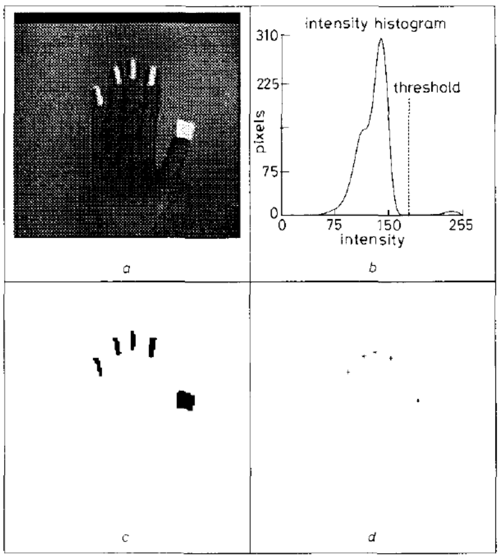


Figure 2.2.: A simple glove with clearly marked at the end of each finger helps detection of finger tips. Image from [8].

objects emit infrared radiation, this is used to detect human parts directly and the camera is calibrated to detect objects with temperatures between 30°C and 34°C. With this information, a human hand can be observed directly without further processing, and segmentation of fingers impose much less processing than with traditional color images.

The segmentation is done by matching a cylinder with a hemispherical cap with each finger, and then filtering the possible candidates to minimize the number of false positives. Furthermore, the center of the palm of the hand is detected by applying a morphological erosion to a rough estimate of the palm, which again is obtained by cutting off the hand at the estimated wrist. The newly detected fingertips are matched with fingertips from the previous frame in addition to estimates of the new fingertips' position. This approach enables recognition of both static and dynamic gestures, and the article concludes that:

> Our system offers reliable, near-perfect recognition of single finger gesture and high accuracy for double finger gestures.

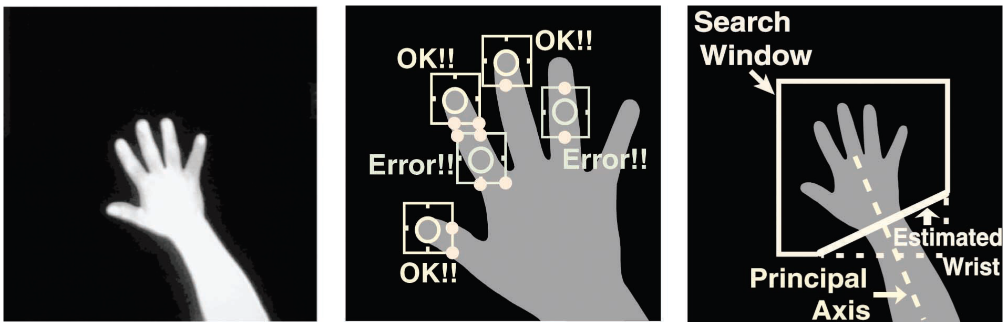


Figure 2.3.: Fingertip detection using an infra-red camera. Image from [24].

However, Oka et al. [24] state that the infrared camera did not work well on cold hands and that this system is not able to detect 3D hand and finger motions, which may be necessary for other gestures.

In addition to these approaches, shape recognition using for example edge detectors or morphology, learning detectors using for instance machine learning techniques called boosting, 3D model-based detection which attempts to match a projected model to the image, and motion detectors which assume that the background constant and only detect moving objects have been implemented. For a more detailed discussion, see Zabulis et al. [44].

Passive Stereo Cameras

In addition to monocular cameras, stereo camera setups are sometimes used. With this setup, the sensor is able to provide 3D information about the environment, which enables new types of gestures. Moreover, this is the sensor which most resembles the method humans use for recognizing gestures, namely human vision. However, this kind of sensor imposes a computational burden when it comes to matching objects in an image from the left camera with the corresponding object in an image from the right camera. This process is called finding the stereo correspondence, and is necessary to know the distance to objects. The stereo correspondence can be calculated in a variety of ways, all with their strengths and weaknesses.

A good comparison of different approaches can be found in Scharstein and Szeliski [35].

Structured Light

A problem with finding stereo correspondence using a passive stereo camera is that texture is needed to be able to pair objects. Texture is not necessarily present on every surface under normal circumstances, so pairing would prove difficult. To overcome this problem, one could use a projector to project texture onto the object which is observed and then complete stereo correspondence.

In fact, if structured light is used, a stereo camera is not necessary as the pattern emitted from the projector is known. The camera then observes how this emitted pattern is displaced by the environment, and an algorithm calculates a 3D grid with points which could have produced the observed data. Chen et al. [5] describe a detailed approach on a structured light system.

A notable solution which uses structured light is PrimeSense's PrimeSensor^{TM}. The sensor works by projecting infrared light onto a scene, and then using a passive camera to record the light [31]. With this approach, the scene is not illuminated with visible light, so measurement of 3D data is possible without disturbing the user. This is the approach used in the Kinect sensor discussed in Chapter 3.

2.2. Gesture Analysis

To recognize a gesture, more than raw data or detected fingers is needed. This section describes several approaches of converting measured information, such as an image of a hand or finger joints' angles, into recognized gestures.

The process of recognizing static and dynamic gestures is based on very different approaches. Often, a dynamic gesture is made of a moving static gesture, thus it needs to first recognize the static configuration and then the path. Furthermore, there is a notable difference in complexity when recognizing continuous gestures compared to isolated gestures, as it is necessary to detect the start and end of each gesture [20].

2.2.1. Static Gestures

Recognizing static gestures is a less complicated task compared to recognizing dynamic gestures. However, some static gesture approaches differ both in what equipment is used and how analysis is performed. Furthermore, static gesture recognition is typically more robust than dynamic gesture recognition.

Angle Analysis

Direct angle analysis is perhaps the most natural choice when using a data glove. Takahashi and Kishino [38] provide an example of this, where information from the data glove is sampled and ten samples are averaged to reduce both noise and minor movement caused by the user. This measurement is then coded so that twelve variables describe the hand's configuration, where ten variables correspond to how the fingers are bent and two variables correspond to orientation. The variables associated with fingers are coded so that an angle of less than 45°assumes a straight joint and an angle greater than 45°assumes a bent joint. However, if the standard deviation of the measurement is greater than 20°the variable is marked as "uncertain". Similarly, the first of the two variables that describe orientation is marked as either "hand pointing upwards" or "hand pointing downwards". The second variable describes if the back, palm or side of the hand is shown.

With this information a data structure which is based on a binary tree is generated so that each leaf node in the tree is a successful gesture. This structure provides an efficient lookup table to find what gesture is the most probable for a given configuration. According to the paper, gesture recognition is performed rather successfully. In five trials, most of the 46 hand configurations were recognized.

Model-to-Image Matching

If information about fingers' angles can not be directly read from a data glove, but a camera is used instead, the model-to-image matching approach would be a replacement [14]. The idea behind this approach is to create a 3D model of the object that is performing the gesture. The model could for instance be a hand with fingers and all the fingers' joints. Reasonable constraints are applied to the model, for example that the index finger must extend from the palm of the hand and not the tip of the ring finger.

For a camera recognize the model, Kuch and Huang [17] suggest to make an initial guess of the current pose. A 2D representation of the model in the current estimated position is projected onto the plane of the image, and this projection is compared to the image from the camera. Based on this comparison, an error variable is calculated and the model is moved or rotated slightly. An error variable corresponding to the new configuration is calculated. It is compared with the old variable, and the configuration with the best match is chosen. With this approach the error is minimized, and a best guess of the configuration is made.

2.2.2. Dynamic Gestures

Dynamic gestures are gestures that require more than a single frame to be recognized. Dynamic gestures have a higher complexity than static gestures, as they can be seen upon as static gestures in motion. Hence, to recognize a dynamic gesture, most approaches consist of recognizing a sequence of static gestures, and how this sequence moves.

Finite State Machines

A Finite State Machine (FSM) is, as the name suggests, a state machine with a finite number of states. A state is a collection of variables which uniquely defines the configuration which a system might be in. A state could for example be a specific gesture, such as “looking to the left”. An FSM is based on the principle that a system can only be in one state at any given time, and in this state a defined number of transitions can happen. For example could the action “look right” change the state “looking straight forward” to “looking to the right”. The state machine can only be in a single state at a given time, that is one can not look to the right and the left at the same time.

This approach can be seen in Hasanuzzaman et al. [12] where an FSM is implemented using a simple FIFO (First In First Out) queue. The queue is used to hold information about which parts of a gesture have been performed. For every frame, the current pose is detected, and if it is different from the previous frame, the new pose is added to the queue (a transition). If the queue contains the images “up, straight forward, down” or “down, straight forward, up”, the a nodding (*Yes*) gesture is registered, and similarly for a shaking head (*No*) gesture.

Another example of an FSM implementation is Davis and Shah [8]. In this paper a specific hand configuration is marked as the starting position. When one of the defined hand gestures is made, the system performs a corresponding action. As this action can be continuous, the gesture may be held for an arbitrary length of time until the starting position is resumed. An example of such a gesture could be that a user points to the left, and while the user points to the left, a robot turns left. When the user stops pointing left and returns to the starting position, the robot stops.

Hidden Markov Models

The Hidden Markov Models (HMM) approach is currently one of the most used techniques for recognizing dynamic gestures. The approach uses a statistical analysis of how the gesture should be treated, and is built on the principle that gestures fulfil the Markov assumption – that is that the Markov property hold for the system. Mitra and Acharya [21] state that:

> A time-domain process demonstrates a Markov property if the conditional probability density of the current event, given all present and past events, depends only on the nth most recent event.

The HMM framework further assumes that there are N states, where each state S, has an output probability distribution function. This function gives the probability that if the system is in state S_i, the system observes that it is, in fact, in the state S_i [42]. In addition to this function, a transition probability function which gives the likelihood of a given action in state S_i results in a transition to state S_j [21].

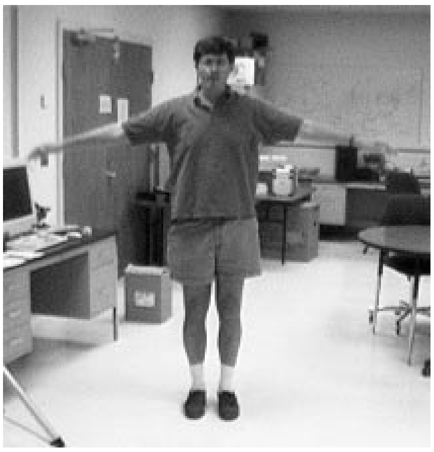
There are three key problems when using an HMM:

1. Finding the probability functions
2. Evaluating the current state
3. Recovering the state sequence

The first problem is often solved by training [42, 21, 24]. Numerous sequences are recorded as training sets, and the correct gesture is associated with the recorded action. This is used as input to the Baum-Welch algorithm [41] to calculate the probability distribution functions. The second step is often solved using the forward-backward algorithm, which "computes posterior probabilities of a sequence of states given a sequence of observations" [34, p. 446]. Lastly, the third problem can be solved using the Viterbi algorithm [9].

Optical Flow

Optical Flow is another approach which is used, for example by Cutler and Turk [6]. The approach is based on detecting optical flow in a set of images, and then running blob detection on the detected flow. These blobs are compared to a generated database of how other gestures should be made, and parameters such as the number of blobs, the direction of motion, the relative motion of two detected blobs and the size of the blobs are compared. For example is “clapping” detected as two blobs with horizontal motion where the blobs have opposing relative motion and a rather small size, while “flapping” is detected as two blobs with rotational movement with the same relative motion and a rather large size. These gestures are shown in figures 2.4 and 2.5.


(a) Flapping action

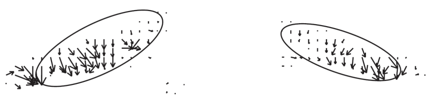
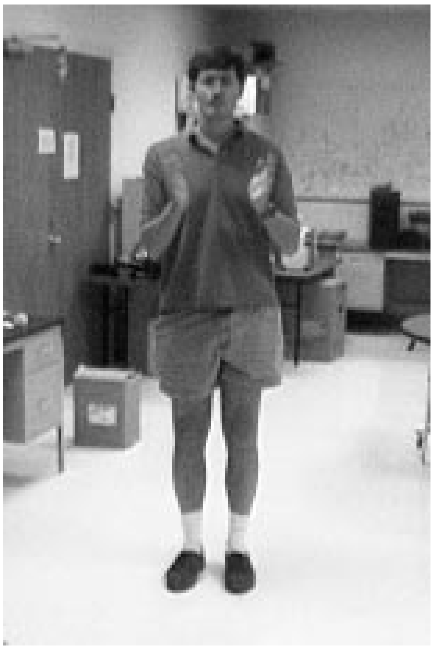

(b) Flow and detected blob

Figure 2.4.: Flapping action. Images from [6].


(a) Clapping action

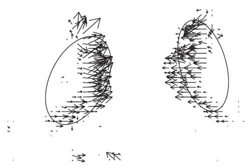

(b) Flow and detected blob

Figure 2.5.: Clapping action. Images from [6].

3. The Kinect as a Sensor

The Kinect, also known as “Kinect for Xbox 360” or “Project Natal”, is a device which originally was meant as a controller-free way of operating the Xbox 360 game console. The sensor has been very popular from the release date November 4th 2010 and sold 133,333 units per day in the first 60 days on sale according to Guinness World Records [33]. From the launch date the sensor’s potential was recognized to perform other tasks than controlling computer games, hence development of drivers for PC was initiated. A notable example of this was Adafruit’s “Hack the Kinect for Xbox 360” prize, where USD 2,000 (later increased to USD 3,000) was awarded to anyone who would provide open drivers for the Kinect [1]. The winner of this contest was announced November 10th [2], and several open source framework followed and as of spring 2011 the two dominating frameworks are OpenKinect’s libfreenect [27] and OpenNI [28] (**Open** Source **N**atural **I**nteraction).

OpenKinect is an open source project which is based on results acquired by reverse engineering communication with the Kinect by observing USB communication. The project aims to support a variety of features such as hand and skeleton tracking, 3D reconstruction and audio cancellation, but these features are not finished as of June 13th 2011 and the project’s “Roadmap” page [27] states:

> Clearly this is a large effort and requires cross-discipline coordination with academic experts, developers, testers, and users. It will also take many months or years to complete this effort.

OpenKinect’s current project is *libfreenect* which allows communication with the Kinect hardware. There are bindings and wrappers to several languages such as C, C++, C# and python.

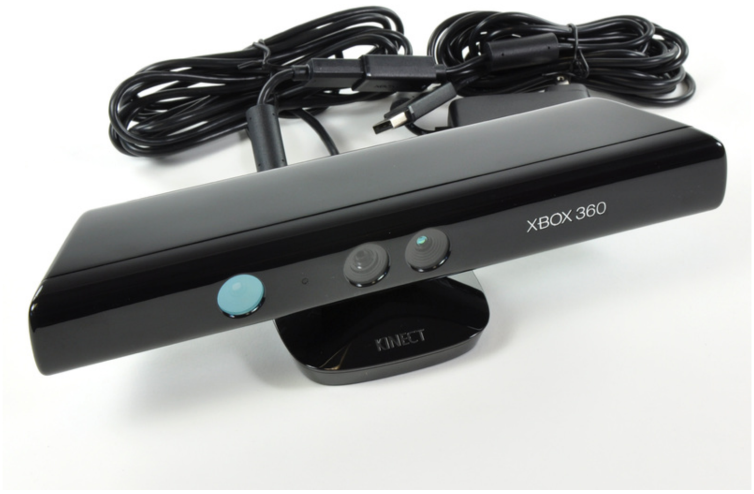


Figure 3.1.: The Kinect sensor including the external power adapter (in the background) which is provided to support older Xbox 360s' USB interface. Image from [15]

OpenNI is an open source framework which utilizes closed source middleware from PrimeSense called NITE. PrimeSense is the company which provides the user recognition software which is used on the Xbox 360 and is further developed than OpenKinect's software. NITE provides several of the features which OpenKinect aims to implement. User recognition, skeleton tracking and limited gesture recognition are all implemented and seem to work robustly within certain assumptions. A short presentation of how this tracking algorithm works is presented in Section 3.2.

3.1. Hardware Specifications

The Kinect is equipped with two sensors - a near infra-red camera used for depth detection, and a color camera. Unofficial sources [3, 10], state that the maximum resolution for these cameras is 640x480px with 11-bit resolution and 640x480px with 32-bit resolution for the depth and color camera respectively. The frame rate is specified as 30 frames per second (fps) for both cameras. The color camera supports higher resolution (1280x1024 px) if the frame rate is decreased to 15fps. In addition the Kinect has an audio interface and a USB controller. As peak power consumption slightly exceeds that which USB can provide according to its specification, an external power supply is necessary. Microsoft has published little information about the hardware, but some information is obtained though analysis of the components which can be found in [26].

According to the retailer Play.com [29], the field of view is as follows:

Horizontal field of view	57°
Vertical field of view	43°
Physical tilt range	±27°
Depth sensor range	1.2m - 3.5m

The detection algorithm uses structured light which is described in Section 2.1.2 and the depth resolution is, according to Shotton et al. [37], a few centimeters.

3.2. Detection Algorithm

When data from the Kinect is acquired the depth image, that is image with depth information, is analyzed to extract information about users' position and pose. According to Shotton et al. [37], the software used with the Kinect sensor is the first "robust interactive human body tracking" which runs at "interactive rates on consumer hardware while handling a full range of human body shapes and sizes undergoing general body motions". The algorithm works by dividing a user's body into 31 labels which are recognized as 3D approximations of the user's body joints. An example of this segmentation is shown in Figure 3.2. The algorithm is optimized using a GPU and uses less than 5ms per frame. The following paragraphs describe Shotton et al. [37]'s approach.

Analysing depth images has several advantages compared with analysis of color images; depth images provide high quality data in low light settings,

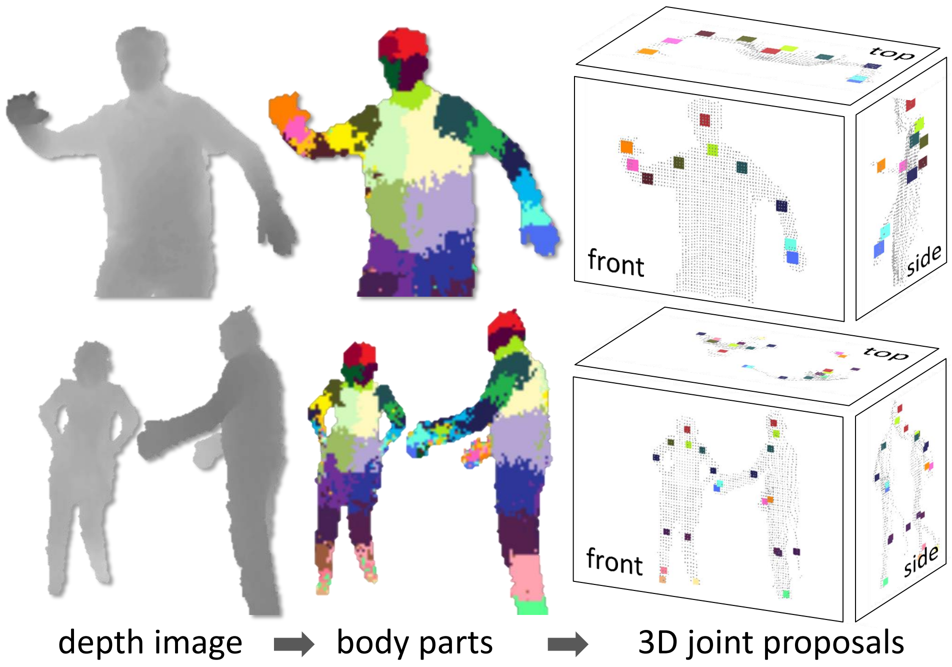


Figure 3.2.: Recognition of users' body joints from depth images via segmented body parts [37].

they are color and texture invariant, the scale is calibrated and background extraction is simplified. In addition, synthetic training sets are more easily generated with depth images than with color images, hence populating training databases is simplified. To create this kind of training database is, however, a formidable task and the paper reports using a database of approximately half-a-million frames in a few hundred sequences.

To segment a body into different parts an image classifier is used. The image classifier used in this approach is based on randomized decision forests. It is trained using a subset of the database mentioned above, due to very similar neighbouring poses in a moving gesture, with approximately 100 000 static poses. The CMU mocap database [40] was used in early experiments and provided acceptable results for the limited set of poses.

The input to the randomized decision forests is a set of features $f_\theta(I, \mathbf{x})$. A feature is defined as a function which is defined for any image I at any position $\mathbf{x}$, and takes the parameter $\theta = (\mathbf{u}, \mathbf{v})$. θ describes offset in a fixed world space frame and is scaled inside $f_\theta(I, \mathbf{x})$ so that the features become 3D translation invariant. According to the article, the features only give a weak response to which part of the body a given pixel belongs to, but when using decision forests it is sufficient.

To train the decision trees, a random subset of 2000 pixels is selected from each image and an algorithm, which is based on partitioning and Shannon entropy calculated from the normalized histogram of the body part labels, is used. Training 3 trees to depth 20 from 1 million images takes about a day on a distributed implementation with 1000 cores. Further details are found in the paper.

To extract body joints an algorithm which consists of three main steps is used:

- A density estimator per body part with a weight based on the body part probability and the world surface area of the pixel.
- A mean shift technique to find modes in the density efficiently.
- A “push back” algorithm which translates the approximated joint

location, which is placed on the user's observed surface, to the most likely 3D placement inside the point cloud.

The paper reports that this joint detector is very precise, with 91.4% of the joints correctly detected with less than 10cm to the ground truth. In addition, when only evaluating the head, shoulders, elbows and hands, 98.4% of all joints are properly detected within 10cm.

3.3. Limitations

Although Shotton et al. [37]'s approach seems very favourable, both hardware and software impose certain limitations. Although the limitations might be insignificant when the sensor is used as a game controller, they might be vital when for example using the sensor in robotics. The major limitations are as follows:

- Does not work in sunlight (hardware)
- Reflective and transparent surfaces not properly detected (hardware)
- The resolution limits fine-grained gestures such as finger gestures from a distance (hardware)
- Certain objects are simplified or undetected (hardware or firmware)
- The Kinect is assumed stationary (software)
- User initialization takes time (software)
- User labels may switch when the Kinect is moved (software)

3.3.1. Sunlight

According to OpenKinect [26] the light projected by the Kinect is from a 60mW 830nm laser diode. As sunlight has a wide spectre of infrared light, the grid projected by the Kinect is blinded by bright sunlight and the IR-camera is unable to detect the grid. This yields very poor depth recognition in areas exposed to much sunlight.

3.3.2. Reflective and Transparent Surfaces

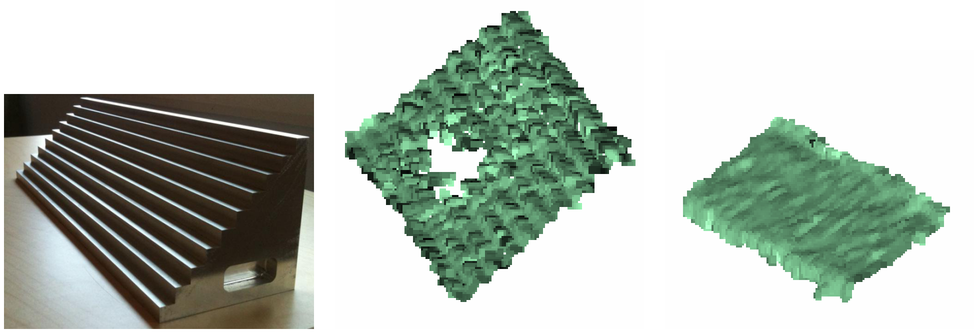
Detection of reflective or transparent surfaces is always difficult when using optical sensors, and this is also the case with the Kinect. This difficulty is due to the fact that most optical sensors observe light reflected from an object, and if this reflection is either lower or higher than expected, observation tends to be difficult, as little information reaches the sensor.

3.3.3. Limited Resolution

Due to the somewhat limited resolution of the Kinect, at 640x480px with a few centimeters depth resolution, all types of gestures can not be registered. Typical gestures could be waving, holding up an arm, or bending one arm to a 'stop' position if the whole body is visible. However, if a smaller area of the body is observed, such as an arm or the upper body, more detailed gestures such as one-hand gestures representing letters from the American Sign Language can be recognized.

3.3.4. Certain Objects Simplified or Undetected

According to Øystein Skotheim [43], the Kinect has problems when detecting certain objects. This can be seen when observing a step object which consists of 10 steps that are 10mm high and 10mm deep. As Figure 3.3 shows, some objects become smoothed when observed by the Kinect - even at. In addition to this, objects such as hair are often too fine-grained for the Kinect to discover.



(a) Steps object (b) 84cm (c) 120cm

Figure 3.3.: Certain objects become smoothed when observed by the Kinect, even from a rather short distance. Reprinted with permission from [43].

3.3.5. The Kinect is Assumed Stationary

To simplify background extraction the Kinect is assumed to be stationary, and thus that everything that moves is very likely to be a human. Hence, when the Kinect is mounted on a moving platform and this assumption no longer holds, many false positives are detected. This is a major problem when using the Kinect on a mobile platform, and must be handled explicitly if the sensor is to be used this way.

Figure 3.4 shows multiple "users" which are false positives when the Kinect is placed on a mobile platform. Note that the silhouette of the correct positive, the green user, is very accurate.

3.3.6. User Initialization Takes Time

When a user is detected, he or she must stand in a special stance - arms straight out from the shoulders with forearms and hands pointing upwards - for several seconds before the algorithm calibrates a virtual skeleton and adjusts it to the user. This is also a major problem on a mobile platform if the platform travel with a somewhat decent speed. It would

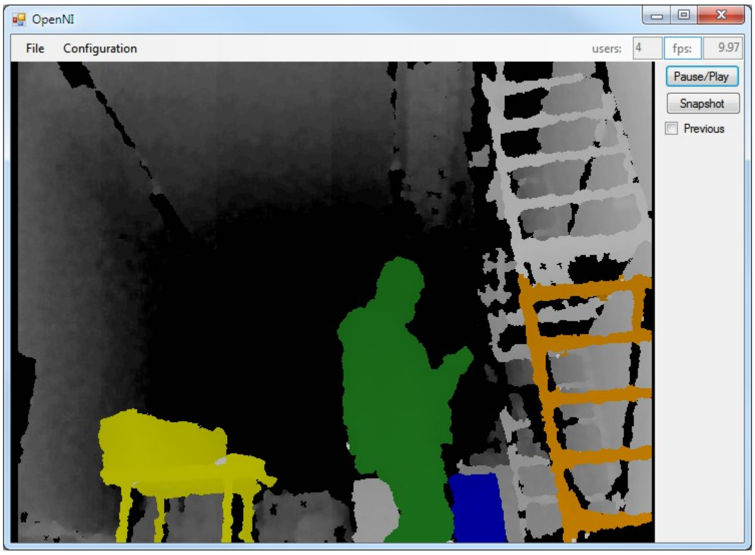


Figure 3.4.: Several false users are detected, marked with color, by the standard NITE middleware when the Kinect moves. The correct response would have been only the green silhouette.

be very unfortunate if the platform passed the user before calibration was finished.

3.3.7. Inconsistent User Labels

Another major problem is that user labels may switch between frames. The problem arises when the labeling of a user is lost for one or more frames and when the user is rediscovered, it is marked as another user. The problem is illustrated in Figure 3.5.

This inconsistent labeling causes problems when tracking a user along multiple frames, and when comparing one frame with the next. If this user switch is made during a gesture, a gesture recognizer may interpret the input as two different users and not relate the gesture performed by the "first" user to that of the "second" user. Hence, this must be improved to enable observation from a moving platform.

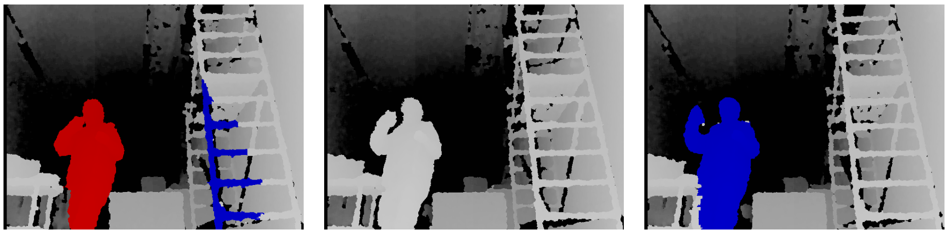
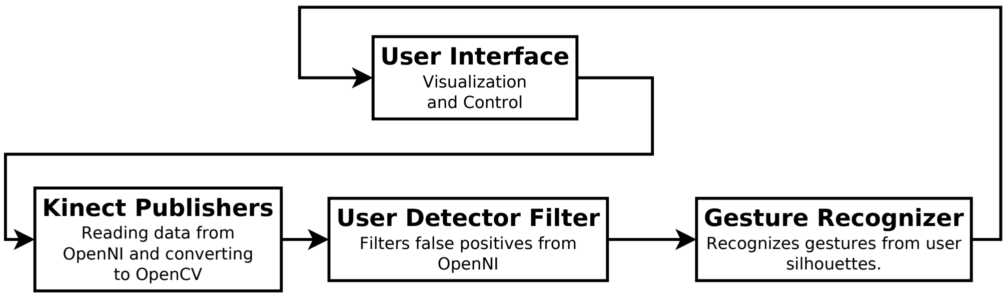

(a) Frame 1 (b) Frame 2 (c) Frame 3

Figure 3.5.: The figure shows three consecutive frames where the user is first marked by one label (shown as red in Frame 1), then the user labels are lost in Frame 2, and when user labels are available in Frame 3, the user has been marked with the wrong label (shown as blue).

4. System Design

To conquer some of the limitations mentioned in Section 3.3, a system for data handling, analysis and visualization was created for this thesis. As hardware limitations are difficult to improve without modifying the hardware itself, the focus of this thesis has been on improving the software limitations. The problems arise from operating in environments where the assumption that the Kinect is stationary no longer holds.

The system is composed of four major components. These are: the Kinect data handler, the user detector filter, the gesture recognizer and a visualization and control component. Communication between nodes is event driven to reduce polling, as polling spends time on unnecessary checking of variables. An overview of the system is shown in Figure 4.1.



User Interface
Visualization and Control
Kinect Publishers
Reading data from OpenNI and converting to OpenCV
User Detector Filter
Filters false positives from OpenNI
Gesture Recognizer
Recognizes gestures from user silhouettes.

Figure 4.1.: System overview of the four main components.

4.1. Kinect Publisher

To read data from the Kinect sensor, the OpenNI interface is used. This interface was chosen because it is further developed than the libfreenect interface. In addition, OpenNI is implemented as the standard Kinect package in ROS (Robot Operating System), which can be used for controlling the robot at a later time. To ease communication and and the ability to interchange parts of the system, a package called “Kinect Publishers” was created for this thesis. This package abstracts data extraction from the Kinect and converts the data format to the OpenCV format used by the rest of the system.

At project start-up the current unstable version of the OpenNI interface was 1.0.0.25 (published Jan 10th 2011), and the current unstable version of the NITE middleware was 1.3.0.18. These are the versions used in this project. The OpenCV version used is 2.2 with Emgu CV version 2.2.1.1150.

The package consists of two main components - the *DepthReader* and the *UserReader* which read and buffer depth images and user silhouettes respectively. The package runs in its own thread to utilize cores on multi-core architectures. The idea behind this package is that the rest of the system should not need to wait for the external device when data is needed. The buffer is, however, rather small because if the system can not process the information fast enough, it is desirable to process as new information as possible.

4.2. User Detector Filter

Because of the problems when using the Kinect on a moving platform, presented in Section 3.3.5, the false positives have to be removed from the data set. This is done by the improved user detector step. The system is designed so that multiple classifiers can be run in parallel, and two classifiers are implemented: one based on optical flow and one based on a simple feature matcher. The output from the improved user detector is a silhouette of a user which is more probable to be a human than the estimates from OpenNI. This implementation is further discussed in Chapter 5.

4.3. Gesture Recognition

When reliable user silhouettes are available, gesture recognition can begin. The algorithm performs three steps in recognizing a gesture. Firstly, a user silhouette is segmented and labeled. This is a simplified approach based those discussed in Section 3.2 and Oka et al. [24]'s approach mentioned in Section 2.1.2. Secondly, the position of the hand, the position of the elbow and the position of the shoulder are detected. Then these joints are used as parameters to a finite state machine which recognizes the gestures. This implementation is further discussed in Chapter 6.

4.4. Visualization and Control

In addition to these core components, a visualization and control package is created. This package handles the execution and connection between the different components in addition to parameter adjustment and visualization of the output. To be able to fine-tune parameters and test different approaches several configuration options are available, in addition to the ability to play recorded data steams. A class which extends .Net's native PictureBox was created to allow zooming to better show details of the algorithm.

5. User Detector Filter

As the algorithms that run on the Kinect assume that the Kinect is stationary, many false positive users are detected. Although neither Microsoft nor PrimeSense have released detailed documentation on how the device works, the reason for the false positives is most likely an assumption that objects which move relative to the Kinect, have a high probability of being human. However, when the Kinect moves, most of the surrounding objects move from the Kinect's point of view, thus they are marked as users.

The user detector filter improves the accuracy of OpenNI's results by filtering out these false positives, thus only allowing correct users to be further processed by the gesture detector. To classify a human user from the background, several criteria could be set to verify or falsify if a given user should be counted as a user or the background. The two criteria used in the system developed for this paper is that a verified user should move differently from the Kinect and that such a user must have somewhat the same features as a human. Both of these criteria are needed as calculation of optical flow is somewhat noise-intolerant, and as there are objects which can not be filtered out by the feature filter because of human resemblance.

To be able to efficiently apply various criteria, each criterion is implemented as its own filter. This approach was chosen to allow multiple criteria to be applied in parallel, and to allow extendability and the ability to select specific filters optimized for specific environments.

The result from each filter consists of three lists: verified users, falsified users and uncertain users. These lists contain each user visible at that given time instance, once in one of the three lists. When all filters have finished execution, the lists are combined in a voter. The voter is a simple process that decides how the results from the different filters should be interpreted, and it is implemented in the following way:

- If one or more filters have marked the user as false, the user is marked as false.
- If one or more filters have marked the user as valid, but no filters has marked that user as false, the user is marked as valid.
- If all filters marks a user as uncertain, the user is marked as valid.

As only two filters are implemented in this thesis, the given voter works sufficiently. For example, if three filters marked a user as valid, while one marked the user as false, it will still be marked as false. However, a more complex voter might be useful if more filters are implemented.

Currently, there are only implemented two criteria which can verify or falsify users. These are: the criterion that movement of a user should be different from that of the surroundings, and that a user silhouette should have some similarities to a human silhouette. These criteria are implemented in the $OpticalFlowUserFilter$ and $TemplateUserFilter$ respectively. However, before the user data is handled by the filters, users are relabeled in case of the inconsistent user label problem.

5.1. Inconsistent User Labels

Although the OpenNI library usually gives good results, with good tracking of each user and consistent user labels, there are some instances where this tracking is lost. Although as a missing frame usually is no problem at a rather high frame rate, a more serious problem occurs when the tracking is re-initiated. This is the problem described in Section 3.3.7, where a user with a specific label, for example “red”, is marked with a different label, for example “blue”, after one or several frames where that specific user is not detected. This is problematic because if relabeling happens in the middle of a gesture, it will seem like the gesture is started by one user and finished by another, hence the gesture will not be recognized correctly.

The idea behind the relabeling algorithm is to map the currently observed labels to previously observed labels using the centroid of each silhouette. The centroids are calculated and stored for each user for each step, and the centroids from the previous step that are the closest to the ones from the current step are matched with each other. An illustration of the approach is given in Figure 5.1.

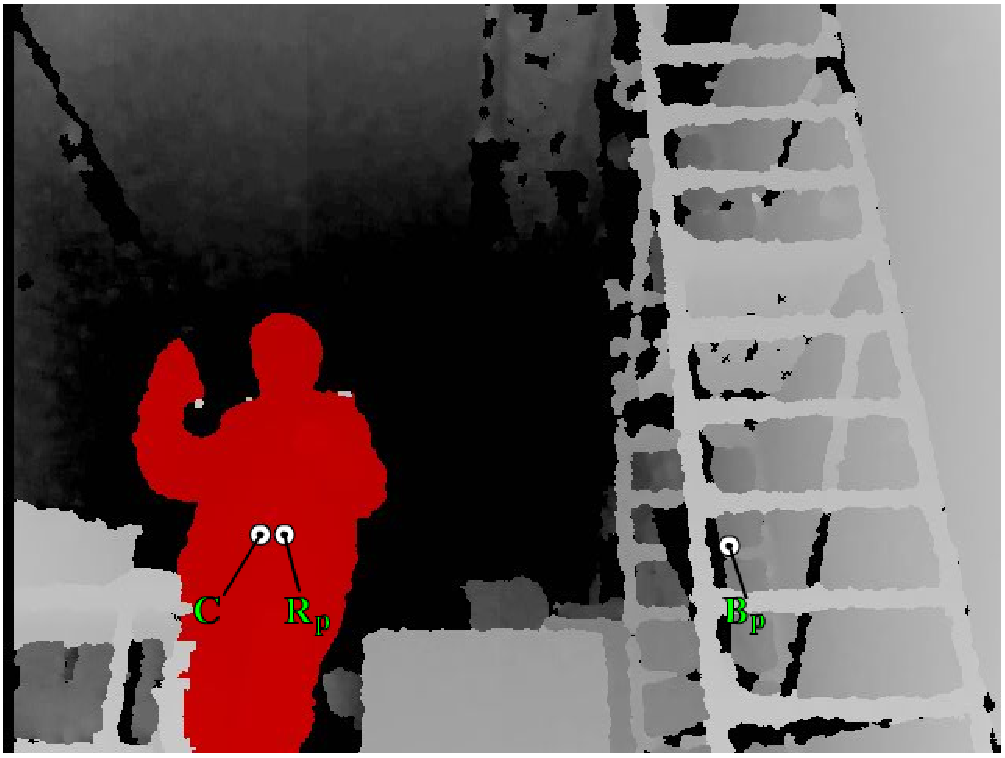


Figure 5.1.: The distance between R_p (Red previous) and C (Current), and the distance between B_p (Blue previous) and C are calculated, and the shortest distance to each "current point" is selected. In the image above, the C had the shortest distance to the red label and therefore it was marked as red.

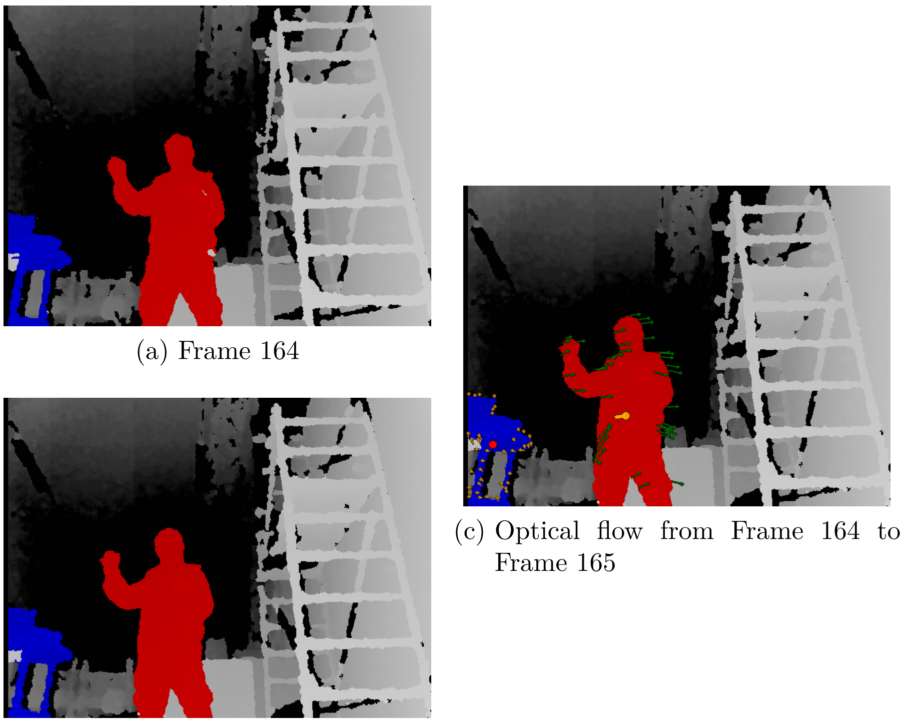
To keep the computational burden low, the current implementation only selects the single best match for each label. This simplification might yield suboptimal relabeling, because selecting the locally best result might not give the globally optimal result. Although the current implementation seems sufficient, as is shown in Section 7.1, a suggested solution to this suboptimality is presented in Appendix B.

When the locally best matches are found, the algorithm stores the labels in an array called *labelsRemap*. The array contains a mapping so that at a given index, which corresponds to a label provided by OpenNI, *labelsRemap* contains the optimized label. If an optimized label does not exist for a given label, the algorithm selects OpenNI's suggestion. However, if this label also is chosen as an optimized result for another user, the two labels are swapped. With this approach every user is ensured to have a different label. When *labelsRemap* is processed, the algorithm iterates through the image and copies the image, but instead of copying the value d, the value $labelsRemap[d]$ is copied. In addition, the list of visible users is converted in the same way.

Because the labeling is done after the images are read by the Kinect, but before the images are sent to the filtering process, they are transparent to the filters. Hence, the labeling can be turned on and off or changed without changing the interface or data handling of the filters. When the labeling is done, the data is sent to the different filters.

5.2. Optical Flow User Filter

The *OpticalFlowUserFilter* is, as the name suggests, based on optical flow in the image. Optical flow is calculated per user silhouette and an average of the detected movement is registered and compared to the Kinect's movement. To calculate optical flow, OpenCV's algorithm *calcOpticalFlowPyrLK* which uses an iterative implementation of Lucas-Kanade optical flow in pyramids is used [25, 4]. To find points in one frame which have a high probability of being detected in the next frame, *cvGoodFeaturesToTrack* - a corner detector based on Shi and Tomasi [36]'s approach, is used. The function *calcOpticalFlowPyrLK* finds out where the provided points have moved in the given frame, and the difference between the provided and the calculated points is used to approximate the velocity of the suggested user. See Figure 5.2 for an illustration of the differences registered between two images.


(a) Frame 164

(b) Frame 165

(c) Optical flow from Frame 164 to Frame 165

Figure 5.2.: Shows the optical flow calculated at the different points. In c) one can see the transition from the points in Frame 164, marked with a circle, to the points in Frame 165, marked with the end of the line.

When this is calculated, the suggested user with the highest difference in velocity relative to the Kinect is selected as the verified user, if the difference is higher than a specified threshold. If the difference is lower than the threshold, the previously selected best user is selected as the verified user. The algorithm also keeps track of the two latest verified users, and these are put in the “uncertain” list.

5.3. Feature User Filter

The other implemented filter, the $FeatureUserFilter$, tries to efficiently filter out suggested users by calculating several features which are associated with human silhouettes. These features are compared to those of the suggested users and if the difference between the expected values and the calculated values is higher than a certain threshold, the users are falsified.

At this time there are three features that are calculated, a simple area measurement, a feature called the $AverageWidestHorizontalFillPercentage$ and a feature called the $AverageHighestVerticalFillPercentage$. The area is used as it is very easy to calculate and is a good indicator on if the user could resemble a human. The average horizontal fill percentage and the average highest vertical fill percentage are measures on how dense the user is. To calculate the horizontal fill percentage, the suggested user silhouette is scanned horizontally and the widest continuous part of each line. To find the widest horizontal fill percentage, the widest line is divided by the total width of the row (the number of pixels between the first observed and the last observed pixel), and these percentages are averaged over the whole body. To calculate the highest vertical fill percentage the process is repeated vertically. The intent of these features is to exclude objects such as tables, where the table legs will have very low fill percentage compared to a human.

5.4. Implementation Details

To allow the system to have multiple criteria for verifying and falsifying a user silhouette, a system for loading and handling different filters has been implemented. The system initializes the different filters and handles events that are sent from the Kinect Publisher package. These events are analyzed in each filter before the resulting events are sent to the voter. The voter combines the outputs from the different filters, before the final, filtered result is sent to the Gesture Detector package.

An abstract class ($UserFilterWorker$) which all the different filters extend was also created. This class handles thread creation, each filter is executed in its own thread, in addition to control and data handling. These threads are event-driven and safe communication between threads is also handled, to some extent, by the abstract class.

Communication between threads are as shown in Figure 5.3. Due to the event-driven implementation, each filter can run in its own thread and all the filters are executed in parallel. As calculation of the movement of users is independent of the calculation of the shape resemblance, and vice versa, the filters do not need to communicate with each other to complete each task. Because the number of filters is low (only two), the gain of concurrency is not as noticeable as one might hope, but if several filters are added, multi-core utilization is increasingly important, hence a concurrent approach is prefered.

With this somewhat complex model regarding data management and protection, the abstract class greatly reduces the implementation time and effort needed to create new filters. This is because most of the complex data flow handling is inherited from the abstract class and this results in increased development time available to make the filter do what it is supposed to.

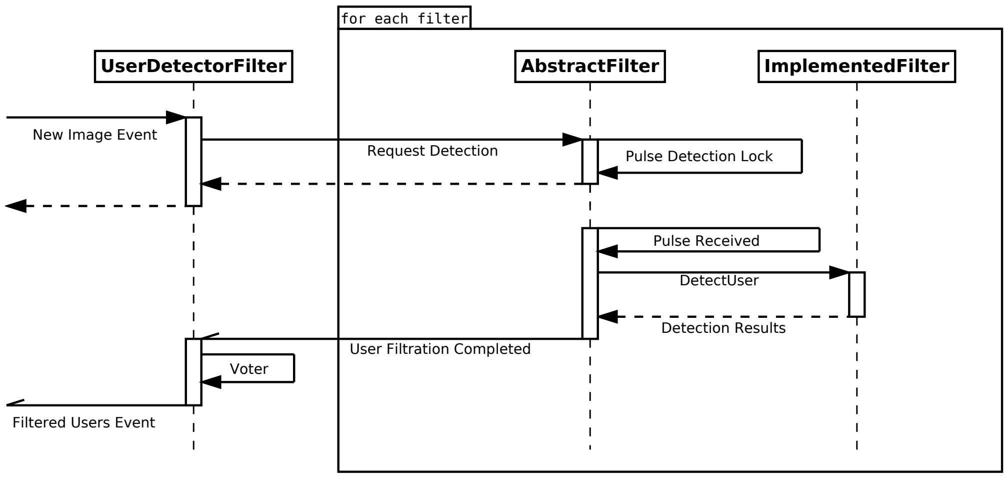


Figure 5.3.: The figure shows communication from the external “New Image Event” is received by the *UserDetectorFilter*, to the data is handled by the voter and the “Filtered Users Event” is raised. Although the implemented filters extend the abstract filter, the class is split into the part inherited and the part extended. This is done to show that the implemented filter only needs to override the *DetectUser* method call.

6. Gesture Detection

When the user detector filter has removed the false positives that the OpenNI interface produced, gesture detection can initiate. The original idea in this thesis was to send the filtered data back to the OpenNI interface, but due to the initialization time of the Kinect's algorithm and lack of access to the source code of the NITE middleware, this proved both suboptimal and problematic. Because of this, a new joint detection algorithm and gesture detector is implemented for this thesis.

The gesture detector module assumes that a high quality silhouette of a user is provided from the improved user detector module, and analyzes this to detect simple gestures. Gesture detection is then executed in a three-step process:

1. Segmentation and labeling
2. Joint detection
3. Pose state machine

In this chapter, all references to the left or right side is the user's left or right side when looking at the Kinect, and hence the opposite when observed from the camera of the Kinect.

6.1. Segmentation and Labeling

The segmentation and labeling step divides the user into several different regions. The segmentation algorithm is based on some assumptions: the person must be standing rather straight and be turned somewhat towards the Kinect. In addition, the current implementation does not work sufficiently if one or both of the arms are in front of a user's head. All in all, there are four main regions that are detected: legs (if they are visible), torso, head and arms. An example of a segmented user is shown in Figure 6.1.

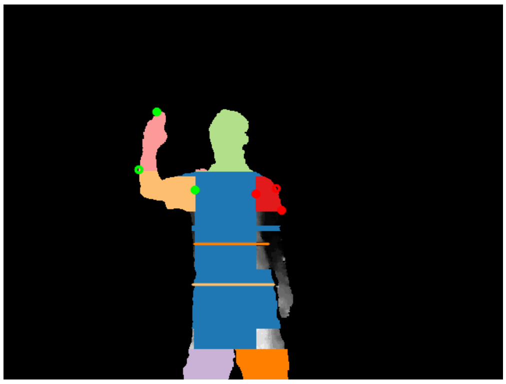


Figure 6.1.: The image shows a segmented image of a user. The left and right legs, from the user's perspective, are segmented and labeled as orange and light purple respectively. The torso is labeled with a blue color. The head is labeled with a light green color, and the left and right shoulders are labeled with a red and yellow color. The right arm is marked as pink and the left arm is undetected (gray), as it is outside the valid area which is defined later.

To provide data for the segmentation, the silhouette of the image is scanned from top to bottom and every horizontal line is stored in an array. A row is divided into multiple lines at any point where the silhouette is empty, and the widest line at each image-row is stored in a separate array. An explanation of some of the terms used with respect to the lines is given in Figure 6.2.

6.1.1. Leg Detection

The first body parts to be segmented are the legs, as the process of detecting these is the simplest. To detect the legs, the array of lines mentioned

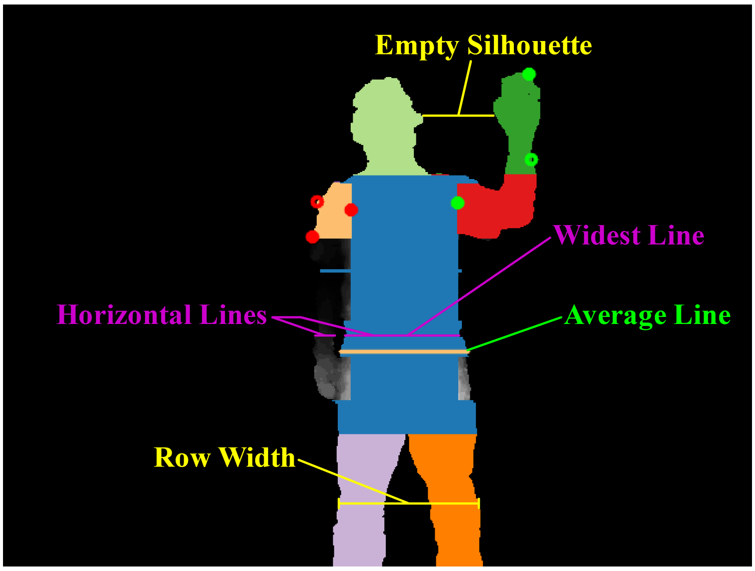


Figure 6.2.: Some of the terms describing the lines used for segmentation.

above is scanned from the bottom and up. As long as two lines are detected at the currently most bottom row, they are marked as legs. If more than two lines are detected, a test is performed to see whether one or more of them are below a given threshold, so that they should be detected as noise (which may occur at the circumference). If there are exactly two lines that are not marked as noise, the algorithm proceeds until there are no such two lines. The first leg detected is marked as (the user's) right leg, and the second the left leg.

6.1.2. Torso Detection

When the legs have been properly handled, the torso is detected. There are four criteria for a line to be accepted as the torso:

- The line must be the widest at any row
- The line must not be shorter than the average widest line multiplied with a threshold ($MinBodyWidth$)

- The line must not be wider than the average widest line multiplied with a threshold ($MaxBodyWidth$)
- The line must not already be labeled

The thresholds can be configured in the GUI, but the default values have given good results on several users. The average line is calculated from the lines which are not already labeled as legs and is either a simple mean of the values (default), or the median line of the set (configurable in the GUI). If the lines are too short to be the body, they are marked as such, and if they are too wide they are split into two or three new lines (the lines are split as they are likely to be one of the arms).

To split the lines that are too wide, the algorithm iterates through the widest lines that have not already been labeled. If the current line is marked as too wide, the line's width is set to the width of the previous line. However, the line can not exceed the original width of the current line, and the line can not be set so short that it would be detected as "too short". If the line is sufficiently cut on either side, new lines are added to be detected at a later stage, as they are likely to be the user's arms.

6.1.3. Head Detection

To detect the head, the lines are iterated once more from the bottom and up. The criteria this time is that there should not be any line that was too wide to be the torso above the place where the head started, as the head should be made of lines that are too short to be the torso. In addition, the center of each line which builds up the head should not be too far from the average. This is because these lines probably are one of the hands which is raised so that it is to the side of the head. The detection of the head is rather important as it is used as a relative measurement for both the arms and joint detection described later in this chapter. This is used to provide a user independent algorithm which allows users of different height and body shape to be detected.

If head detection was unsuccessful, that is if the size of the detected head is smaller than threshold (default is 20x20 pixels), further processing is stopped. This option is included to gain a more robust behaviour towards non-human silhouettes that are not filtered out by the User Detector Filter.

6.1.4. Arm Detection

The only remaining segmentation step is the segmentation of the arms. This approach is somewhat simplified to optimize computation time du to the limited information needed for gesture recognition. In this project, only gestures where one or both arms are raised approximately straight out to each side and moved in a semi-circle around the elbow are considered (see Figure 6.3).

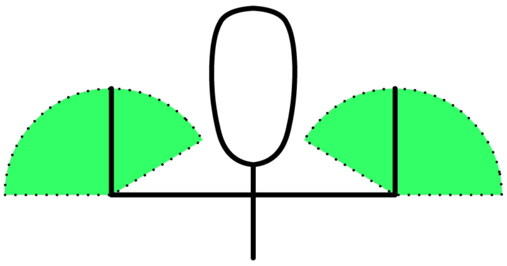


Figure 6.3.: The ranges where arm movement is detected are marked by the dotted lines. Gestures may include of one or both arms.

As a user's shoulder is located somewhat below that user's head, this is used to detect the shoulders. As the algorithm aims to be distance independent, the height of the user's head is used to provide an estimate of how far down on the user's body the algorithm should look for the shoulders. This is based on the assumption that a person's body parts are proportional to each other, as described by Leonardo da Vinci in the text accompanying the famous Vitruvian Man shown in Figure 6.4.

The current implementation uses the user's head's height multiplied with a configurable ratio, *HeadShoulderRatio*, to mark the shoulders. The algorithm labels every line which is marked as "Undefined" between the last line of the head, and the head height multiplied with the *HeadShoulderRatio*, as: *Noise*, *LeftShoulder* or *RightShoulder*. The line is marked as noise if the width is less than a configurable threshold, and marked as either the left or right shoulder depending on its placement relative to the center of the average line.

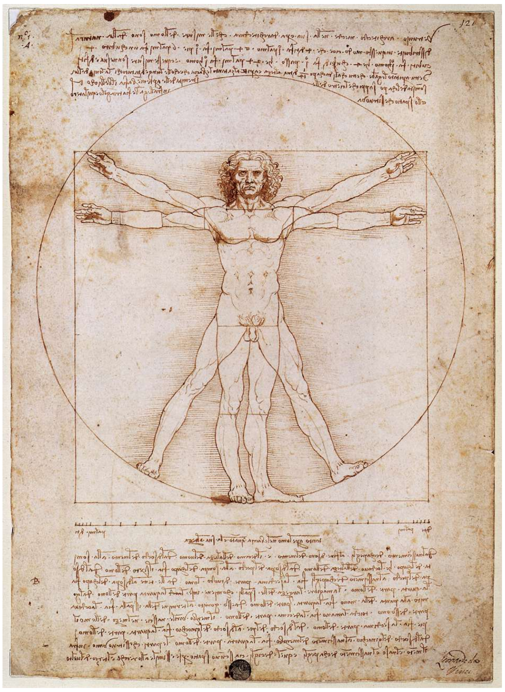


Figure 6.4.: Leonardo da Vinci's Vitruvian Man describes the proportions of a "well shaped man". The information in the accompanied text correlates with that of the Roman architect Marcus Vitruvius Pollio in his work De Architectura [30].

When the shoulders are detected, the algorithm uses these as a basis to segment the arms. To initiate the segmentation, the algorithm labels all lines from the bottom of the area marked as shoulders, to the bottom of the head as the left or right arm. Then, as long as there has been no line where no arms were detected, the unlabeled lines above the bottom of the head are marked as either the left or the right arm. If one of the arms does not have an associated line at any given height, the lines above are not marked for that arm.

After this step a test similar to the one done after head segmentation is performed. The test checks that at least one of the arms contain more than a specified number of lines (default 10) to be processed by the joint detector. If this is not the case, the probability of the detected user being human is low and detection of body joints is skipped.

With this, all the information needed for body joint detection is ready, and this detection can begin. Before this, the segmented image is visualized by the program as shown in Figure 6.1.

6.2. Joint Detection

When segmentation is finished, six of the user's body joints are recognized if they are within the detectable range. These joints are as follows:

- Left and right shoulder
- Left and right elbow
- Left and right hand

To simplify body joint detection, the algorithms for detection of the left and right joints are mirrored. This mirroring brings trivial changes, such as where the leftmost point is chosen for the left shoulder, the rightmost point is chosen for the right shoulder.

6.2.1. Shoulder Detection

The first joint to be detected is the left shoulder joint. This is calculated by simply averaging the lines that are labeled as $LeftShoulder$, and the height position (y-value) of this average is used to mark the shoulder joint. The x-value is selected as the leftmost point of the leftmost line that is marked as $LeftShoulder$ and is at the same height as the shoulder joint. To find the right shoulder joint, the process is repeated with the trivial changes mentioned in the previous section.

6.2.2. Hand Detection

When an estimate of the shoulder is found, the point furthest away from this, but still inside the silhouette of the arm (marked as either shoulder or arm) is detected. The distancing method used is the sum of squared differences between the observed pixel's x and y values and the x and y values of the shoulder point. The depth is not taken into account as it adds another dimension of complexity and complicates the calculation, thus increasing execution time. The detected point is in most cases the hand, but it can also be the elbow if the arm is bent towards the head.

In case the detected point corresponds to the elbow instead of the hand, the algorithm detects vertical lines in the image of each arm. With this approach, wherever there are two or more lines, the probability of these lines representing the forearm and overarm is high. There are, however, some cases where the torso segmentation creates double lines. Because of this, a flood-fill is executed until every region is filled, and the two largest regions are selected as the forearm and overarm. If the forearm is sufficiently large (it must be larger than the distance between the shoulder to the previously detected hand candidate multiplied with a configurable ratio) a new hand candidate is calculated by finding the point in the forearm which is the furthest away from the previously detected point (the assumed elbow).

6.2.3. Elbow Detection

The last step of the body joint detection algorithm is to detect the elbow. The approach for this is to select the point which is furthest away from the shoulder and the hand, but still connected to the arm. This is done by solving a simple optimization problem, where for each pixel in the arm image, the distance to both the hand and the shoulder is calculated. Then the minimum value of these two distances is found and compared to the maximum distance found so far. This maximized point, that is the point where the minimum distance is maximal, is stored as the elbow.

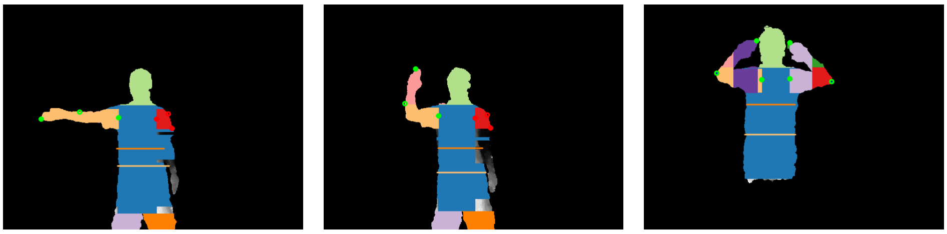

(a) Right arm straight (b) Right arm raised (c) Arms toward head

Figure 6.5.: Three different scenarios where the user silhouette is detected and body joints are displayed. Detected joints are shown by a green circle. Detected joints which are rejected are shown as red circles.

6.2.4. Joint Rejection

To know if the joints represent valid points, the set of joints must fulfill two criteria:

- the distance between the shoulder joint and the detected furthest point away from the shoulder must be larger than the height of the head multiplied with a ratio ($HeadArmRatio$)
- the hand point can not lie more than a certain distance below the shoulder point.

This is done to ensure that the joints that are sent to the pose recognizer are valid, to reduce the risk of a false positive gesture recognition. This also makes the algorithm more robust if the estimates from the user filter is wrong, or that the user has a posture the gesture detector does not recognize.

6.3. Pose State Machine

Reliable gesture recognition can be implemented using the relative position of joints detected by the method described above. The pose state machine is implemented using a finite state machine with four states per arm. Which state is detected at a given time step is dependent on two factors: the current position of the joints and the previous state.

To increase the robustness of the detector, gestures are required to have at least one of the arms either raised or towards the head. This is done because the state machine changes to the “down” state if it is in the “straight” state and tracking is lost for a single or a few frames.

Right / Left	Down	Straight	Raised	Towards head
Down			X	X
Straight			X	X
Raised	X	X	X	X
Towards Head	X	X	X	X

Table 6.1.: There are 12 arm configurations which result in an allowed gesture. Legal gestures are marked with an X.

7. Results

This chapter shows the results of each component of the program and subcomponents with interesting results are presented in their own sections. A discussion of these data is provided in the next chapter.

The results in this report are collected by running the developed system with different configurations to show the effects of each component. Source data consists of recorded video files, captured using OpenNI's sample program $NiViewer$, which are replayed using the developed system. In addition to recorded video files, the system accepts real-time data from the Kinect, but this is not used in this report as certain features can not be shown using different configurations.

7.1. User Relabeling

Snapshots showing the user relabeling process is shown below. Figures 7.1 and 7.2, and figures 7.3 and 7.4 contain snapshots from the first data set while figures 7.5 and 7.6 are snapshots from the second data set. As the relabeling happens rather infrequently, there are not many results where this effect is shown, hence the number of figures is limited.

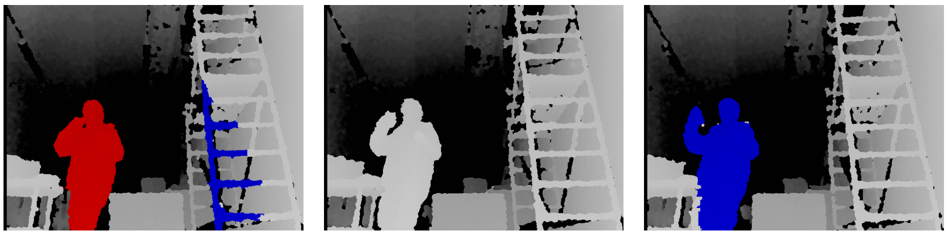

(a) Frame 172 (b) Frame 173 (c) Frame 174

Figure 7.1.: Snapshots when using the original OpenNI code

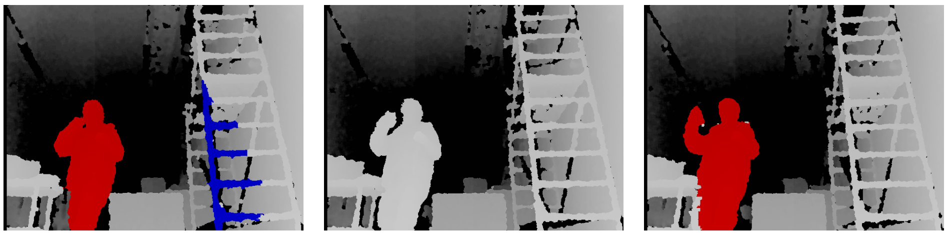

(a) Frame 172 (b) Frame 173 (c) Frame 174

Figure 7.2.: Snapshots when enabling user labeling

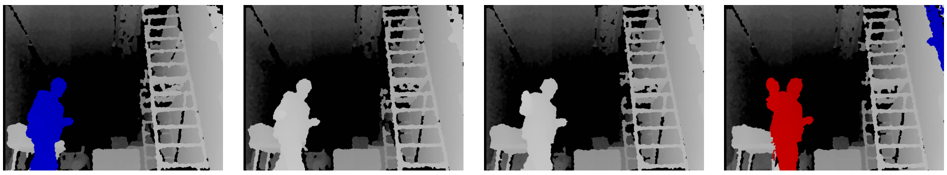

(a) Frame 194 (b) Frame 195 (c) Frame 196 (d) Frame 197

Figure 7.3.: Snapshots when using the original OpenNI code

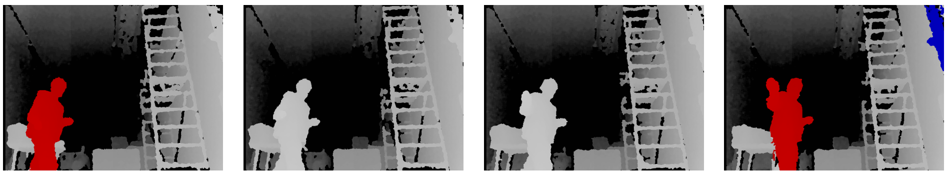
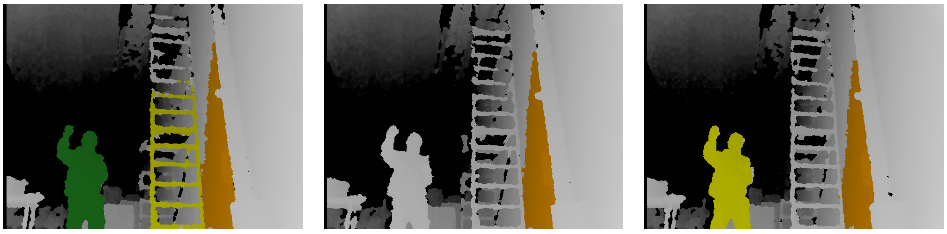
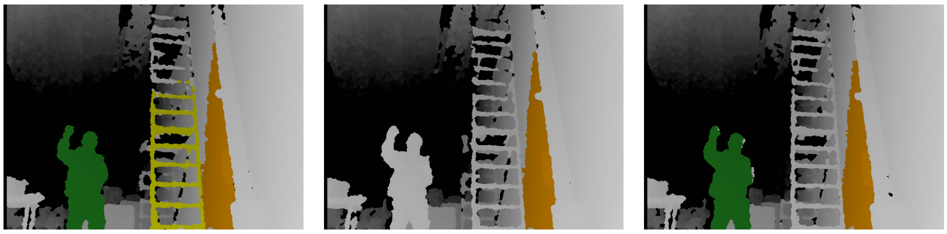

(a) Frame 194 (b) Frame 195 (c) Frame 196 (d) Frame 197

Figure 7.4.: Snapshots when enabling user relabeling



(a) Frame 608 (b) Frame 609 (c) Frame 610

Figure 7.5.: Snapshots when using the original OpenNI code (from the second data set)



(a) Frame 608 (b) Frame 608 (c) Frame 610

Figure 7.6.: Snapshots when enabling user relabeling (from the second data set)

7.2. User Detector Filter

Results for the user detector filter is split into three parts. First some results from the Optical Flow Filter are shown, then some results from the Feature Filter and at last some results of the final filter. The results are selected to show the general behaviour of the filters in addition to show some corner cases.

The results are visualized in the following way:

- Verified labels (labels which probably correspond to human users) are marked with solid colors.
- Falsified labels (labels which probably do *not* correspond to human users) are marked with horizontal colored lines.
- The background is a normalized version of the depth map image from OpenNI.

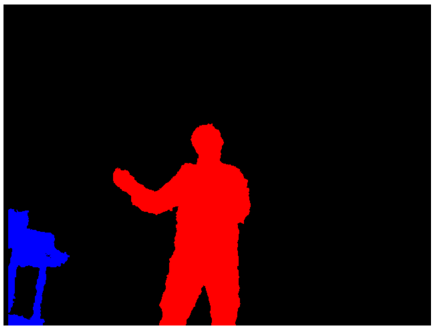
Minor differences in colors between users in images without depth background and the corresponding users in images with depth background are expected. This is because the label colors are adjusted to the depth of the image, so that parts of the label that are farther away are darker and closer parts are lighter.

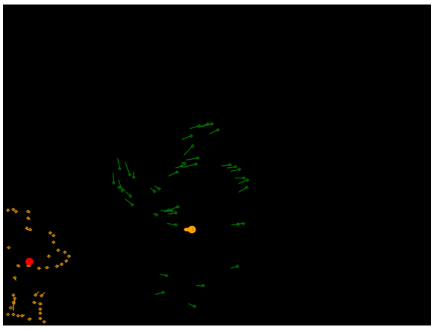
Note that in this section it will be made a distinction between “users”, which are any silhouettes detected by the Kinect’s algorithms, and “human users” which are the silhouettes which represent humans.

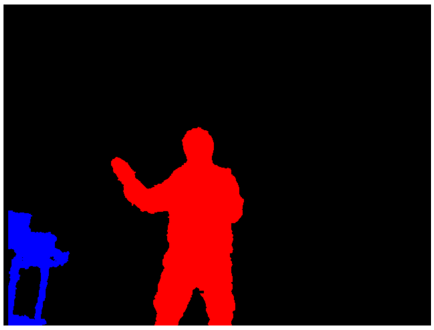
7.2.1. Optical Flow Filter

As described in Section 5.2 the Optical Flow Filter calculates movement of an object in two succeeding images. This section presents four images per snapshot. The leftmost images are the users in each of the frames (frame $n-1$ and frame n), and the upper right image is an illustration of the optical flow which is calculated between the two frames. The last image of each set of image, is the resulting image when filtering is finished in the format described above.

In this section, the circles with connected lines are referred to as arrows. These represent a change from the circle, to the end of the line.


(a) Frame 161


(c) Optical flow


(b) Frame 162

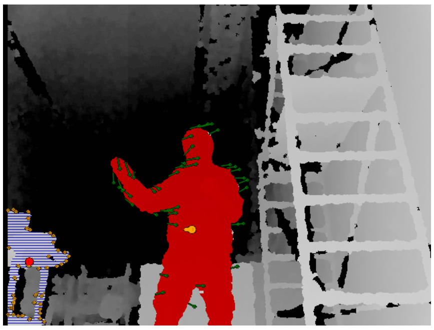
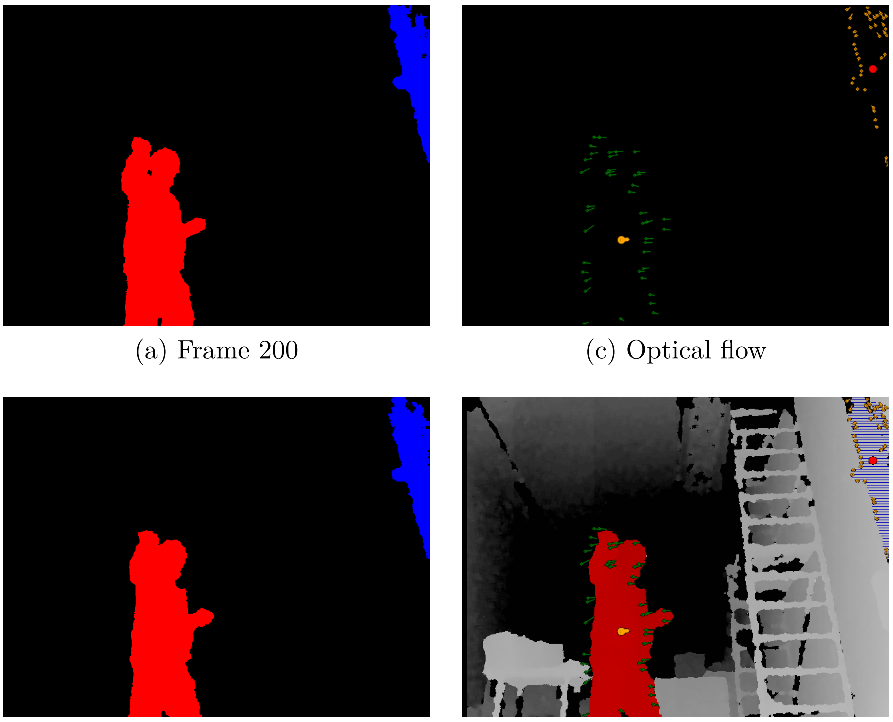

(d) Filtered Image

Figure 7.7.: Optical flow from certain points in Frame 161 is estimated by observing the differences in Frame 161 and Frame 162. The arrows in c) represent the velocity of each point. The larger circle with a thick line at the center of each suggested user represents the average velocity of that user. In d) one can see that the blue label is removed as it is filtered out by the algorithm.



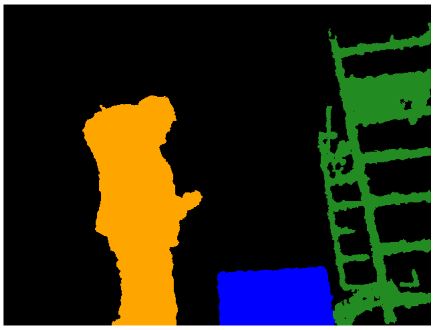
(a) Frame 200

(c) Optical flow

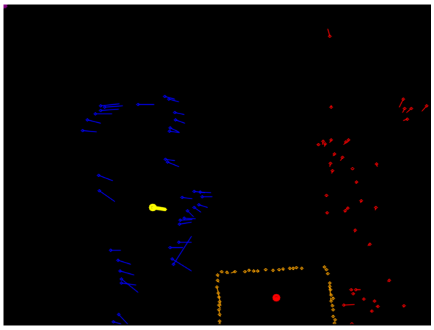
(b) Frame 201

(d) Filtered Image

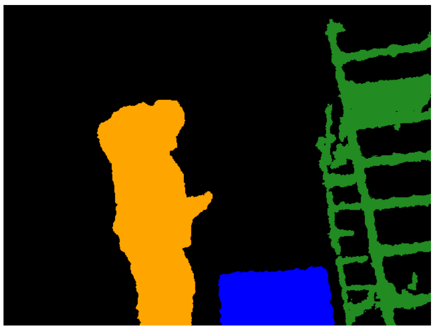
Figure 7.8.: Shows c) optical flow between the users in frames 200 and 201. In d) one can see that the blue label is removed as it is filtered out by the algorithm.



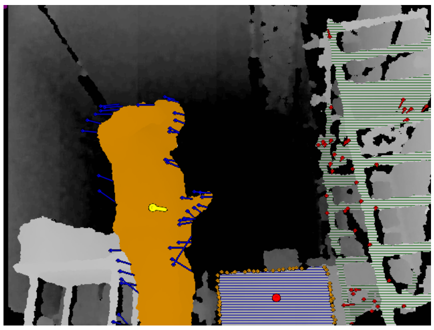
(a) Frame 329



(c) Optical flow

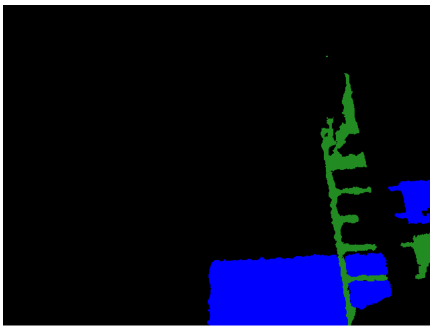


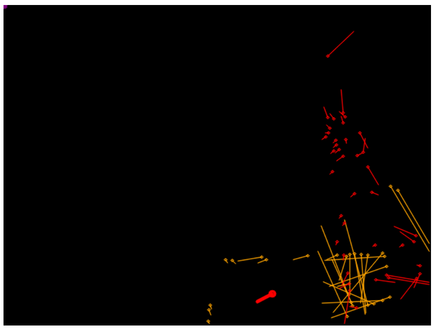
(b) Frame 330

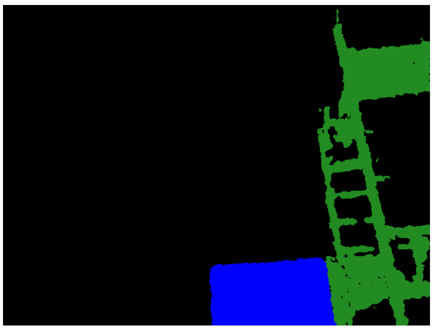


(d) Filtered Image

Figure 7.9.: Shows c) optical flow between the users in frames 328 and 329. In d) one can see that the green and blue labels are removed as they are filtered out by the algorithm.


(a) Frame 322


(c) Optical flow


(b) Frame 323

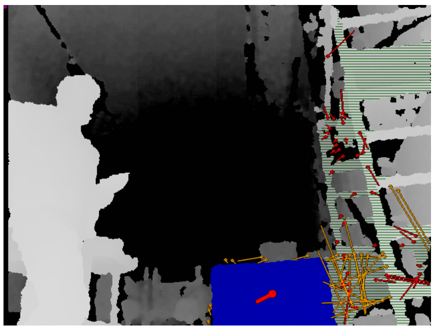
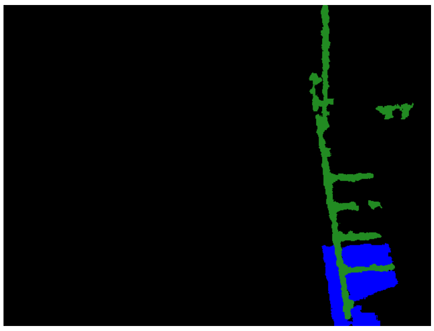
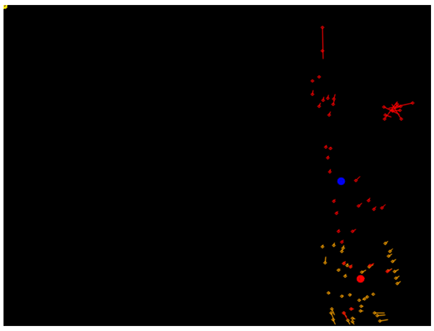
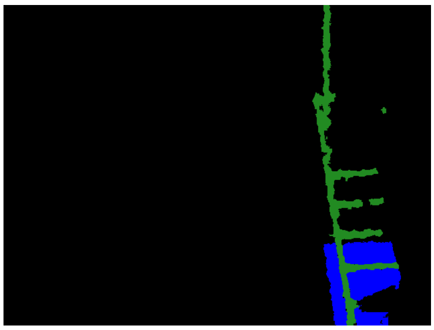

(d) Filtered Image

Figure 7.10.: Figure c) shows optical flow between the users in frames 322 and 323. However, notice that the blue area to the right of the green line in Frame 322 is marked as green in Frame 323. Because of this, some of the orange lines (which are associated with the blue figure) are very long. This is also the case for some of the red lines (which are associated with the green figure). Also notice that the blue figure is marked as a human user in d).


(a) Frame 286


(c) Optical flow


(b) Frame 287

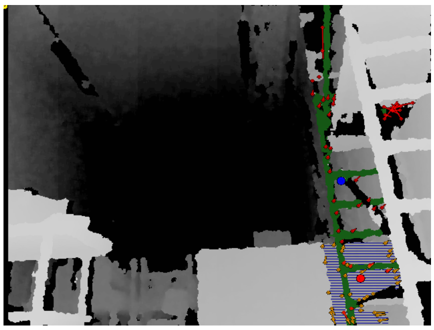
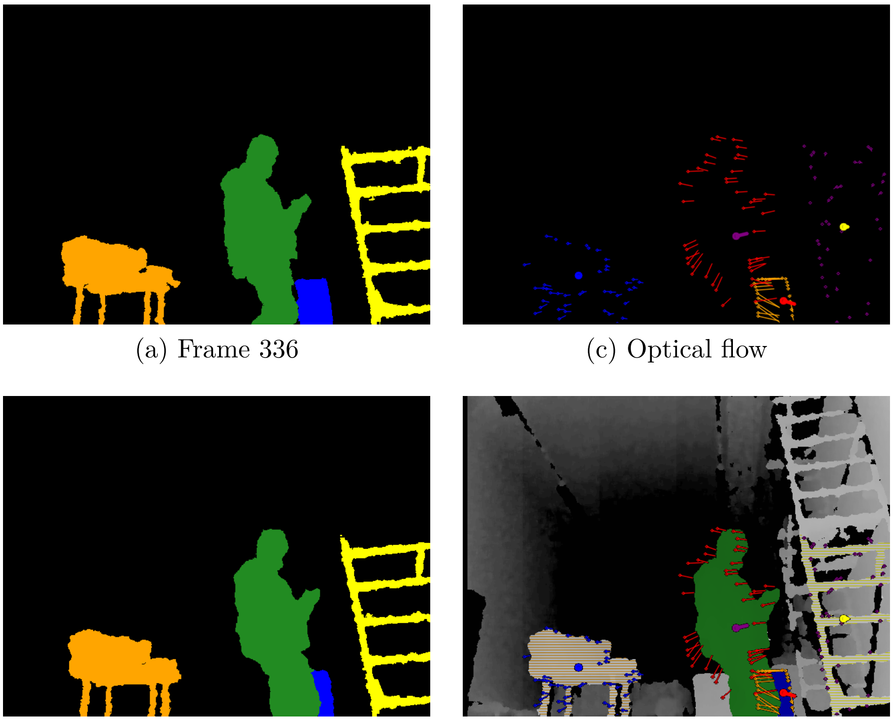

(d) Filtered Image

Figure 7.11.: Shows c) optical flow between the users in frames 286 and 287. In d) one can see that the blue label is correctly filtered out, but the green label is still visible.


(a) Frame 336

(c) Optical flow

(b) Frame 337

(d) Filtered Image

Figure 7.12.: Shows c) optical flow between the users in frames 336 and 337. In d) one can see that the orange and yellow labels are correctly hidden, but the blue label (at the bottom center) is still visible.

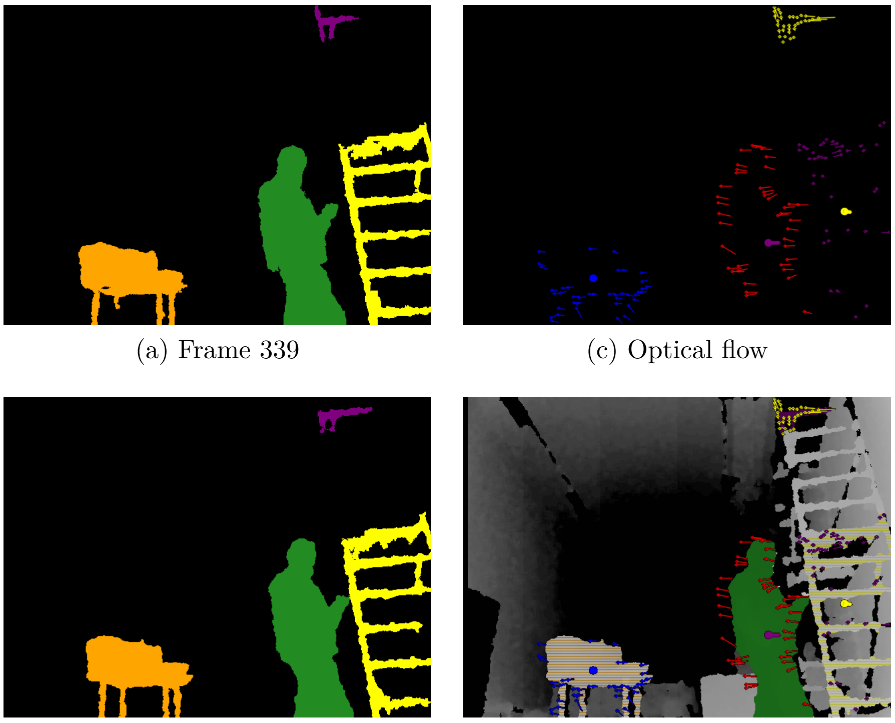

(a) Frame 339
(c) Optical flow
(b) Frame 340
(d) Filtered Image

Figure 7.13.: Shows c) optical flow between the users in frames 339 and 340. In d) one can see that the orange and yellow label are correctly hidden, but the purple label (in the upper right corner) is still visible.

7.2.2. Feature Filter

As the feature filter only operates on a single frame for each time step, only the filtered image is shown for each snapshot. For all frames the tuning variables are held constant, even between different data sets. Table 7.1 lists these values.

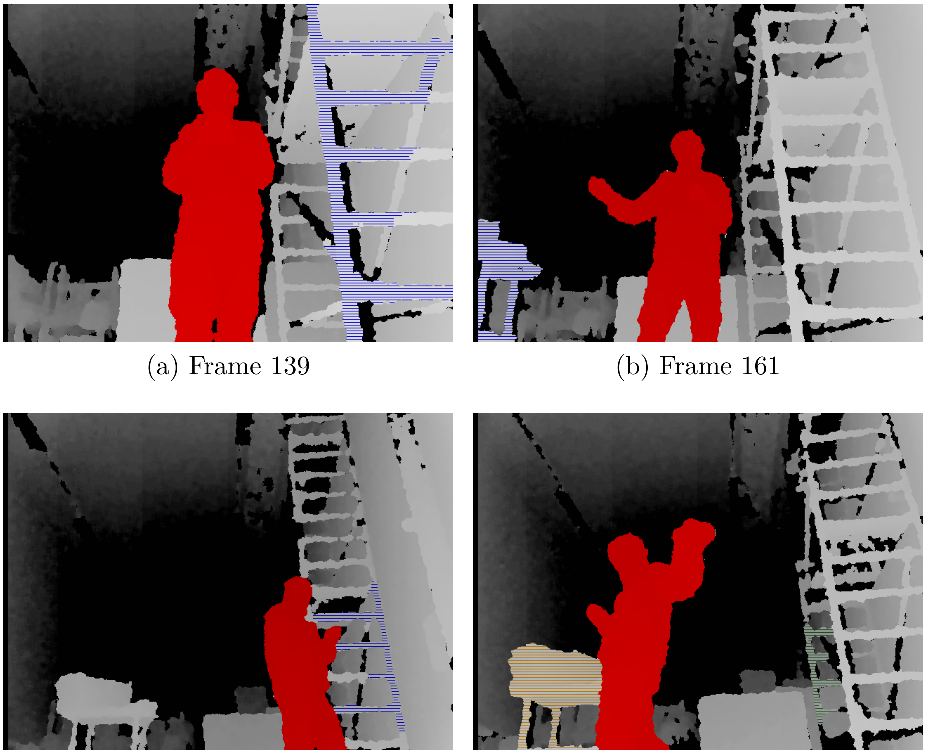
Property	Value
Vertical fill percentage	0.70
Horizontal fill percentage	0.65
Area	10000

Table 7.1.: Property values used by Feature Filter. The first two properties are dimensionless and the third property is measured in pixels.

All labels that have at least one feature which is below these constraints are marked as “false” users. These numbers are found by analysing the features of each label, using the average value and the standard deviation. For the main data set used in this project, the values are represented in Table 7.2: The values are somewhat less restrictive than what would seem optimal from this analysis only, because of analyzes performed on other data sets.

Label	Avg. Width %	Std.dev.	Height %	Std.dev.
1	0.94	0.05	0.86	0.04
2	0.81	0.12	0.54	0.21
3	0.80	0.12	0.42	0.14
4	N/A	N/A	N/A	N/A
5	0.81	0.17	0.78	0.11
6	0.94	0.01	0.45	0.03

Table 7.2.: Label 1 represents a human user, the other labels are various non-human objects. Label 4 did not appear in the data set, but is added for continuity.



(a) Frame 139

(b) Frame 161

(c) Frame 217

(d) Frame 267

Figure 7.14.: Shows the results when the feature filter is executed on frames 139, 161, 217 and 267 from the primary data set. The non-human labels are removed in all the frames.

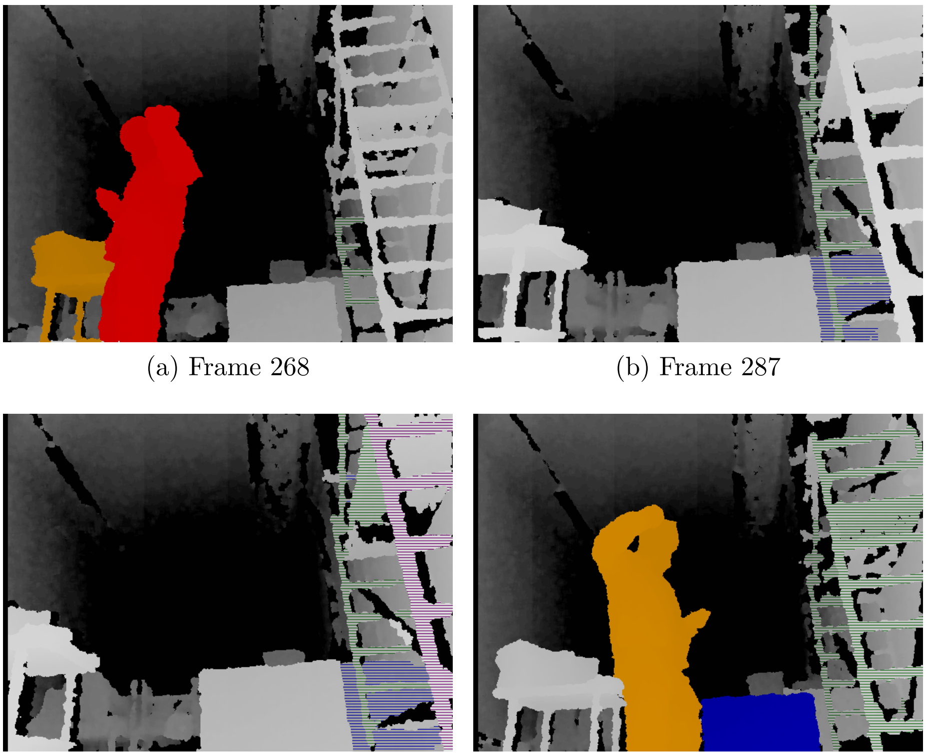
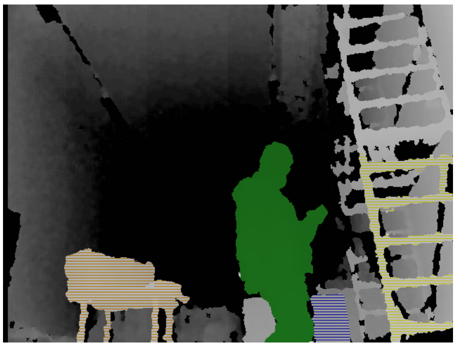
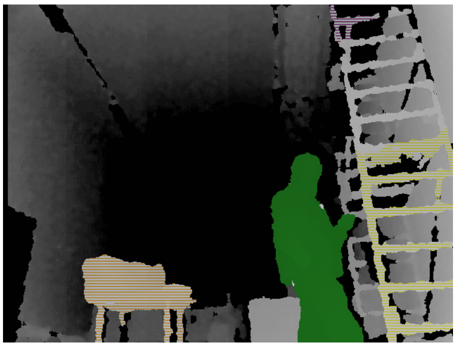
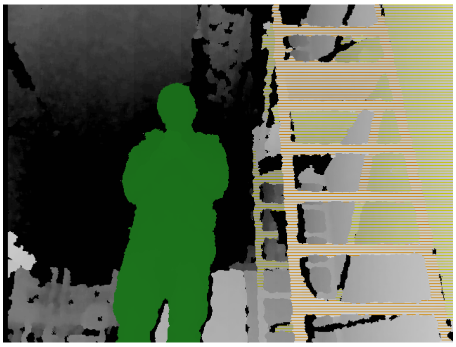

(a) Frame 268
(b) Frame 287
(c) Frame 301
(d) Frame 330

Figure 7.15.: Shows the results when the feature filter is executed on frames 268, 287, 301 and 330 from the primary data set. The non-human labels are removed correctly in frames 287 and 301, but in 268 and 330 the orange label and blue labels respectively are incorrectly verified.


(a) Frame 336


(b) Frame 339


(c) Frame 396 (2)

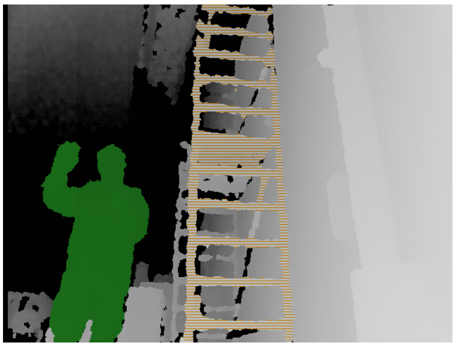
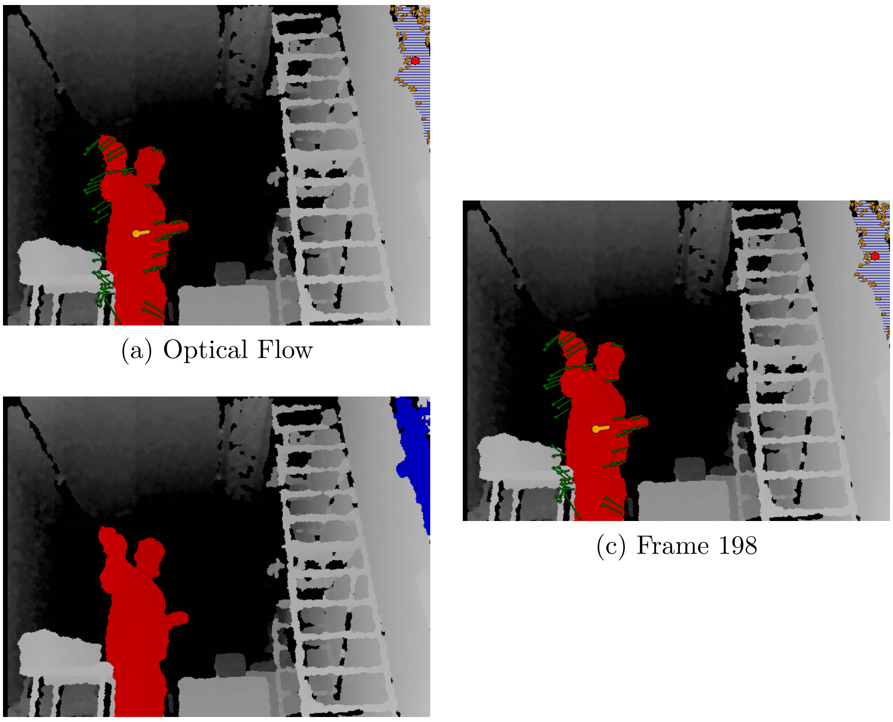

(d) Frame 467 (2)

Figure 7.16.: Shows the results when the feature filter is executed on frames 336 and 339 from the primary data set in addition to frames 396 and 467 from the second data set. The non-human labels are removed in all the frames.

7.2.3. Combined Filters

This section shows some examples on how the filters work when combined. The two images on the left are the results from each of the filters and the image on the right is the combined result. As expected, all labels that are verified in both frames, but only those labels, are verified after the voting phase.

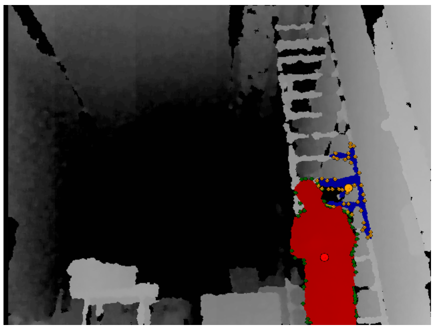


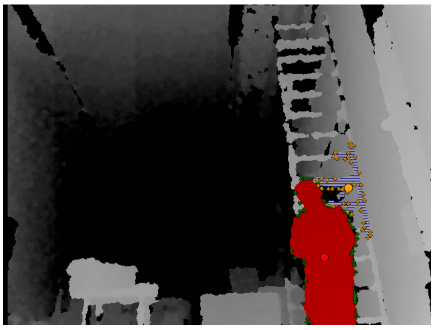
(a) Optical Flow

(b) Feature Filter

(c) Frame 198

Figure 7.17.: Frame 198: The blue label, incorrectly marked by the feature filter, is removed by the voter.


(a) Optical Flow


(c) Frame 242

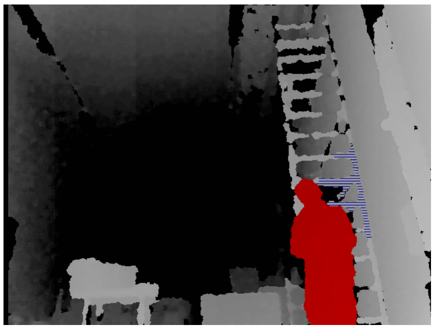
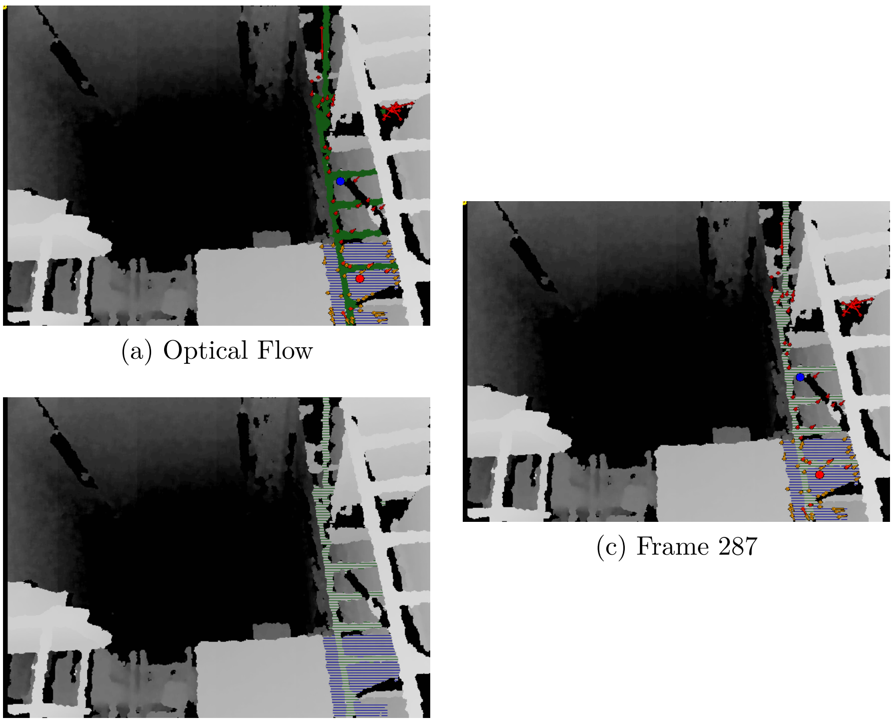

(b) Feature Filter

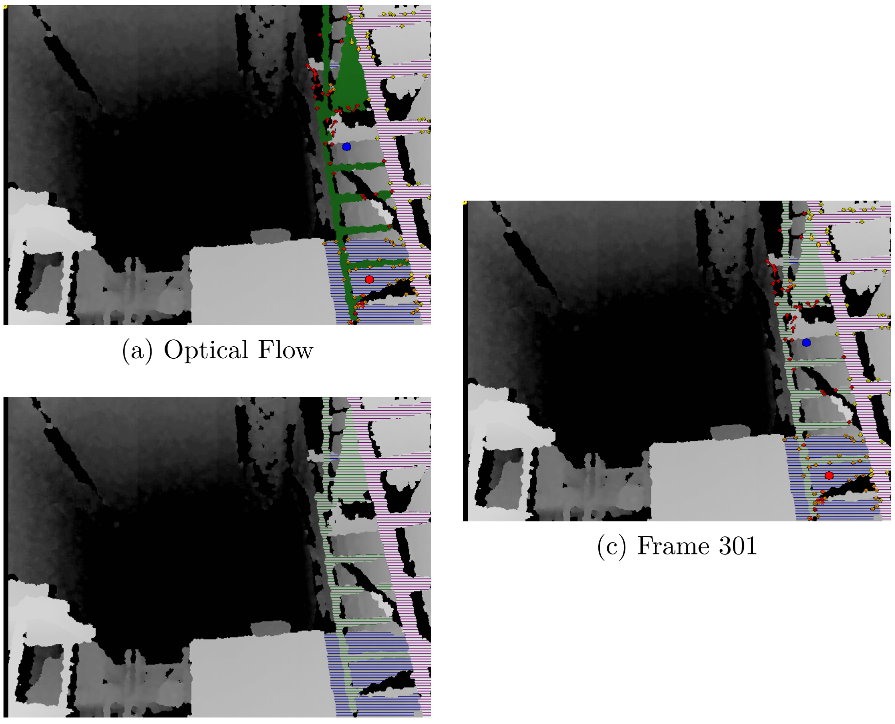
Figure 7.18.: Frame 242: The blue label, incorrectly marked by the optical flow filter, is removed by the voter.


(a) Optical Flow

(b) Feature Filter

(c) Frame 287

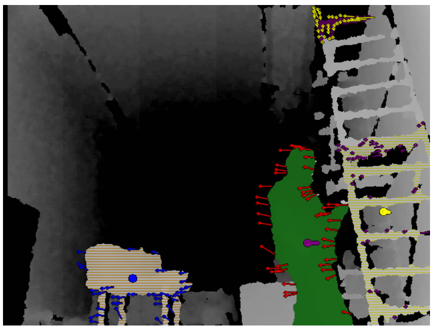
Figure 7.19.: Frame 287: The green label, incorrectly marked by the optical flow filter, is removed by the voter.

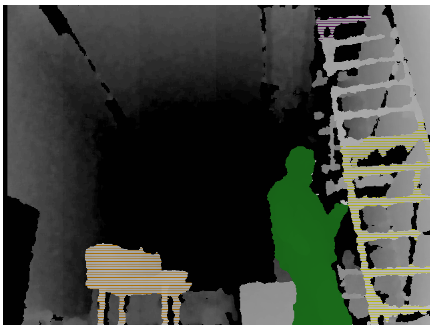

(a) Optical Flow

(b) Feature Filter

(c) Frame 301

Figure 7.20.: Frame 301: The green label, incorrectly marked by the optical flow filter, is removed by the voter.


(a) Optical Flow


(b) Feature Filter

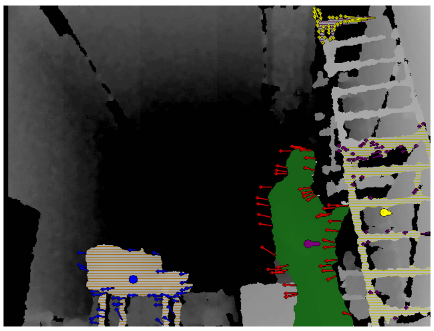
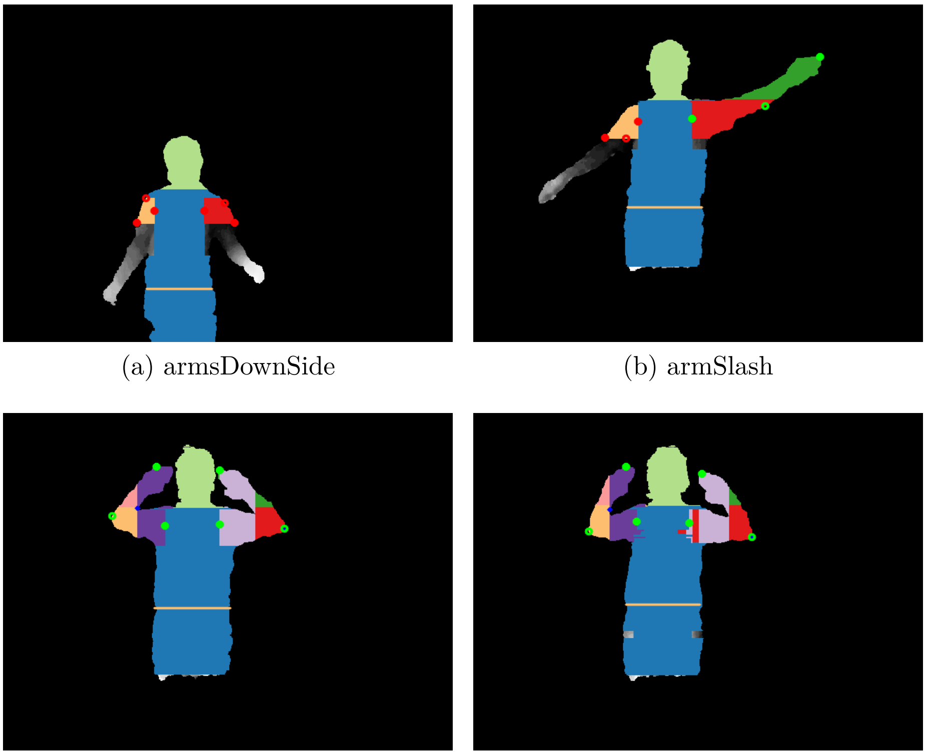

(c) Frame 340

Figure 7.21.: Frame 340: The purple label, incorrectly marked as valid by the optical flow filter, is removed by the voter.

7.3. Segmentation and Joint Recognition

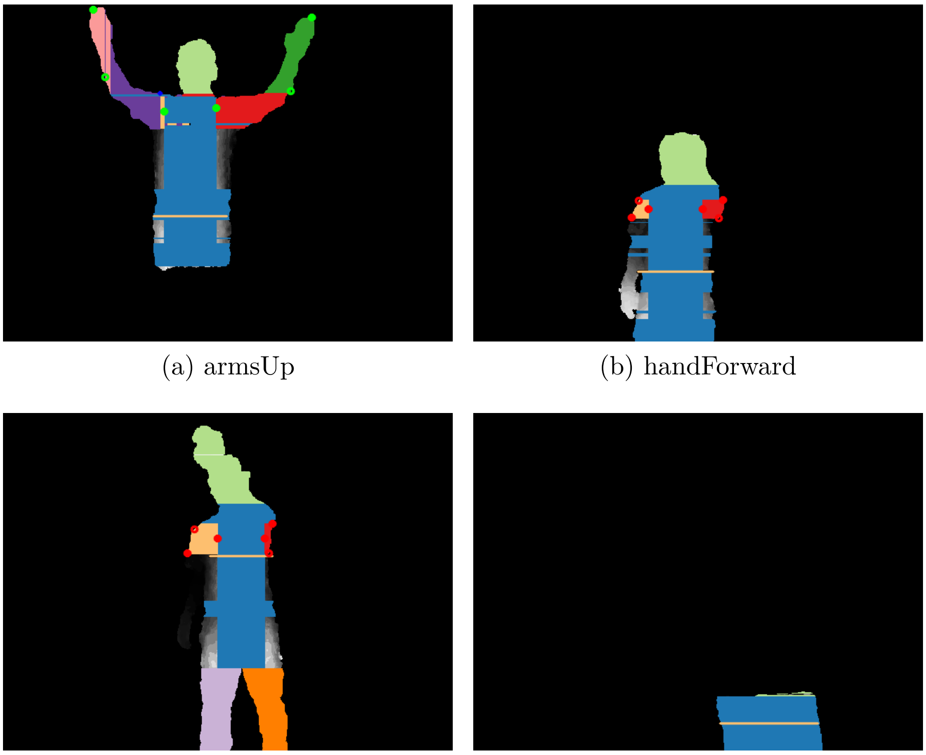
As the segmentation and labeling step assumes only a single human user and segmentation does not incorporate information about movement, a single frame is used in each example. Each image is the resulting image when segmentation is finished and joint detection has been executed. The result of the joint detection is visualized as circles which are green if the joints were accepted, and red if the joints were rejected. If no joints are shown, the algorithm did not execute the joint detection step because the segmentation was not sufficient.



(a) armsDownSide (b) armSlash

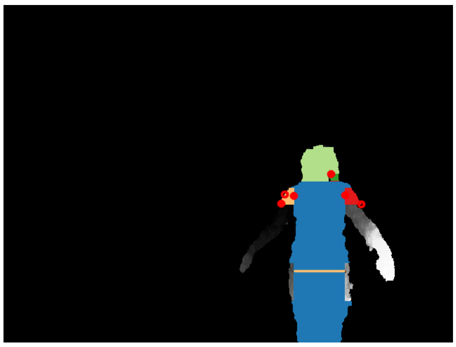
(c) armsTowardHead (d) armsTowardHead2

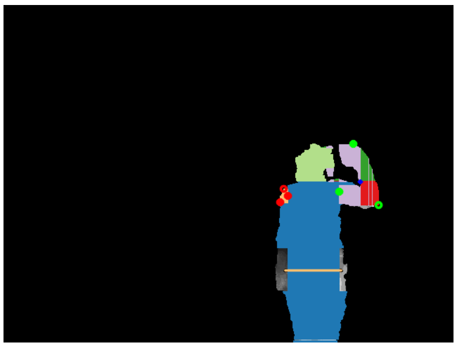
Figure 7.22.: Segmentation and joint detection on different poses. The pink and purple areas seen on c) and d) are where the arm is marked as double.

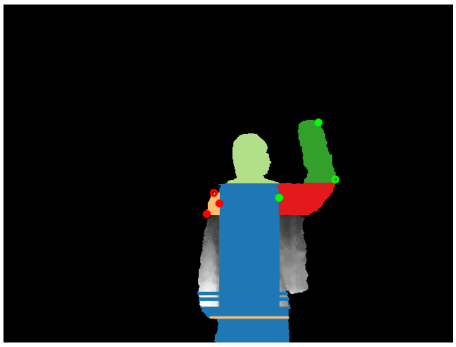

(a) armsUp (b) handForward

(c) hallForward (d) box

Figure 7.23.: Segmentation and joint detection is performed successfully in a). As the users in b) and c) do not have valid poses the joints are rejected (marked with red circles) and in d) the segmentation is not valid, as it is not a human, hence joint detection is not executed.


(a) user2ArmsOut


(b) user2Wave


(c) waveSweater

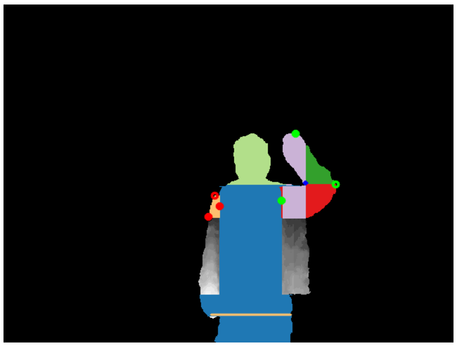
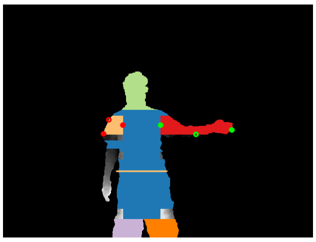
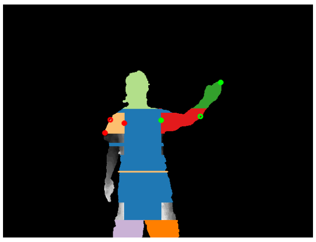

(d) waveSweater2

Figure 7.24.: Images a) and b) show the segmentation algorithm run on a user with different body structure, and c) and d) show segmentation of a user with a loose sweater. As with all the other images, no alteration of parameters was made to get these results.


(a) stopMotion1


(b) stopMotion2

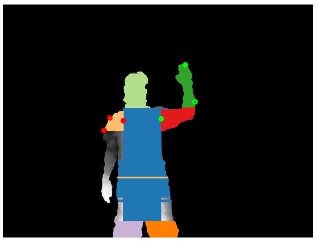
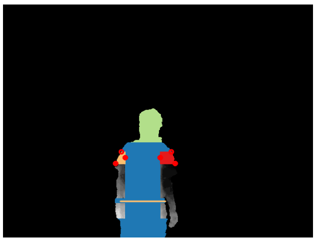

(c) stopMotion3

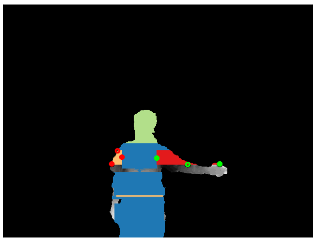
Figure 7.25.: These images show the execution of a "stop-motion" which is to be recognized by the gesture detector.

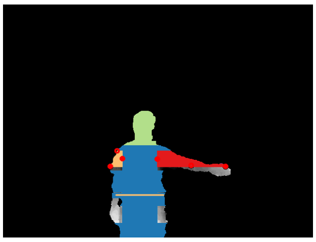
7.4. Gesture Detection

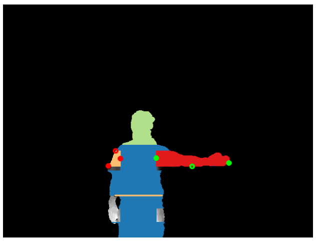
Results from the gesture detector are gathered in the following way: For each step where the state of the gesture detector changes, a snapshot at that point of time is taken, and the state which the detector was in before the change and the resulting state is recorded. Each figure represents a recognized gesture.

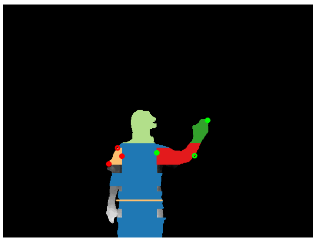
State transitions are encoded using a two-letter code. The first letter represents the user's left (L) or right (R) arm and the second the pose of the hand. The second letter of each code represents the state (U: Undefined, S: Straight, R: Raised, T: Towards head). For example does LU $\rightarrow$ LS represent a transition from the left arm being in an undefined state to the left arm being in a straight out state.

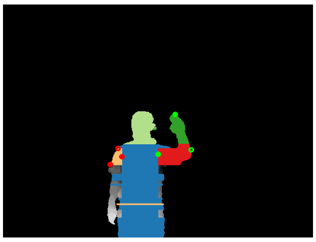

(a) Frame 0: LU

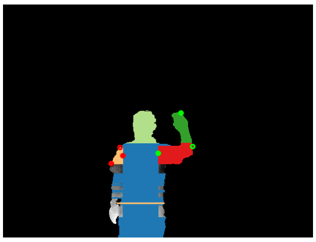

(b) Frame 27: LU → LS

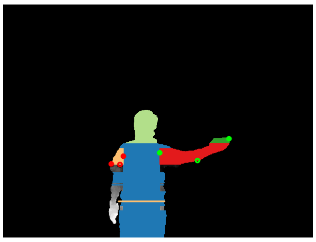

(c) Frame 31: LS → LU


(d) Frame 42: LU → LS


(e) Frame 50: LS → LR


(f) Frame 55: LR → LT


(g) Frame 69: LT → LR


(h) Frame 80: LR → LS

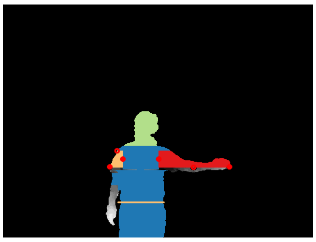
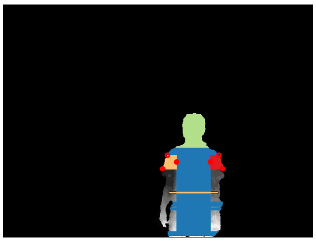
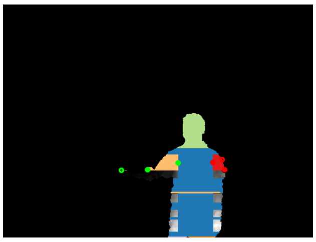
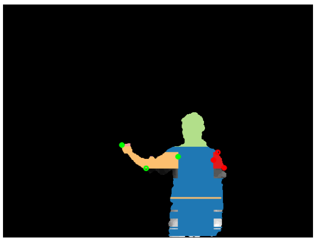
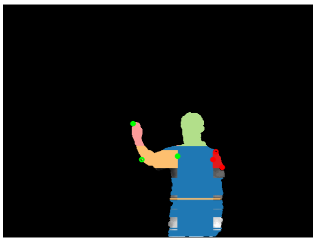

(i) Frame 94: LS → LU

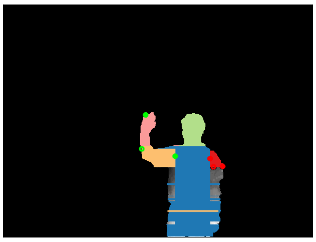
Figure 7.26.: These images show the state changes of the pose state machine when a movie of a person is used as input. The person raises the left arm to a straight out position, then lowers the arm, then raises the arm so it points towards the head.

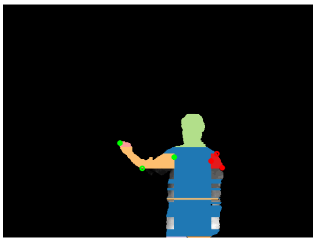

(a) Frame 253: RU, LU

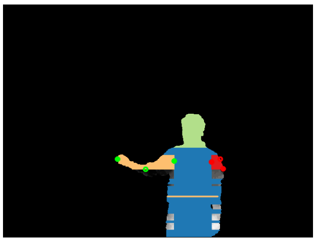

(b) Frame 253: RU → RS

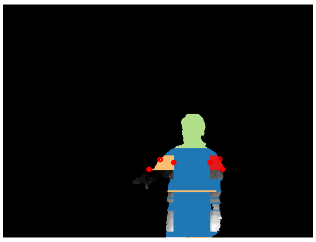

(c) Frame 255: RS


(d) Frame 257: RS → RR


(e) Frame 266: RR


(f) Frame 275: RR → RS


(g) Frame 276: RS


(h) Frame 278: RS → RU

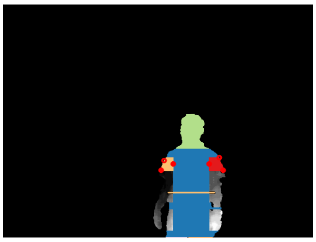

(i) Frame 287: RU

Figure 7.27.: These images show the state changes of the pose state machine when a movie of a person is used as input. The person raises the left arm to a straight out position, then lowers the arm, then raises the arm so it points towards the head.

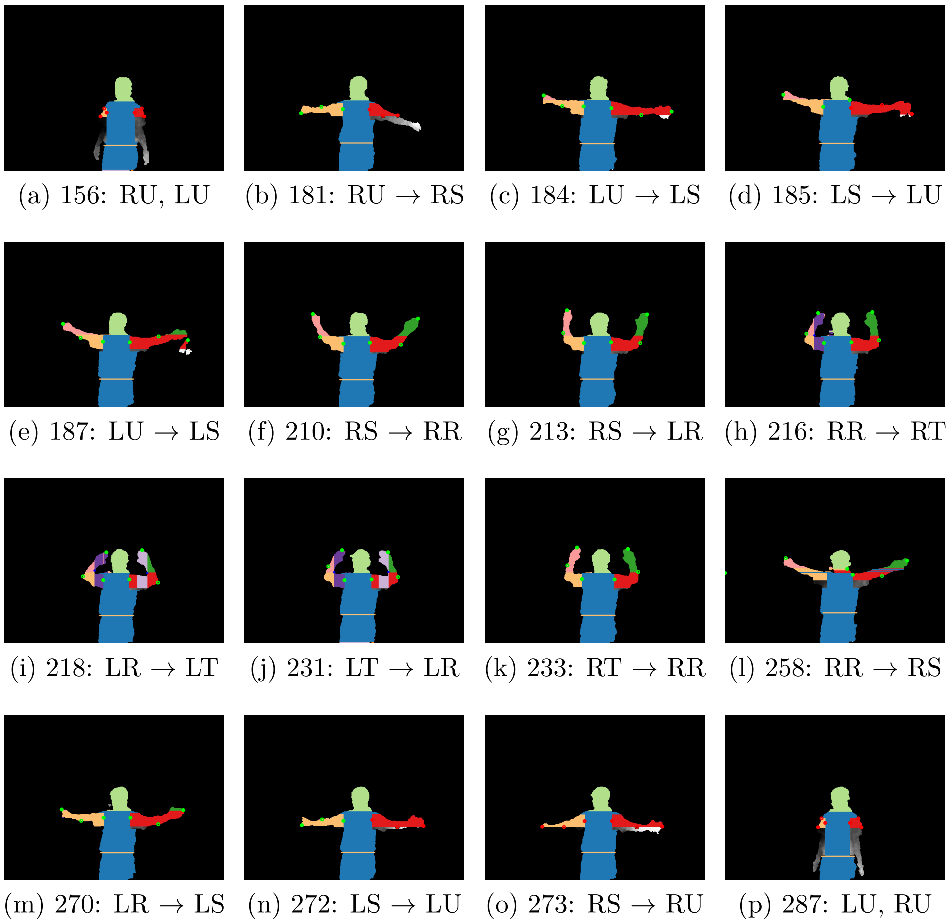

(a) 156: RU, LU (b) 181: RU → RS (c) 184: LU → LS (d) 185: LS → LU
(e) 187: LU → LS (f) 210: RS → RR (g) 213: RS → LR (h) 216: RR → RT
(i) 218: LR → LT (j) 231: LT → LR (k) 233: RT → RR (l) 258: RR → RS
(m) 270: LR → LS (n) 272: LS → LU (o) 273: RS → RU (p) 287: LU, RU

Figure 7.28.: A gesture is made by raising both arms, then holding them towards the head. At frames 185 (d) and 187 (e) a part of the surroundings was detected and incorrectly labeled by the OpenNI code. This resulted in a state change from the left arm being straight to it being undefined.

8. Discussion

In this chapter a discussion of the results produced in this project will be made. The layout is similar to that of Chapter 7, as each set of results will be discussed in its own section. In this chapter, some approaches that were attempted, but for various reasons dropped from the project, are also presented.

8.1. User Detector Filter

The user detector filter is a three-step process consisting of:

1. Processing an external event
2. Filtering
3. Voting and sending the resulting event

The first of these steps is to copy data from an external event, and storing a valid copy of these in buffer arrays. The data is then protected using C#'s *Monitor*-system, so that it is not overwritten while it is in use, even if new data is available. In addition to storing the data, the inconsistent user label problem is fixed by user relabeling.

When the data is ready and properly protected, filtering is initiated. The two filters are run in parallel, and this is properly handled when using the event-driven approach. The gain of running the filters in parallel is not very prominent, as the optical flow filter is much more time consuming than the feature filter. However, as more filters are added, the optical flow filter might still be the limiting factor with respect to time, hence the total execution time of the filters might be the same on a multi-core processor.

The voting process is also overly complicated with only two filters implemented. However, to support extendability and further work, the framework for selecting how the system should behave with other implemented filters has been developed.

8.1.1. User Relabeling

As can be seen in the differences between figures 7.1 and 7.2, 7.3 and 7.4, and 7.5 and 7.6 user relabeling was very successful. There are, however, some situations where this may not be the case as mentioned in Section 5.1. If the relabeling process is incorrect, the optical flow filter might calculate incorrect values as the calculation is done per label. Hence, if the image corresponding to one user, say to the left of an image, is exchanged with another user, say to the right of the consecutive image, the flow in the image might be observed as across the whole image. Because of this, more time could be spent calculating a perfect optimization if this is necessary, even though the solution used in this project works well on the input data collected. Such an approach is described in Appendix B.

8.1.2. Optical Flow Filter

Optical flow filtering worked well in most cases, as shown in figures 7.7, 7.8 and 7.9. However there are some examples where the results are somewhat unstable. The reason for this is that noise in the depth measurement and/or incorrect labeling results, as seen in Figure 7.10, have great impact on the results. In this example, a part of the blue label has been marked as green in the succeeding frame. This makes the algorithm believe that a part of the object moved from one side of the image to the other. Hence the algorithm detects that movement is much greater than it moves in the real world.

Another potential problem is the obstruction of a stationary object (S)

by a moving object (M). As M moves in front of S, S's silhouette deforms because of the obstruction. Because of this, the optical flow at S's perimeter is equal to M's obstruction, hence it seems that S is moving. However, as the rest of S is stationary, the average flow should be smaller than that of M. This effect is shown in Figure 7.12.

As the optical flow filter compares movement of the Kinect to that of the detected users, the movement of the Kinect must be known. An attempt to find this was made by using optical flow of the whole image, excluding detected users, and calculating the average movement at this point. This works rather well if the depth map of the background is complex, and the distance to the different objects changes as the Kinect moves. However, if the Kinect is placed in a corridor, and the ending wall is too far away to be observed, the distance to the walls of the corridor will move according to the Kinect. Because of this, the Kinect seems more or less stationary according to the depth map. Therefore the movement of the Kinect is simulated in this project, but if the Kinect is placed on a robot, movement can be found using odometry from the wheels or similar.

Another problem origins from the assumption that the user which moves the most has a high probability of being a human. This is true for the most part, but when no users are visible, another label tends to be validated, as can be seen in Figure 7.11. If more accurate data about the Kinect's movement is acquired, this problem might be improved.

With the test data collected for this project, the optical flow filter performs rather well, except for a considerable amount of false positives, mostly because of deformation of labels. However, optical flow might be the only way to distinguish objects with a shape similar to a human from other objects.

8.1.3. Feature Filter

When looking at the results from the feature filter, one can see that the difference in behavior of the two filters yields different weaknesses. As movement is ignored, and only a single frame is processed per time step, the amount of data needed to calculate the necessary information is lower. In addition, calculation of features is much faster than the optical flow calculation.

The reason for choosing the $AverageWidestHorizontalFillPercentage$ and the $AverageHighestVerticalFillPercentage$ is that these give a good indication of the density of the observed object. In addition, both features are easily calculated with low computational cost.

The feature filter performs very well on the data sets collected for this thesis. Although the features are very basic, they are calculated with low overhead and each feature is calculated with a single pass of the image. Most of the images only yield a single label, which is the correct one. There are some cases where this is not true, such as when a large piece of the wall is marked. The performance of the feature filter was surprisingly high, in terms of both calculation speed and quality of the results. However, there are certain limitations which are described in Section 8.3.1.

8.2. Gesture Detector

As mentioned earlier, the gesture detector is split into three parts: segmentation, joint detection and a pose state machine. These three parts must be run consecutively in the specified order as the second part uses the results from the first part and the third part the results from the second part. To simplify calculations, certain assumptions have been imposed; the silhouette must be of a human, the silhouette must be of high quality (low noise etc.), the user must be facing the Kinect to some extent and the user must not have a hand in front of the body - especially not the head.

In addition, the area where movement is allowed is somewhat restricted. Detection of arms held downwards is not implemented, but it should not be very difficult to extend the segmentation if future appliances require this.

8.2.1. Segmentation

To perfectly segment the human body is a difficult and time consuming task. As this is not needed for the joint recognition, effort has been put to only segment the parts of a body that are needed to recognize the arm poses. This is of course the arms, but to detect the arms other parts such as the legs if they exist, head, torso are also segmented.

When these assumptions have been fulfilled, the segmentation has worked fairly well on the data recorded in this project. Although there are some cases, such as when depth detection has failed at a small area on a user's chest, or that other objects have been detected as a part of the body, where segmentation has not worked properly, a very high percentage of the images with a user have been sufficiently segmented to detect a user's joints.

As the aim of segmentation has been joint detection and not perfect segmentation, some mislabeling, such as marking the whole right arm as the right shoulder, has been accepted as a "correct" answer. In addition, not labeling the arms when they point downwards or the sides of the torso has been treated as correct. The only problems with the segmentation process comes if noise at the very top of the head is present, or if the silhouette from OpenNI is not as good as assumed. If noise is present, the segmentation step tends to mark this as one of the arms which may resolve in a wrongly detected hand or elbow. If the silhouette is of poor quality, such as in Figure 7.28 e), the figure might label a part of an area that is not the user's arm with one of the arm-labels. This might cause problems in the joint detection step.

8.2.2. Joint Detector

To recognize a pose, the pose state machine only needs four variables per arm: the shoulder's y-coordinates , the elbow's x-coordinate and the hand's x and y coordinates. In the results presented in Section 7.3, one can see that where the arm configurations are within the valid limits, recognition of all three joints are fairly precise. Where the arm configuration is invalid, the joints are marked as rejected in every case.

The only observed problem with the joint detector is when the segmentation process results in a part of the head marked as the arms described above. In an early version of the segmentation algorithm the torso could be marked as very thin, which marked most of the body as arms. This lead to problems when detecting the elbow which uses the point the most distant to the hand and shoulder. To improve this problem, the lines which are labeled as too wide can no longer be split so that the line representing the torso is shorter than a certain length.

8.2.3. Pose State Machine

The pose state machine also yields near-perfect results on the data sets. However, there is a problem when detecting a gesture where the arm is held straight out to the side. If the straight out gesture is the current state and joint detection fails for a single frame, for example if OpenNI loses track of the user, the state will be changed to Undefined. As this might lead to a straight arm gesture being detected multiple times, gestures using only the straight arm pose should not be used. However, if a straight arm pose is used in combination with a raised arm pose or a towards head pose, the event could be set to trigger when the raised/towards pose is recognized. If this is done, one could ensure that the raised arm is correctly recognized.

8.3. Limitations

Although the current implementation works well, additional testing and optimization should be made before the system is ready to be used in a real world scenario. In addition, the choice of sensor imposes limitations on which environments the system is suitable for and on which conditions that must apply for gestures to be successfully recognized.

8.3.1. Limited Testing Database

The main limitation of the developed system is the limited size of the testing database. Due to lack of time and resources, only a few users in a limited set of environments were used to calibrate and identify the filters.

This leads to a problem with the feature filter, as it depends on detecting features which can separate a human from other objects, and features that yield the best results for one data set do not necessarily work for other data sets. In addition, the segmentation step is also dependent on certain features, such as the width of the head compared to the width of the body. Although the current configuration worked well amongst all users, segmenting users with very different physique might not work properly.

To conquer these limitations, a large database of users with a wide span of body builds, in various environments must be collected and the configuration must be optimized.

8.3.2. Execution Speed

Another aspect that must be optimized is the execution speed of the algorithm, especially the optical flow filter. This filter is the one that has the highest execution time per frame, and when it is turned off, a sudden increase in the processed frames per second is detected. On a 2.4 GHz Intel Core 2 Quad processor, the increase is from about 13fps to about 28fps when turning off the optical flow filter.

Although some optimization has been made, described in Appendix A, there are several approaches which could improve execution speed, such as scaling the depth images to a smaller size, thus reducing execution time, but also the quality of the results. Another approach is to store the points detected by *cvGoodFeaturesToTrack*, update these points using *OpticalFlow.PyrLK* and use the updated points in the next frame instead of running *cvGoodFeaturesToTrack* for each frame. With this approach, new points are only detected if there is less than a certain number of valid points left. The number of points decreases as *OpticalFlow.PyrLK* can not find the optical flow for every point.

8.3.3. Hardware Limitations

As none of the hardware limitations are improved by this system, the hardware limitations from Section 3.3 apply for the improved system as well. This means that the system might perform poorly in sunlight, when observing reflective or transparent surfaces, and that gestures which use finger configurations can not be detected if the whole body is observed.

8.3.4. Pose State Machine Transitions

The last limitation is that the pose state machine can not change states in any other sequence than described in Section 6.3. This may be a problem if the user is not detected when the gesture is started. Even though the person has an arm raised, the gesture is not recognized as the state machine is in the $Undefined$ state. This is implemented to improve robustness, as the probability of a non-human silhouette being detected as first having an arm straight out and then an arm raised, is significantly lower than just detecting a single frame which could be a person with a raised arm.

Another problem due to forced state change sequence, is that if the raised arm gesture or the towards head gesture is made and the user either moves out of the Kinect's view or OpenNI's tracking is lost, the system will stay in this state until the program is terminated. This can be solved by adding a timeout - if a gesture is undetected for n frames, the gesture is reset.

9. Conclusion

The thesis addresses the problems that arise when using the Kinect on a non-stationary platform, in addition to the problem of long initialization time when a new user is detected.

The results from Chapter 7 clearly show that the developed system improves the results from the closed source middleware. The improvement of the relabeling algorithm yields optimal results regarding correct tracking of the human users, even though this is not guaranteed by the algorithm. The implemented approach is very effective and the stated problems no longer occur in the captured data sets.

The false-positive users problem has also been minimized. Although there are certain parts of the captured data where this problem still can be observed, it is typically only in one or two frames in succession. Except for this, the only detected users are the human users, hence the true positives.

Furthermore, the gesture recognizer, which segments a user, detects his or her joints and recognizes gestures using these, is an efficient approach to the gesture detection problem. Due to the precautions taken, the gesture recognizer has a very low frequency of false-positives, even if the data from the earlier stages have some errors. This makes it very robust, which is vital when operating alongside humans.

This thesis presents an original approach for introducing the Kinect sensor to the world of social robotics, through gesture recognition. As gestures now can be recognized, even when the Kinect moves, and without the initialization delay in the Kinect's algorithms, users can give orders to a robot without having to access a computer or operation panel.

9.1. Further Work

Although this thesis presents a promising approach to social robotics, some further development is needed before the system can be used in a real-life setting. The main focus of future development should be to acquire a larger database with different environments and users, as suggested in Section 8.3.1.

A node should also be created to register movement, for example using ROS interface. Such a node was partly implemented using the “rosnode” interface and C#’s ability to load libraries, but due to limited time it was not prioritized for this thesis.

Optimization of the relabeling process could also be improved, and a detailed description of two ideas for such implementations is presented in Appendix B.

In addition, one could imagine a more direct approach for the feature filters. Instead of just comparing certain features of a user with expected results, one could match a user to a template. However, due to the flexibility of the human body and the vast complexity of different poses, it might become very difficult and time-consuming to create templates that would work in every possible scenario.

References

[1] Adafruit. The Open Kinect project – THE OK PRIZE. Website, 2010. http://adafruit.com/blog/2010/11/04/the-open-kinect-project-the-ok-prize-get-1000-bounty-for-kinect-for-xbox-360-open-source-drivers/ Aquired: 13:49 08.04.2011.

[2] Adafruit. Open Kinect driver(s) released. Website, 2010. http://adafruit.com/blog/2010/11/10/we-have-a-winner-open-kinect-drivers-released-winner-will-use-3k-for-more-hacking-plus-an-additional-2k-goes-to-the-eff/ Aquired: 15:21 08.04.2011.

[3] amazon.com. Kinect Sensor with Kinect Adventures! http://www.amazon.com/Kinect-Sensor-Adventures-Xbox-360/dp/B002BSA298/, 2010. Aquired: 19:55 05.04.2011.

[4] Jean-Yves Bouguet. Pyramidal implementation of the lucas kanade feature tracker description of the algorithm, 2000. URL http://robots.stanford.edu/cs223b04/algo_tracking.pdf.

[5] S.Y. Chen, Y.F. Li, and Jianwei Zhang. Vision Processing for Real-time 3-D Data Acquisition Based on Coded Structured Light. *Image Processing, IEEE Transactions on*, 17(2):167 –176, February 2008.

[6] R. Cutler and M. Turk. View-based interpretation of real-time optical flow for gesture recognition. In *Automatic Face and Gesture Recognition, 1998. Proceedings. Third IEEE International Conference on*, pages 416 –421, April 1998.

[7] Emgu CV. Emgu CV: OpenCV in .NET (C#, VB, C++ and more). Website, 2011. http://emgu.com Aquired: 16:00 06.05.2011.

[8] J. Davis and M. Shah. Visual gesture recognition. *Vision, Image and Signal Processing, IEE Proceedings -*, 141(2):101 –106, April 1994.

[9] Jr. Forney, G.D. The viterbi algorithm. *Proceedings of the IEEE*, 61 (3):268 – 278, march 1973. ISSN 0018-9219.

[10] GameStop. Kinect for Xbox 360. `http://www.gamestop.com/xbox-360/accessories/kinect-for-xbox-360-with-kinect-adventures/90774`, 2010. Aquired: 09:49 11.04.2011.

[11] S. Goto and F. Yamasaki. Integration of percussion robots "robot-music" with the data-suit "bodysuit": Technological aspects and concepts. In *Robot and Human interactive Communication, 2007. RO-MAN 2007. The 16th IEEE International Symposium on*, pages 775 –779, August 2007.

[12] M. Hasanuzzaman, V. Ampornaramveth, Tao Zhang, M.A. Bhuiyan, Y. Shirai, and H. Ueno. Real-time vision-based gesture recognition for human robot interaction. In *Robotics and Biomimetics, 2004. ROBIO 2004. IEEE International Conference on*, pages 413 –418, August 2004.

[13] En Wei Huang and Li Chen Fu. Gesture stroke recognition using computer vision and linear accelerometer. In *Automatic Face Gesture Recognition, 2008. FG '08. 8th IEEE International Conference on*, pages 1 –6, September 2008.

[14] Thomas S. Huang and Vladimir I. Pavlovic. Hand gesture modeling, analysis, and synthesis. In *In Proc. of IEEE International Workshop on Automatic Face and Gesture Recognition*, pages 73–79, 1995.

[15] IFixIt. `http://www.ifixit.com/Guide/Image/meta/dcGosZx6dEwevBXt`, 2010. Aquired: 22:50 05.04.2011.

[16] Sacha Krakowiak. What is Middleware. Website, 2003. `http://middleware.objectweb.org/` Aquired: 13:50 06.05.2011.

[17] J.J. Kuch and T.S. Huang. Vision based hand modeling and tracking for virtual teleconferencing and telecollaboration. In *Computer*

Vision, 1995. Proceedings., Fifth International Conference on, pages 666 –671, June 1995.

[18] Logitech. Logitech webcam c200. `http://www.logitech.com/en-us/webcam-communications/webcams/devices/5865`, February 2011. Aquired: 12:50 08.02.2011.

[19] Logitech. Logitech hd pro webcam c910. `http://www.logitech.com/en-us/webcam-communications/webcams/devices/6816`, February 2011. Aquired: 12:49 08.02.2011.

[20] Jani Mäntyjärvi, Juha Kela, Panu Korpipää, and Sanna Kallio. Enabling fast and effortless customisation in accelerometer based gesture interaction. In *Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia*, MUM '04, pages 25–31, New York, NY, USA, 2004. ACM.

[21] S. Mitra and T. Acharya. Gesture recognition: A survey. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 37(3):311 –324, May 2007.

[22] K. Morrow, C. Docan, G. Burdea, and A. Merians. Low-cost virtual rehabilitation of the hand for patients post-stroke. In *Virtual Rehabilitation, 2006 International Workshop on*, pages 6 –10, April 2006.

[23] National Association of the Deaf. What is american sign language? `http://www.nad.org/issues/american-sign-language/what-is-asl`, February 2011. Aquired: 22:52 08.02.2011.

[24] Kenji Oka, Yoichi Sato, and Hideki Koike. Real-time fingertip tracking and gesture recognition. *IEEE Computer Graphics and Applications*, 22:64–71, 2002. ISSN 0272-1716.

[25] "OpenCv". Motion analysis and object tracking. Website, 2010. `http://opencv.willowgarage.com/documentation/cpp/motion_analysis_and_object_tracking.html` Aquired: 19:54 06.06.2011.

[26] OpenKinect. Hardware Information. Website, 2011. `http://openkinect.org/wiki/Hardware_info` Aquired: 19:15 13.04.2011.

[27] OpenKinect. the OpenKinect project. Website, 2011. `http://www.openkinect.org` Aquired: 11:32 13.06.2011.

[28] OpenNI. Introducing OpenNI. Website, 2011. `http://www.openni.org` Aquired: 15:24 13.04.2011.

[29] Play.com. "kinect including kinect: Adventures!". Website, 2011. `http://www.play.com/Games/Xbox360/4-/10296372/Kinect-Including-Kinect-Adventures-/Product.html#TechnicalDetailsTab` Aquired: 12:55 19.05.2011.

[30] Vitruvius Pollio. *The Ten Books on Architecture - Book III*, chapter 1. Project Gutenberg, 2006.

[31] PrimeSense. Primesensor reference design. `http://www.primesense.com/?p=514`, February 2011. Aquired: 22:00 08.02.2011.

[32] D.L. Quam. Gesture recognition with a dataglove. In *Aerospace and Electronics Conference, 1990. NAECON 1990., Proceedings of the IEEE 1990 National*, pages 755 –760 vol.2, May 1990.

[33] Guinness World Records. Fastest-selling consumer electronics device. Website, 2011. `http://www.guinnessworldrecords.com/Search/Details/Fastest-selling-consumer-electronics-device/74941.htm` Aquired: 13:37 08.04.2011.

[34] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Pearson Education, second edition edition, 2003.

[35] Daniel Scharstein and Richard Szeliski. A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *International Journal of Computer Vision*, 47:7–42, 2002. ISSN 0920-5691.

[36] Jianbo Shi and C. Tomasi. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*, pages 593 –600, June 1994.

[37] Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake. Real-Time Human pose Recognition in Parts from Single Depth Images. In *Computer Vision and Pattern Recognition*. Microsoft Research Cambridge & Xbox Incubation, IEEE, June (to appear) 2011.

[38] Tomoichi Takahashi and Fumio Kishino. Hand gesture coding based on experiments using a hand gesture interface device. *SIGCHI Bull.*, 23:67–74, March 1991.

[39] K.N. Tarchanidis and J.N. Lygouras. Data glove with a force sensor. *Instrumentation and Measurement, IEEE Transactions on*, 52(3):984 – 989, June 2003.

[40] Carnegie Mellon University. ”cmu graphics lab motion capture database”. Website, 2002. `http://mocap.cs.cmu.edu/` Aquired: 13:19 19.05.2011.

[41] Lloyd R. Welch. Hidden markov models and the baum-welch algorithm. *IEEE Information Theory Society Newsletter*, 53(4), December 2003.

[42] Ying Wu and Thomas Huang. Vision-Based Gesture Recognition: A Review. In Annelies Braffort, Rachid Gherbi, Sylvie Gibet, Daniel Teil, and James Richardson, editors, *Gesture-Based Communication in Human-Computer Interaction*, volume 1739 of *Lecture Notes in Computer Science*, pages 103–115. Springer Berlin / Heidelberg, 1999.

[43] Øystein Skotheim. Kinect sensor - preliminary study, May 2011.

[44] X. Zabulis, H. Baltzakis, and A. Argyros. Vision-based Hand Gesture Recognition for Human-Computer Interaction, 2009.

Appendices

A. Optimizations

A few measures have been taken to improve the performance of the system, some of these are mentioned below:

- Using ROI in images
- Moving conditions outside loops and duplicating code
- Extracting limits outside loops

By using the ROI in images, only certain parts of the image are processed by both OpenCV algorithms and the implemented algorithms for this thesis. This is done by registering the leftmost, rightmost, topmost and bottommost points of the object of interest at the first iteration of the image, and results in improved execution time.

Some conditions have a high cost of calculating, and some places in the code the test for the condition is moved outside the loop. In terms of pseudo-code this would result in:

```
for each row:
  for each column
   if Test():
   //Do something
   else:
   //Do something else
```

being changed to:

```
if Test():
  for each row:
    for each column
```

```
      //Do something
else:
  for each row:
    for each column
      //Do something else
```

This results in `Test()` being executed once instead of *number of rows* times *number of column* times.

When profiling the algorithm, using the open source SlimTune profiler, it was shown that an alarming amount of time was used on a call to the OpenNI wrapper. A very high percentage of the total execution time was spent on getting the height of the depth image. Because of this, the limits in loops were moved outside the loop in the following way:

```
for (x=0; x < image.getWidth(); x++)
  for (y=0; y < image.getHeight(); y++)
    //Do something
```

was changed to:

```
imageHeight=image.getHeight()
imageWidth=image.getWidth()
for (x=0; x < imageWidth; x++)
  for (y=0; y < imageHeight; y++)
    //Do something
```

The reason for the inefficiency is most likely that the calls to the wrapper (such as image.Height), demands exclusive access to the wrapper. Due to this, the image must acquire a lock *width* times *height* for the *getHeight()* call, and *width* times for the *getWidth()* call. With the optimized code, each call is made once per loop.

B. Relabeling Improvement

To improve the relabeling algorithm the following two measures can be taken:

- Using for example maximum flow to map optimally
- Using other features than the centroid, such as shape or area.

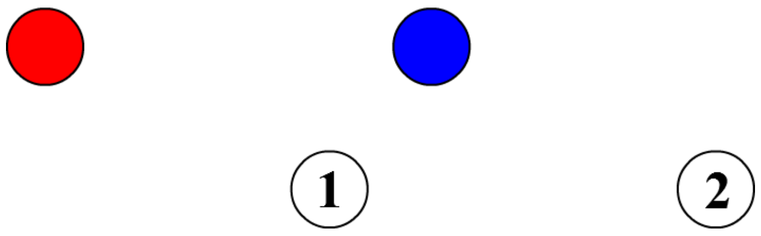


Figure B.1.: The circles marked as 1 and 2 are centroids of the current users, and the blue and red circles are the previously detected red and blue labels.

Consider the scenario in Figure B.1. With the current implementation, the figure marked as 1 will be labeled as blue, as it is closest to the blue circle, while the figure marked as 2 will be labeled as a random color (chosen by OpenNI). An example of this scenario would be if the Kinect was turned quickly to the the left. The optimal result could then be found using a max-flow algorithm, by adding a source super-node with edges to each of the current (numbered) centroids, and a sink super-node with edges to all of the previous (colored) centroids.

However, if instead this scenario was the result of the Kinect turning slightly to the right and a new object was detected, the correct mapping would be to map the blue label to object 1 and a new label to 2. To be able to distinguish between these cases, one could implement features to recognize if the object is previously detected. Features could for example consist of area, density and circumference, and certain criteria could be set for a match to be accepted.

When considering such a complex approach, one should keep in mind that the algorithm is executed in every frame, and that all information about users passes through the relabeling process. Because all later steps depend on the results from the relabeling, the process can easily become a bottleneck that slows down the total execution speed of the system.